EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

Inconsistency resolving of safety and utility in access control

Jianfeng Lu[1*], Ruixuan Li[2], Jinwei Hu[3] and Dewu Xu[1]

Abstract

Policy inconsistencies may arise between safety and utility policies due to their opposite objectives. In this work we provide a formal examination of policy inconsistencies resolution for the coexistence of static separation-of-duty (SSoD) policies and strict availability (SA) policies. Firstly, we reduce the complexity of reasoning about policy inconsistencies by static pruning technique and minimal inconsistency cover set. Secondly, we present a systematic methodology for measuring safety loss and utility loss, and evaluate the safety-utility tradeoff for each choice. Thirdly, we present two prioritized-based resolutions to deal with policy inconsistencies based on safety-utility tradeoff. Finally, experiments show the effectiveness and efficiency of our approach.

Keywords: access control, safety, utility, separation-of-duty

1. Introduction

The safety and utility policies are very important in an access control system for ensuring security and availability when performing a certain task. Safety policies are used to describe safety requirements which ensure that users who should not have access do not get access. Such focus on safety requirements probably stems from the fact that safety policies have been mostly viewed as a tool for restricting access. An example of the safety policy is a *static separation-of-duty* (SSoD) policy, which precludes any group of users from possessing too many permissions [1]. An equally important aspect of access control is the utility policies that enables access [2,3]. In our previous work [4], we have introduced the notion of availability policies which is an example of an utility policy. In this paper, we introduce the notion of *strict availability* (SA) policies, which is also an example of utility policy that requires that the cooperation among at most a certain number of users is necessary to perform a task. Due to their opposite objectives, safety policies and utility policies can conflict with each other. For example, let p_1 and p_2 be two permissions, and u_1 and u_2 two users. Assume that an SSoD policy requires that neither u_1 nor u_2 possess all permissions in $\{p_1, p_2\}$. An SA policy requires both u_1 and u_2 possess all permissions in $\{p_1, p_2\}$. Clearly, the two policies cannot be satisfied simultaneously.

This paper examines this kind of conflict: *policy inconsistencies* that result from the incompatibility between safety policies and utility policies, especially for the coexistence of SSoD policies and SA policies. Policy inconsistencies differ from the traditional policy conflicts [5] in that the composition of safety and utility policies is never supposed to be inconsistent. That means policy inconsistencies are checked at compile-time to prevent the construction of any *safety* or *utility* policy that may conflict with each other. A policy inconsistency results in a policy compilation error. Hence, the resolution for policy inconsistencies is a policy design problem, whereas policy conflicts are resolved at runtime. In practice, the policy administrator may define many safety and utility policies and these policies may be inconsistent. However, it is not easy to detect and resolve these policy inconsistencies. Thus, it is very important to help the policy administrator to detect and resolve the policy inconsistencies at compile-time. The above discussion motivates the problem considered in this paper.

In our previous work [4], we have addressed the problem of consistency checking for the coexistence of safety and utility policies [4]. In this paper, we aim for providing a formal examination of policy inconsistency

* Correspondence: lujianfeng@zjnu.edu.cn
[1]College of Mathematics-Physical and Information Engineering, Zhejiang Normal University, Jinhua, Zhejiang, China
Full list of author information is available at the end of the article

resolution for safety and utility policies, which can help the policy administrators to specify reasonable access control policies when both safety and utility policies coexist. Our contributions are as follows:

- We formally define the policy inconsistency for the coexistence of safety policies and utility policies.
- We describe a static pruning technique that aims to reduce the number of policies that need to be taken into account.
- We compute the minimal inconsistency cover set that is responsible for the policy inconsistencies; thus we only need to examine the minimum number of policies.
- We present a systematic methodology for measuring safety loss and utility loss, and evaluate the safety-utility tradeoff for each candidate resolution.
- We present two prioritized-based resolutions to deal with policy inconsistencies for safety and utility policies based on safety-utility tradeoff.

The remainder of this paper is organized as follows. Section 2 formally defines the policy inconsistency problem for the coexistence of safety policies and utility policies. Section 3 presents prioritized-based resolutions for policy inconsistencies. 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. Policy inconsistency problem

We assume that there are two countably infinite sets in an access control state: U (the set of all possible users), and P (the set of all possible permissions). An access control state ε is a binary relation $UP \subseteq U \times P$, which determines the set of permissions a user possesses. Note that by assuming that an access control state ε is given by a binary relation $UP \subseteq U \times P$, 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.

Safety policies are used to describe safety requirements which ensure that users who should not have access do not get access. A safety policy is specified by giving a predicate on sets of executions. If conditions on (*users, permissions*) are satisfied, then a set U of users are prohibited from covering a set P of permissions. One example of a safety policy is an static separation-of-duty (SSoD) policy. SSoD policy is considered as a fundamental principle of information security that has been widely used in business, industry, and government applications [6]. An SSoD policy typically constrains the assignment of permissions to users, which precludes any group of users from possessing too many permissions. We first reproduce the definitions of SSoD policies from [4].

Definition 1. *An SSoD policy ensures that at least k users from a user set are required to perform a task that requires all these permissions. It is formally defined as*

- *P and U denote the set of permissions and the set of users, respectively.*
- $UP \subseteq U \times P$, *is a user-permission assignment relation.*
- $auth\_p_{\varepsilon}(u) = \{p | (p \in P) \wedge ((u, p) \in UP)\}$.
- $\forall (P, U, k) \in SSoD, \forall U' \subseteq U : |U'| < k \Rightarrow \cup_{u \in U'} auth\_p_{\varepsilon}(u) \nsupseteq P$.

where $P = \{p_1, ..., p_m\}$, $U = \{u_1, ..., u_n\}$, *each* p_i *in P is a permission,* u_j *in U is a user, and m, n, and k are integers, such that* $2 \le k \le min(m, n)$, *where min returns the smaller value of the two. We write an SSoD policy as ssod <P, U, k>. An access control state* ε *satisfies an SSoD policy e = ssod <P, U, k>, which is denoted by* $sat_e(\varepsilon)$. *And* $sat_E(\varepsilon)$ *represents* ε *satisfies a set E of SSoD policies.*

A utility policy is also specified by giving a predicate on sets of executions. If conditions on (*users, permissions*) are satisfied, then a set U of users are obligated to possess all the permissions in P. We now introduce the notion of *strict availability* (SA) policies, which is an example of utility policies that states properties about enabling access in access control. An SA policy requires that the cooperation among at most a certain number of users is necessary to perform a task.

Definition 2. *An strict availability* (*SA*) *policy ensures that all size-t subsets of U are required to complete a task that requires all these permissions in P. It is formally defined as*

- *P and U denote the set of permissions and the set of users, respectively.*
- $UP \subseteq U \times P$, *is a user-permission assignment relation.*
- $auth\_p_{\varepsilon}(u) = \{p | (p \in P) \wedge ((u, p) \in UP)\}$.
- $\forall (P, U, t) \in SA, \forall U' \subseteq U : |U'| = t \Rightarrow \cup_{u \in U'} auth\_p_{\varepsilon}(u) \supseteq P$.

Where $P = \{p_1, ..., p_m\}$, $U = \{u_1, ..., u_n\}$, *each* p_i *in P is a permission,* u_j *in U is a user, and m, n, and t are integers, such that* $1 \le t \le min(m, n)$, *where min returns the smaller value of the two, and the variable t in size-t is used to represent the cardinality of a set. We write an SA policy as sa <P, U, t>. An access control state* ε *satisfies an SA policy f = sa <P, U, t>, which is denoted by* $sat_f(\varepsilon)$. *And* $sat_F(\varepsilon)$ *represents* ε *satisfies a set F of SA policies.*

Definition 3. *UCP* (*the Utility Checking Problem*) *is defined as follows: Given an access control state* ε *and a set F of SA policies, determining whether* $sat_F(\varepsilon)$ *is true.*

Theorem 1. *UCP is in P.*

PROOF. Given an access control state ε and a set F of SA policies, if for each SA policy $f = sa(P, U, t)$ in F that $sat_f(\varepsilon)$ is true, then $sat_F(\varepsilon)$ is true. In the following, we prove that $sat_f(\varepsilon)$ is true if and only if each permission $p \in P$ is assigned to no less than $(|U| + 1 - t)$ users in the user set U, where $|U|$ represents the cardinality of set U.

For the "only if" part, $sat_f(\varepsilon)$ being true means that the users in each size-t subsets of U together possess all the permissions in P. Suppose, for the sake of contradiction, that $sat_f(\varepsilon)$ is true, and there exists a permission $p \in P$ that is only assigned to $(|U| - t)$ users in U. Then we can find a user set U' where $|U| = t$, and each users in U' do not possess p. Thus $sat_f(\varepsilon)$ is false, and this contradicts the assumption; therefore, each permission $p \in P$ must be assigned to no less than $(|U| + 1 - t)$ users in U.

For the "if" part, if each permission $p \in P$ is assigned to no less than $(|U|+ 1 - t)$ users in U, then the users in each size-t user set U' will together possess p. Thus all the permissions in p will be covered by each size-t user set. In other word, the users in each size-t user set together are authorized for all permissions in P. Therefore, $sat_f(\varepsilon)$ is true.

Together with the above discussions, we now give a linear-time algorithm for determining whether $sat_F(\varepsilon)$ is true: For each SA policy $sa <P, U, t>$ in F, and for each permission $p \in P$. One first computes the set of all users the permission p is a member of, and compares this number with $(|U|+1-t)$. This algorithm has a time complexity of $O(N_U\ N_P\ M)$, where $N_U$is the number of users in U, $N_P$the number of permission in P, and M is the number of SA policies. □

An availability policy $ap<P, U, t>$ ensures that there exists a size-t subset of U that the users in this subset are required to possess all these permissions in P [4]. We now show that $sa<P, U, t>$ is at least as restrictive as $ap<P, U, t>$.

Definition 4. *Let $P_1$and P_2be two policies. We say that $P_1$is at least as restrictive as P_2(denoted by $P_1 \succcurlyeq P_2$) if $\forall\varepsilon(sat_{P_1}(\varepsilon) \Rightarrow sat_{P_2}(\varepsilon))$. When $P_1 \succcurlyeq p_2$but not $P_2 \succcurlyeq p_1$, we say that $P_1$is more restrictive than P_2(denoted by $P_1 \succ P_2$). And when $(P_1 \succcurlyeq p_2) \wedge (P_2 \succcurlyeq p_1)$, we say $P_1$and $P_2$are equivalent (denoted by $P_1 \triangleq P_2$).*

By definition, the $\succcurlyeq$ relation among all policies is a partial order. The $\succ$ relation among all policies is a quasi order.

Theorem 2. *Given an SA policy $f = sa<P, U, t>$, and an availability policy $g = ap<P, U, t>$, $f \succ g$ if and only if $|U| > t$.*

PROOF. For the "only if", We show that if $f \succ g$ then $|U| > t$. Suppose, for the sake of contradiction that $|U| \leq t$. By Definition 2, $t \leq |U|$, then $|U| = t$. For any access control state ε, if $sat_g(\varepsilon)$ is true, then $(\exists U' \subseteq U) \wedge (|U'| = t)[\cup_{u\in U'} auth\_p_\varepsilon(u) \supseteq P]$, and $U' = U$ as $|U| = t$. Then $\exists U' \subseteq U \wedge |U'| = t(\cup_{u\in U'} auth\_p_\varepsilon(u) \supseteq P)$ has the same meaning as $(\forall U' \subseteq U) \wedge (|U'| = t)\ (\cup_{u\in U'} auth\_p_\varepsilon(u) \supseteq P)$. That means $P_1 \triangleq P_2$, which contradicts the assumption. Therefore, if $f \succ g$ then $|U| > t$.

For the "if" part, if $|U| > t$ then $f \succ g$. By Definition 2, for each access control state ε that satisfies f if and only if the users in all size-t subsets of U together possess all the permissions in P, Let U' is a subset of U, that the users in U' together possess all of the permissions in P, and $|U'| = t$, then ε satisfies $ap<P, U, t>$. Therefore, $\forall\varepsilon(sat_f(\varepsilon) \Rightarrow sat_g(\varepsilon))$, and $f \succcurlyeq g$. We construct a new state ε' that satisfies g but does not satisfy f as follows: assign all permissions in P to only one user $u \in U$, but do not assign any permissions in P to any other users in U. Then we can find a user set $(U' \subset U) \wedge (|U'| = t)[\bigcup_{u\in U'} auth\_p'_\varepsilon(u) \supseteq P]$, and $sat_g(\varepsilon')$ is true. However, for any user set U'' that $(U'' \subset U) \wedge (|U''| = t) \wedge (u \notin U'')$, as $\bigcup_{u\in U''} auth\_p'_\varepsilon(u) \not\supseteq P$, $sat_f(\varepsilon')$ is false. Therefore, if $|U| > t$, then $f \succ g$. □

Intuitively, SA policies are a natural complement to SSoD policies in access control. Neither SA nor SSoD by itself is sufficient to capture both safety and utility requirements. Without the utility requirement, an access control state can satisfy any SSoD policy if the state does not contain any user set that covers all the permissions needed to accomplish the sensitive task. Similarly, without the safety requirement, any SA policy can be satisfied by giving all permissions to all users, which allows each single user be able to accomplish any task. In many cases, it is desirable for an access control system to have both SSoD and SA policies. However, these policies may conflict with each other due to their opposite objectives. Therefore, a formal description of policy inconsistency is necessary to detect and resolve it.

Definition 5. *CCP (the Consistency Checking Problem) is defined as follows: Given a set E of SSoD policies and a set F of SA policies, determining that whether there exists an access control state ε that $sat_E(\varepsilon) \wedge sat_F(\varepsilon)$ is true.*

Corollary 1. *CCP is coNP-complete.*

PROOF. That CCP is coNP-complete follows directly from the fact that the problem of determining whether $sat_E(\varepsilon)$ is true is coNP-complete (Theorem 1 in [4]), and the problem of determining whether $sat_F(\varepsilon)$ is true is in P (Theorem 1). □

Consider the following example of SSoD and SA policies. It is not easy to check whether the policies in the set Q is consistent.

Example 1. *Consider a set Q of SSoD and SA policies as follows.*

$$Q = \{e_1, e_2, f_1, f_2\}$$
$$e_1 = ssod\langle\{p_1, p_2, p_3\}, \{u_1, u_2, u_3\}, 2\rangle$$
$$e_2 = ssod\langle\{p_1, p_2\}, \{u_1, u_2\}, 2\rangle$$
$$f_1 = sa\langle\{p_1, p_2\}, \{u_1, u_2, u_3\}, 2\rangle$$
$$f_2 = sa\langle\{p_2, p_3\}, \{u_2, u_3\}, 1\rangle$$

We now show that the above SSoD and SA policies are inconsistent. Given any access control state ε, if $sat_{f_2}(\varepsilon)$ is true, that means p_2 and p_3 must be authorized to both u_2 and u_3. If $sat_{f_1}(\varepsilon)$ is true, then p_1 must be authorized to either u_2 or u_3. If u_2 possesses p_1, u_2 will possess all of the permissions in $\{p_1, p_2, p_3\}$, which violates both e_1 and e_2. If u_3 possesses p_1, u_3 will possess all of the permissions in $\{p_1, p_2, p_3\}$, which violates both e_1. Therefore, there does not exist an access control state ε that satisfies all of the four policies in Q.

In general, there may be many policy inconsistencies in a large access control policy set. Thus the following issues should be considered: (1) A large number of policy inconsistencies are possible, but many of them may be the result of a small number of policies that apply to aggregates. The key is to figure out the minimum number of policies that are responsible for the policy inconsistencies. (2) Once all the inconsistencies are known, we must determine the appropriate resolutions with little effort to resolve them, and estimate their impact on the policies. Like traditional policy conflict resolution, the theoretical resolution of policy inconsistencies is basically the same: remove some policies in the policy set. The primary difficulty is to determine which policies should be removed, and the resolution addresses the inconsistency most effectively.

3. Policy inconsistency resolution approaches

In this section, we provide a formal examination of policy inconsistencies resolution for the coexistence of SSoD and SA policies.

3.1. Reducing complexity

Once all the inconsistencies are known, we must find a way to resolve them. However, determining which policy to remove is difficult because there may be many policy inconsistencies. In order to simplify the resolution task, we consider as few policies as possible. Thus we reduce the complexity of reasoning about policy inconsistencies by the techniques of static pruning and minimal inconsistency cover set.

3.1.1. Static pruning

SSoD and SA policies can conflict with each other due to their opposite objectives. In general, not all SSoD or SA policies should be taken into account as they do not cause inconsistencies. The following theorem asserts that the special cases of SSoD(or SA) policies do not affect its compatibility with SA(or SSoD) policies. This enables us to remove them from our consideration. This greatly simplifies the problem.

Theorem 3. *Let* $Q = \{e_1, ..., e_m, f_1, ..., f_n\}$, *where* $e_i = ssod <P_i, U_i, k_i> (1 \leq i \leq m)$, $f_j = ap\langle P'_j, U'_j, t_j\rangle\ (1 \leq j \leq n)$. *If* $\exists e_i \in Q[(|P_i - R| > 0) \vee (|U_i \cap T| = 0)]$, *where* $R = \bigcup_{j=1}^{n} P'_j, T = \bigcup_{j=1}^{n} U'_j$, *then let Q' = Q' - {*$e_i$*}; If* $\exists f_j \in Q[(|U'_j \cap S| < t_j) \vee (|P'_j \cap W| = 0)]$, *where* $S = \bigcup_{i=1}^{m} U_i$, $W = \bigcup_{i=1}^{m} P_i$, *then let Q' = Q' - {*$f_j$*}. Q is consistent if and only if Q'is consistent.*

PROOF. For the "only if" part, it is clear that if Q is consistent then Q' is consistent as $Q' \subseteq Q$.

For the "if" part, we show that if Q' is consistent then Q is consistent. Q' is consistent implies that there exists an access control state ε satisfies all policies in Q'. We now construct a new state ε' that satisfies both Q' and Q as follows: for each $e_i \in Q/Q'$, where $|P_i - R| > 0$. Add all users in $U_i$to ε, but do not assign any permissions in $P_i \cap R$. In this way, ε' satisfies $e_i$as no less than $k_i$users in $U_i$together having all permissions in P_i, and note that adding new users will not lead to inconsistency of policies in Q'. If $|U_i \cap T| = 0$, not assigning any permission in $P_i$to any user in $U_i$will not lead to inconsistency of policies in Q', but the new state satisfies e_i. For each $f_j \in Q/Q'$, where $|U'_j \cap S| < t_j$, add all users in U'_j to ε, and assign all permissions in P'_j to each user in $U'_j \cap S$. Then there is at least one user $u \in U'_j \big/ S$ in each size-$t_j$user set in U'_j, as u has all the permissions in P'_j, thus each size-$t_j$user set in U'_j together having all the permissions in P'_j. In this way, ε' satisfies f_j, and note that adding new users, and assigning permissions to these new users will not lead to violation of policies in Q'. If $|P'_j \cap W| = 0$, assigning any permissions in P'_j to each user in U'_j will not lead to inconsistency of policies in Q', and thus the new state ε' satisfies f_j. Therefore, Q is consistent if and only if Q' is consistent. □

3.1.2. Minimal inconsistency cover set

There may exist many policy inconsistencies in a policy set which contains a large number of SSoD and SA policies. But many of these inconsistencies may result from only a small number of these policies, and they may be disjoint with each other. We find the *minimal inconsistency cover* set is the minimal number of policies that represent a policy inconsistency. Therefore, the key question is how to organize the policy inconsistencies, so as to examine the minimum number of policies that are responsible for all the inconsistencies.

Definition 6. *We define a minimal inconsistency cover (MIC) set responsible for a policy inconsistency that includes the smallest number of policies.*

Note that for a policy inconsistency, there might be several policy sets that are responsible for this inconsistency. By definition, we say that a set S is an MIC set, if there does not exist another set S' responsible for this inconsistency and $S' \subset S$. We have the following property for MIC.

Theorem 4. *Given any two MIC sets A and B, let P_A denotes the union of permissions in all policies in A, and U_A denotes the union of users in all policies in A. P_B and U_B have the similar meanings. Then $(P_A \cap P_B = \emptyset) \vee (U_A \cap U_B = \emptyset)$.*

PROOF. We assume that $(P_A \cap P_B = \emptyset) \vee (U_A \cap U_B = \emptyset)$ is false, then $(P_A \cap P_B \neq \emptyset) \wedge (U_A \cap U_B \neq \emptyset)$. There are four cases should be considered:

(1) Permissions and users for $\{e_1, ..., e_m\} \subseteq A (m \geq 1)$ and $\{e'_1, \ldots, e'_n\} \subseteq B(n \geq 1)$ are shared;
(2) Permissions and users for $\{e_1, ..., e_m\} \subseteq A (m \geq 1)$ and $\{f_1, ..., f_n\} \subseteq B (n \geq 1)$ are shared;
(3) Permissions and users for $\{f_1, ..., f_m\} \subseteq A (m \geq 1)$ and $\{f'_1, \ldots, f'_n\} \subseteq B(n \geq 1)$ are shared;
(4) Permissions and users for $\{e_1, ..., e_m, f_1, ..., f_n\} \subseteq A$ $(m \geq 1, n \geq 1)$ and $\{e'_1, \ldots, e'_l, f'_1, \ldots, f'_k\} \subseteq B(l \geq 1, k \geq 1)$ are shared.

For case (1), there exists at least one permission $p \in P_{\{e_1,\ldots,e_m\}}$, but p does not belong to any other policies in A. By Theorem 3, $\{e_1, ..., e_m\}$ does not affect the inconsistency of other permissions in A, and thus $\{e_1, ..., e_m\}$ can be removed from A. This would contradict the assertion that A is an MIC set. Moreover, there exists at least one permission $p \in P_{\{e'_1,\ldots,e'_n\}}$, but p does not belong to any other policies in B. Thus $\{e'_1, \ldots, e'_n\}$ also can be removed from B. For case (2) and case (3), the proof is essentially the same as the case (1). It should be noted that there exists at least one user u belongs to the policies in $\{f_1, ..., f_n\}$, but u does not belong to any other policies in B. Thus $\{f_1, ..., f_n\}$ should be removed from B by Theorem 3. For case (4), no policies can be removed from $\{e_1, \ldots, e_m, f_1, \ldots, f_n\} \cup \{e'_1, \ldots, e'_l, f'_1, \ldots, f'_k\}$, which means these policies may conflict with each other due to their opposite objectives. Therefore, these policies should be included by only one MIC set. This would contradict the assertion that A and B are two MIC sets. Together with the above discussions, given any two MIC sets, that $(P_A \cap P_B = \emptyset) \vee (U_A \cap U_B = \emptyset)$. □

We now give an algorithm to generate the MIC sets for an access control policy set. Algorithm 1 includes an underlying presumption that all SSoD and SA policies which do not cause policy inconsistencies have been removed from our consideration by using "static pruning" technique. Given a policy set Q, the algorithm first divides Q into several subsets by the step 1 to 20. By the step 21 to 27, the algorithm combines the different sets which share the permissions and users. This algorithm has a worst-case time complexity of *O(mnMN)*, where m is the number of SSoD policies, n is the number of SA policies, M is the number of users, N is the number of permissions. The fact that CCP is intractable (coNP-complete) means that there exist difficult problem instances that take exponential time in the worst case, while efficient algorithms for CCP exist when the number of policies is not too large. MIC helps to reduce the complexity of reasoning about policy inconsistencies.

Example 2. *Continuing from Example 1, we add four policies $\{e_3, e_4, f_3, f_4\}$ to* Q, *Consider the combination of following SSoD and SA policies.*

$$Q' = \{e_1, e_2, e_3, e_4, f_1, f_2, f_3, f_4\}$$
$$e_1 = ssod\langle\{p_1, p_2, p_3\}, \{u_1, u_2, u_3\}, 2\rangle$$
$$e_2 = ssod\langle\{p_1, p_2\}, \{u_1, u_2\}, 2\rangle$$
$$e_3 = ssod\langle\{p_4, p_5\}, \{u_4, u_5\}, 2\rangle$$
$$e_4 = ssod\langle\{p_4, p_5, p_6\}, \{u_4, u_5, u_6\}, 2\rangle$$
$$f_1 = sa\langle\{p_1, p_2\}, \{u_1, u_2, u_3\}, 2\rangle$$
$$f_2 = sa\langle\{p_2, p_3\}, \{u_2, u_3\}, 1\rangle$$
$$f_3 = sa\langle\{p_5, p_6\}, \{u_4, u_6\}, 1\rangle$$
$$f_4 = sa\langle\{p_4, p_5, p_6\}, \{u_4, u_6\}, 2\rangle$$

By Theorem 3, no policy can be removed from our consideration by static pruning. But the permissions in $\{p_4, p_5, p_6\}$ and the users in $\{u_4, u_5, u_6\}$ only exist in $\{e_3, e_4, f_3, f_4\}$, and the policies in $\{e_3, e_4, f_3, f_4\}$ do not affect the consistency of $\{e_1, e_2, f_1, f_2\}$. By Algorithm 1, Q' can be divided into two policy set $Q'_1 = \{e_1, e_2, f_1, f_2\}$, and $Q'_2 = \{e_3, e_4, f_3, f_4\}$, such that each set is an MIC set. As shown in Example 1, the policies in Q'_1 are inconsistent. It is easy to find that the policies in Q'_2 are inconsistent, too. Continuing from Example 2, assume that there exist another two policies $e_5 = ssod <p_1, p_2, p_4, p_5, p_6\}, \{u_1, u_2, u_3, u_4, u_5, u_6\}, 3>$ and $f_5 = sa <\{p_1, p_2, p_3, p_4, p_5, p_6\}, \{u_1, u_2, u_4, u_6\}, 3>$, then the whole policies in $\{e_1, e_2, e_3, e_4, e_5, f_1, f_2, f_3, f_4, f_5\}$ is only one MIC set.

3.2. Measuring the safety-utility tradeoff

Given an MIC set for a policy inconsistency. Often, there may exist many choices for resolving this inconsistency. An interesting question for them is "which choice is optimal?". Our methodology helps policy administrators answer this question.

```
Algorithm 1. ComputeMIC (Q)
Input: $Q = \{e_1, ..., e_m, f_1, ..., f_n\}$
Output: the MIC sets of $Q : S_1, ..., S_x$
1: initialize $S_1 = \emptyset$, $i = 1$, $j = 1$, $k = 1$;
2: while $(i < m || j < n)$ do
```

```
3:  if $((P_{e_i} \cap P_{S_k} \neq \emptyset) \wedge (U_{e_i} \cap U_{S_k} \neq \emptyset))$then
4:     $S_k = S_k \cup e_i$;
5:     $i++$;
6:  else
7:     $k++$;
8:     continue;
9:  end if
10:      $k = 1$;
11:   if $((P_{f_j} \cap P_{S_k} \neq \emptyset) \wedge (U_{f_j} \cap U_{S_k} \neq \emptyset))$then
12:      $S_k = S_k \cup f_j$;
13:      $j++$;
14:   else
15:      k++;
16:      continue;
17:   end if
18:   $k = 1$;
19: end while;
20: $MIC(Q) \leftarrow S_1, ..., S_x$;
21: for $S_k \in MIC(Q)$ do
22:   if $\exists S_t \in MIC(Q)[(P_{S_t} \cap P_{S_k} \neq \emptyset) \wedge (U_{S_t} \cap U_{S_k} \neq \emptyset)]$then
23:      $MIC(Q) = MIC(Q) - S_t - S_k$;
24:      $S_k = S_k \cup S_t$;
25:      $MIC(Q) \leftarrow S_k$;
26:   end if
27: end for
28: return $MIC(Q)$.
```

Example 3. *Let us consider the same policies as the one from Example 1. After removing some policies from Q, the rest of policies will be consistent with each other. For example, resolving the policy inconsistency has the following choices.*

- *Removing only one policy:*$\{e_1\}$, $\{f_1\}$, *or* $\{f_2\}$.
- *Removing two policies:*$\{e_1, e_2\}$, $\{e_1, f_1\}$, $\{e_1, f_2\}$, $\{e_2, f_1\}$, $\{e_2, f_2\}$, *or* $\{f_1, f_2\}$.
- *Removing three policies:*$\{e_1, e_2, f_1\}$, $\{e_1, e_2, f_2\}$, $\{e_1, f_1, f_2\}$, *or* $\{e_2, f_1, f_2\}$.

Currently we lack a method for measuring the safety-utility tradeoff in policy inconsistency resolving. Removing SSoD policies result in *safety loss* for the whole safety requirement in Q. Similarly, Removing SA policies result in *utility loss* for the whole utility requirement in Q. Hence before making the choice, one must ensure that the *safety loss* and *utility loss* are limited to an acceptable level. To use our method, one must choose a measure for safety loss (S_{loss}) and utility loss (U_{loss}).

Definition 7. *Let $e_1$and $e_2$be two SSoD policies, we say that $S_{loss}^{e_1} \geq S_{loss}^{e_2}$ if and only if $e_1 \succcurlyeq e_2$. And $S_{loss}^{e_1} > S_{loss}^{e_2}$if and only if $e_1 \succ e_2$.*

Where $S_{loss}^{e_1}$ denotes the safety loss caused by removing e_1. As is intuitive, choosing to remove the policy with higher restrictive will cause more safety (or utility) loss.

Theorem 5. *For any SSoD policies e_1 = ssod <P_1, U_1, k_1>and e_2 = ssod <P_2, U_2, k_2>, $e_1 \succ e_2$if and only if* $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + | P_1 - P_2|)$.

PROOF. For the "if" part, given $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + | P_1 - P_2|)$, we show that $\forall \varepsilon(\neg sat_{e_2}(\varepsilon) \Rightarrow \neg sat_{e_1}(\varepsilon))$. There are two cases for $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + | P_1 - P_2|)$: (1) $P_1 \subseteq P_2$, (2) $P_1 \supset P_2$. $\neg sat_{e_2}(\varepsilon)$ being true means that there exist k_2-1 users in U_2 together having all the permissions in P_2. For case (1), there also exists k_2-1 users in U_1 together having all the permissions in P_1 as $(P_1 \subseteq P_2) \wedge (U_1 \supseteq U_2)$, and $(k_1 \geq k_2 + |P_1 - P_2|) \Rightarrow (k_1 - 1) \geq (k_2 - 1)$. Therefore, there exists k_1-1 users in U_1 together having all the permissions in P_1, in other words, $\neg sat_{e_1}(\varepsilon)$ is true. For case (2), there also exist k_2-1 users in U_1 together having all the permissions in $P_1 \cup \{P_2 - P_1\}$ as $(U_1 \supseteq U_2)$. At most $|P_1 - P_2|$ users together having all the permissions in $\{P_2 - P_1\}$, and $(k_1 \geq k_2 + | P_1 - P_2|) \Rightarrow (k_2 - 1) \leq (k_1 - 1) - |P_1 - P_2|$. Thus there exists k_1-1 users in U_1 together having all the permissions in P_1, $sat_{e_1}(\varepsilon)$ is also false. Therefore, $\forall \varepsilon(\neg sat_{e_2}(\varepsilon) \Rightarrow \neg sat_{e_1}(\varepsilon))$ is true.

For the "only if" part, given $e_1 \succcurlyeq e_2$, we show that $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + |P_1 - P_2|)$ is true. Suppose, for the sake of contradiction, that $\neg((U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + |P_1 - P_2|))$ is true. In other words, both $U_1 \supseteq U_2$ and $k_1 \geq k_2 + |P_1 - P_2|$ are false. Let e_1 and e_2 are two SSoD policies, where e_1 = *ssod* <P_1, U_1, k_1>, e_2 = *ssod* <P_2, U_2, k_2>. If $U_1 \supseteq U_2$ is false, then $\exists u \in U_2/U_1$. Assuming that $sat_{e_1}(\varepsilon)$ is true, assign all the permissions in P_2 to u, and then $sat_{e_2}(\varepsilon)$ is false as k_2>1. Therefore, $U_1 \supseteq U_2$ is true. If $k_1 \geq k_2 + |P_1 - P_2|$ is false, then $k_1 < k_2 + |P_1 - P_2|$. If $P_1 \subseteq P_2$, then $k_1 < k_2 \Rightarrow k_1 \leq k_2 - 1$. $sat_{e_1}(\varepsilon)$ being true means that at least k_1 users in U_1 together having all the permissions in P_1. We assume that there exist k_1 users in U_1 together having all the permissions in P_1 in ε; then there exist k_2-1 users in U_2 together having all the permissions in P_2 as to ε (let $U_1 = U_2$, and these k_1 users also have all the permissions in $\{P_2 - P_1\}$), then $sat_{e_2}(\varepsilon)$ is false. If $P_1 \supset P_2$, let $k_1 < k_2 + |P_1 - P_2|$; given an access control state ε that $sat_{e_1}(\varepsilon)$ is true, for each permission in $\{P_2 - P_1\}$, assign it to $|P_1 - P_2|$ different users, and these users are not assigned any other permissions in P_1, and then k_1-$|P_1 - P_2|$ users together having all the permissions in P_1. Therefore, there exist less than k_2 users in U_2 together having all the permissions in P_2 (let $U_1 = U_2$), and therefore, $sat_{e_2}(\varepsilon)$ is false. This contradicts the assumption that $e_1 \succcurlyeq e_2$. Therefore, if $e_1 \succcurlyeq e_2$, then $(U_1 \supseteq U_2) \wedge (k_1 \geq k_2 + |P_1 - P_2|)$. □

Definition 8. *Let $f_1$and $f_2$be two SA policies, we say that $U_{loss}^{f_1} \geq U_{loss}^{f_2}$if and only if $f_1 \succcurlyeq f_2$. And $U_{loss}^{f_1} > U_{loss}^{f_2}$ if and only if $f_1 \succ f_2$.*

Theorem 6. *For any SA policies f_1 = sa <P_1, U_1, t_1>and f_2 = sa <P_2, U_2, t_2>, $f_1 \succcurlyeq f_2$if and only if ($P_1 \supseteq P_2$) ∧ ($U_1 \supseteq U_2$) ∧ ($t_1 \leq t_2$).*

PROOF. For the “if” part, given ($P_1 \supseteq P_2$) ∧ ($U_1 \supseteq U_2$) ∧ ($t_1 \leq t_2$), we show that $\forall \varepsilon(sat_{f_1}(\varepsilon) \Rightarrow sat_{f_2}(\varepsilon))$ is true. $sat_{f_1}(\varepsilon)$ being true means that any size-t_1 user set U_1' from U_1 together having all the permissions in P_1. Since ($P_1 \supseteq P_2$) ∧ ($U_1 \supseteq U_2$) ∧ ($t_1 \leq t_2$), for each $U_1' \subseteq U_2 \subseteq U_1$, $\bigcup_{u \in U_1'} auth\_p_\varepsilon(u) \supseteq P_1 \supseteq P_2$, and $|U_1'| = t_1 \leq t_2$. Therefore, $sat_{f_2}(\varepsilon)$ is also true.

For the “only if” part, given $f_1 \succcurlyeq f_2$, we show that ($P_1 \supseteq P_2$) ∧ ($U_1 \supseteq U_2$) ∧ ($t_1 \leq t_2$) is true. Suppose, for the sake of contradiction, that ¬($P_1 \supseteq P_2$) ∧ ($U_1 \supseteq U_2$) ∧ ($t_1 \leq t_2$) is true, thus ($P_1 \subset P_2$) ∨ ($U_1 \subset U_2$) ∨ ($t_1 > t_2$) is true, then $\exists P \in P_2/P_1$. Assuming that there exists an access control state ε, and $sat_{f_1}(\varepsilon)$ is true. Let P be not assigned to any user in U_2, that does not affect $sat_{f_1}(\varepsilon)$. But $sat_{f_2}(\varepsilon)$ is false, as no size-t_2 user set from U_2 can together cover P_2. Thus the assumption is false, and $P_1 \supseteq P_2$ is true.

If $U_1 \subset U_2$ is true, then $\exists u \in U_2/U_1$. We now can construct a state ε that makes $sat_{f_2}(\varepsilon)$ true, but $sat_{f_1}(\varepsilon)$ false. By Theorem 1, $sat_f(\varepsilon)$ being true means that each size-t user sets from U cover the permission set P. The above discussion shown that $P_1 \supseteq P_2$ is true, and let $t_1 = t_2$. As $|U_2| + 1 - t_2 > |U_1| + 1 - t_1$, $sat_{f_1}(\varepsilon)$ is true, which contradicts the assumption, and thus $U_1 \supseteq U_2$ is true.

If $t_1 > t_2$ is true, let $f_1' = sa\langle P_2, U_2, t_1\rangle$. As shown above, $f_1 \succcurlyeq f_1'$, such as for any state ε that $\neg sat_{f_1'}(\varepsilon) \Rightarrow \neg sat_{f_1}(\varepsilon)$. Thus we only need to construct a state ε that $sat_{f_2}(\varepsilon)$is true, but $sat_{f_1'}(\varepsilon)$ is false as follows. Find a size-t_1 user set $U' \subset U_2$, and partition P_2 into t_1 disjoint sets $v_1, \ldots, v_{t_1}$, such that the permissions in each set be assigned to each user in U', respectively. Without any one user in U' can not cover P_2. Since $t_1 > t_2$, we can find a size-t_2 user set $U'' \subset U'$ that the users in U'' do not together have all the permissions in P_2. In other words, $sat_{f_1'}(\varepsilon)$ is false, and $sat_{f_1}(\varepsilon)$ is also false. This contradicts the assumption, and thus $t_1 \leq t_2$ is true. Consequently, if $f_1 \succcurlyeq f_2$, then ($P_1 \supseteq P_2$) ∧ ($U_1 \supseteq U_2$) ∧ ($t_1 \leq t_2$). □

After computing the rank of $S_{loss}$for each SSoD policy and $U_{loss}$for each SA policy. A fundamental problem in inconsistency resolving is how to make the right tradeoff between safety and utility. However, it is inappropriate to directly compare safety with utility. The most important reason is that removing SSoD policies will increase the safety loss for the whole policies, but will not increase the utility gain. Similarly, removing SA policies will increase the utility loss for the whole policies, but will not increase the safety gain. For example, if we choose to remove {e_1, e_2} in Example 5, then S_{loss} = 100%, U_{loss} = 0%. And if we choose to remove {f_1, f_2}, then S_{loss} = 0%, U_{loss} = 100%.

If safety and utility cannot be directly compared, how should one consider them in a policy set for inconsistency resolution? For this, given a number of policy sets that are candidates for removing, for each of which we measure its safety loss $S_{loss}$and its utility loss U_{loss}. We can obtain a set of (S_{loss}, U_{loss}) pairs, one for each set. An ideal (but unachievable) choice will have the smallest $S_{loss}$and U_{loss}. For this, we need to be able to compare two different (S_{loss}, U_{loss}) pairs.

Definition 9. *Given two pairs (S_{loss}, U_{loss})$_1$, and (S_{loss}, U_{loss})$_2$, we define (S_{loss}, U_{loss})$_1$ ≤ (S_{loss}, U_{loss})$_2$if and only if $(S_{loss}^1 \leq S_{loss}^2) \wedge (U_{loss}^1 \leq U_{loss}^2)$. And ($S_{loss}$, U_{loss})$_1$ < (S_{loss}, U_{loss})$_2$if and only if $(S_{loss}^1 < S_{loss}^2) \wedge (U_{loss}^1 < U_{loss}^2)$.*

Definition 10. *Let A and B be two policy sets; removing A will caused (S_{loss}, U_{loss})$_A$, and removing B will caused (S_{loss}, U_{loss})$_B$. We say that the choice of removing A is at least as optimal as removing B (denoted by (S_{loss}, U_{loss})$_A$$\trianglerighteq$ (S_{loss}, U_{loss})$_B$) if (S_{loss}, U_{loss})$_A$≤ (S_{loss}, U_{loss})$_B$. And the the choice of removing A is better than removing B (denoted by (S_{loss}, U_{loss})$_A$$\triangleright$ (S_{loss}, U_{loss})$_B$) if (S_{loss}, U_{loss})$_A$ <(S_{loss}, U_{loss})$_B$.*

Example 4. *Let us consider the following policy sets from Example 3 that can be removed to resolve the policy inconsistency.* S_1 = {e_1}, S_2 = {f_1}, S_3 = {e_1, e_2}, S_4 = {f_1, f_2}, S_5 = {e_1, e_2, f_1}.

Obviously,
$(S_{loss}, U_{loss})_{S_1} < (S_{loss}, U_{loss})_{S_3} < (S_{loss}, U_{loss})_{S_5}$, and $(S_{loss}, U_{loss})_{S_2} < (S_{loss}, U_{loss})_{S_4} < (S_{loss}, U_{loss})_{S_5}$. Thus S_1 and S_2 are two ideal choices to resolve the policy inconsistency.

3.3. Prioritized-based resolution

The notion of priority is very important in the study of knowledge based systems, since inconsistencies have a better chance to be resolved. The following subsections present two prioritized-based approaches to deal with policy inconsistencies. We first present the possibilistic logic approach, which selects one consistent subbase. And we then give the lexicographical inference approach, which selects several maximally consistent subbases [7]. We assume that knowledge bases Ψ are prioritized. Prioritized knowledge bases have the form Ψ = $\Psi^E \cup \Psi^F$, where $\Psi^E = S_1^E \cup \cdots \cup S_m^E$, $\Psi^F = S_1^F \cup \cdots \cup S_n^F$, E and F denote all the SSoD and SA policies in the system, respectively. Formulas in S_i^E(or S_i^F) have the same level of priority and have higher priority than the ones in S_j^E(or S_j^F) where $j > i$. S_1^E (or S_1^F) contains the one which have the highest priority in Ψ, and S_m^E(or S_n^F) contains the one which have the lowest priority in Ψ.

3.3.1. Possibilistic logic approach

Possibilistic logic approach selects one suitable consistent prioritized sub-base of Ψ, whereas the other policies in complement set for the subbase of Ψ

Algorithm 2. *GeneratePoss*(Ψ)

Input: knowledge bases $\Psi = \Psi^E \cup \Psi^F$

Output: *Poss*(Ψ)

```
1: initialize $Poss(\Psi) = S_1^E \cup S_1^F$, $i = 1$, $j = 1$;
2: while ($i \le m \&\& j \le n$) do
3:    if $Poss(\Psi)$ is inconsistent then
4:       $Poss(\Psi) = Poss(\Psi) - S_i^E - S_j^F$;
5:       if $Poss(\Psi) \cup S_i^E$ is consistent then
6:          $Poss(\Psi) = Poss(\Psi) \cup S_i^E$;
7:          $i{+}{+}$;
8:       else
9:          for $e \in S_i^E$ do
10:            if $Poss(\Psi) \cup p$ is consistent then
11:               $Poss(\Psi) = Poss(\Psi) \cup p$;
12:            end if
13:         end for
14:      end if
15:      if $Poss(\Psi) \cup S_i^E$ is consistent then
16:         $Poss(\Psi) = Poss(\Psi) \cup S_j^F$;
17:         $j + +$;
18:      else
19:         for $f \in S_j^F$ do
20:            if $Poss(\Psi) \cup f$ is consistent then
21:               $Poss(\Psi) = Poss(\Psi) \cup f$;
22:            end if
23:         end for
24:      end if
25:   else
26:      $i{+}{+}$;
27:      $j\ {+}{+}$;
28:      $Poss(\Psi) = Poss(\Psi) \cup S_i^E \cup S_j^F$;
29:   end if
30: end while;
31: return $Poss(\Psi)$.
```

should be removed. We should extract a subbase $\phi(\Psi)$ from Ψ, which is made of the first x-important and consistent strata(levels): $\phi(\Psi) = S_1 \cup \ldots \cup S_x$, such that $S_1 \cup \ldots \cup S_x$is consistent, but $S_1 \cup \ldots \cup S_{x+1}$ is inconsistent.

Definition 11. *We define Poss*(Ψ) *as the set of the preferred consistent possibilistic subbase of* Ψ : *Poss*(Ψ) = {*A: A* $\subseteq \Psi$ *is consistent and* $\nexists B \subseteq \Psi$ *is consistent where* $B \supset A$}.

We now give an algorithm to compute the *Poss*(Ψ) for Ψ (shown in Algorithm 2). This algorithm iteratively adds the SSoD and SA policies with higher priority. Removal of the policies not in *Poss*(Ψ) is essential to satisfy the consistency for the other policies in Ψ. This algorithm has a best-case time complexity of *O(mn)*, and a worst-case time complexity of $O(mnM2^N)$, where$m$ is the number of SSoD policies, n is the number of SA policies, M is the number of users, and N is the number of permissions.

Example 5. *Consider the combination of following SSoD and SA policies.*

$$Q = \{e_1,\ e_2,\ f_1,\ f_2,\ f_3\}$$
$$e_1 = ssod\langle\{p_1, p_2, p_3\},\ \{u_1,\ u_2,\ u_3\},\ 2\rangle$$
$$e_2 = ssod\langle\{p_1, p_2\},\ \{u_1,\ u_2\},\ 2\rangle$$
$$f_1 = sa\langle\{p_1,\ p_2,\ p_3,\ p_4\},\ \{u_1,\ u_2,\ u_3,\ u_4\},\ 3\rangle$$
$$f_2 = sa\langle\{p_1,\ p_2,\ p_3\},\ \{u_1,\ u_2,\ u_3\},\ 3\rangle$$
$$f_3 = sa\langle\{p_1,\ p_2\},\ \{u_1,\ u_2\},\ 1\rangle$$

By Theorems 5 and 6, we can find that $e_1 \succ e_2, f_1 \succ f_2$. Thus $\Psi = \Psi^E \cup \Psi^F$, where $\Psi^E = S_1^E \cup S_2^E$, $\Psi^F = S_1^F \cup S_2^F$, $S_2^E = \{e_2\}$, $S_2^E = \{e_2\}$, $S_1^F = \{f_1\}$, $S_2^F = \{f_2,\ f_3\}$. By Algorithm 2, $Poss(\Psi) = S_1^E \cup S_1^F \cup S_2^E \cup \{f_2\} = \{e_1,\ e_2,\ f_1,\ f_2\}$. Therefore, the removal of f_3 is an optimal choice to resolve the policy inconsistency.

3.3.2. Lexicographical inference approach

The possibilistic way of dealing with inconsistency is not entirely satisfactory since it only considers the first x-important consistent formulas having the highest priority. However, the less certain formulas may be not responsible for inconsistencies that should also be taken into account. The idea of lexicographical inference approach is to select not only one consistent subbase but several maximally consistent subbases. Obviously, the lexicographical inference is more expensive than the possibilistic logic.

Definition 12. *A consistent subbase* $A \subseteq \Psi$ *is said to be lexicographically preferred to a consistent subbase B* $\subseteq \Psi$, *denoted by* $A \rhd_{lex} B$, *if there exists a level* $i(1 \le i \le m)$ *and* $j(1 \le j \le n)$ *such that:*

$$(|A \cap S_i^E| > |B \cap S_i^E|) \wedge (\forall x \in [1,\ i),\ |A \cap S_x^E| = |B \cap S_x^E|) \wedge (|A \cap S_j^E| > |B \cap S_j^E|) \wedge (\forall x \in [1,\ j),\ |A \cap S_y^E| = |B \cap S_y^E|.$$

Definition 13. *We define Lex*(Ψ) *as the set of all preferred consistent lexicographical subbases of* Ψ : *Lex*(Ψ) = {*A:* $A \subseteq \Psi$ *is consistent and* $\nexists B \subseteq \Psi$ *is consistent,* $B \rhd_{lex} A$}.

We now give an algorithm to generate *Lex*(Ψ) that covers all preferred consistent possibilistic subbases of Ψ. The algorithm is similar to Algorithm 2, but we add following improvements as follows. Given the knowledge bases $\Psi = \Psi^E \cup \Psi^F$: if $Poss(\Psi) \cup S_i^E$ or $Poss(\Psi) \cup S_j^F$ is inconsistent, the algorithm does not stop (While in Algorithm 2, any policies in S_k^E, S_l^F will not be considered, where $k > i$, $l > j$), by repeatedly adding policies in S_k^E and S_l^F to *Poss*(Ψ). In the enumeration approach, the algorithm tries all possibilities. Eventually, the algorithm outputs all preferred consistent possibilistic subbases of

Ψ, such as $Lex(\Psi)$. In Example 4. There exists two lexicographically consistent subbases that $A = \{e_1, e_2, f_1, f_2\}$, $B = \{e_1, f_1, f_2, f_3\}$, then $Lex(\Psi) = \{A, B\}$.

4. Illustration and evaluation

Given the results shown in Section 3, we define the following approach to policy inconsistencies resolution.

1. Removing SSoD and SA policies from our consideration which do not cause inconsistencies by static pruning.
2. Generating MIC sets.
3. Consistency checking for each MIC set.
4. Extracting priorities based on safety-utility tradeoff.
5. Employing possibilistic logic (or lexicographical inference)approach

4.1. Running example

We now give a running example to show the validity of our approach for policy inconsistency resolving.

Example 6. *Considering the task of ordering and paying for goods given by Clark and Wilson* [6], *there are four steps:* (1) *ordering the goods and recording the details of the order;* (2) *recording the arrival of the invoice and verifying that the details on the invoice match the details on the order;* (3) *verifying that the goods have been received and the features of the goods match the details on the invoice;* (4) *authorizing payment to the supplier against the invoice. We add another two steps:* (5) *checking the status of the task, and* (6) *commenting on the task. We have a permission corresponding to each step in the task. The permission set is* {*order, goods, invoice, payment, check, comment*}. *Assuming that there are eight users* {*alice, bob, carl, doris, eric, fox, harris, george*} *who prepare to accomplish this task. The policy administrator may define many policies that require safety and utility properties in this example and these policies may be inconsistent. Thus it is very important to help the policy administrators to specify reasonable access control policies without inconsistencies. Assuming that the policy administrator defines the following policies.*

$$Q = \{e_1, e_2, e_3, e_4, e_5, f_1, f_2, f_3, f_4, f_5\}$$
$$e_1 = ssod\langle\{order, goods, invoice\}, \{alice, bob, carl\}, 2\rangle$$
$$e_2 = ssod\langle\{order, goods\}, \{alice, bob\}, 2\rangle$$
$$e_3 = ssod\rangle\{payment, check\}, \{doris, eric, fox\}, 2\rangle$$
$$e_4 = ssod\langle\{payment, check, comment\}, \{doris, eric, fox\}, 2\rangle$$
$$e_5 = ssod\langle\{payment, comment\}, \{doris, eric, fox\}, 2\rangle$$
$$f_1 = sa\langle\{order, goods, invoice, payment\}, \{alice, bob, carl, doris\}, 3\rangle$$
$$f_2 = sa\langle\{order, goods, invoice\}, \{alice, bob, carl\}, 3\rangle$$
$$f_3 = sa\langle\{order, goods\}, \{alice, bob, carl\}, 2\rangle$$
$$f_4 = sa\langle\{payment, check\}, \{doris, eric\}, 1\rangle$$
$$f_5 = sa\langle\{payment, check\}, \{doris, george\}, 2\rangle$$

We now implement the proposed approach to resolve the policy inconsistency problem in Q. Firstly, by Theorem 3, we find that e_4, e_5 and f_5 can be removed from our consideration. Let $Q' = \{e_1, e_2, e_3, f_1, f_2, f_3, f_4\}$, thus we only need to consider the policies in Q'. Secondly, by Algorithm 1, we can get two MIC sets: $\{e_1, e_2, f_1, f_2, f_3\}$ and $\{e_3, f_4\}$. Let $Q_A = \{e_1, e_2, f_1, f_2, f_3\}$, $Q_B = \{e_3, f_4\}$. Thirdly, we check whether the policies in each MIC set are consistent, and find that the policies in $Q_A$are inconsistent, but the policies in $Q_B$are consistent. Thus we only need to resolve the policy inconsistency in Q_A(Section 4.2 will give a more detailed description of consistency checking approach). Fourthly, we measure the safety loss for each SSoD policy and the utility loss for each SA policy. Via Theorem 5, we find that $e_1 \succ e_2, f_1 \succ f_2$. Thus we can have the form for prioritized knowledge bases $\Psi = \Psi^E \cup \Psi^F$(where $\Psi^E = S_1^E \cup S_2^E$, $S_1^E = \{e_1\}$, $S_1^E = \{e_1\}$, $S_2^E = \{e_2\}$, $S_1^F = \{f_1\}$, $S_2^F = \{f_2, f_3\}$.). We give the method for computing the $S_{loss}$and $U_{loss}$for each SSoD and SA policy, respectively as follows:

- $S_{loss}^{e} = \dfrac{rank(e)}{\Sigma_{\{e' \in \Psi^E\}} rank(e')}$
- $U_{loss}^{f} = \dfrac{rank(f)}{\Sigma_{\{f' \in \Psi^F\}} rank(f')}$

Let $rank(e_1) = 2$, $rank(e_2) = 1$, $rank(f_1) = 2$, and $rank(f_2) = rank(f_3) = 1$. Thus $S_{loss}^{e_2} \approx 33.3\%$, $S_{loss}^{e_2} \approx 33.3\%$, $U_{loss}^{f_1} = 50\%$, $U_{loss}^{f_2} = U_{loss}^{f_3} = 25\%$. Lastly, we employ Algorithm 2 to generate possibilistic logic subbase $Poss(\Psi) = \{e_1, e_2, f_1, f_2\}$, and compute its safety-utility pair $(S_{loss}, U_{loss})_{Poss(\Psi)} = (0, 25\%)$. We also generate $Lex(\Psi)$ and find that there exist two lexicographically consistent subbases that $Lex(\Psi) = \{Q_1, Q_2\}$, where $Q_1 = \{e_1, e_2, f_1, f_2\}$, and $Q_2 = \{e_2, f_1, f_2, f_3\}$. $(S_{loss}, U_{loss})_{Q_1} = (0, 25\%)$, $(S_{loss}, U_{loss})_{Q_2} = (66.7\%, 0\%)$.

The results above can help the policy administrator to resolve the policy inconsistency by removing some policies, and can specify reasonable access control policies. For example, if the safety requirement is more critical than the utility requirement in this running example, the policy administrator can choose to remove f_3, as it causes no safety loss, but 25% utility loss. Otherwise, he can choose to remove e_1 where it causes about 66.7% safety loss, but no utility loss.

4.2. Performance evaluation

In order to understand the effectiveness of our approach, we have implemented two algorithms, and performed several experiments using the running example as shown in Section 4.1. One is called improved algorithm based on our approach as discussed in above sections (employ the possibilistic logic approach), whereas the other is called straightforward algorithm discussed based on consistency checking problem [4]. The implementation of these two algorithms was written in Java. Experiments were carried out on a machine with an Intel(R) Core(TM)2 Duo CPU T5750 running at 2.0 GHz, and with DDR2 2 GB 667 Mhz RAM, running Microsoft Windows XP Professional.

Straightforward algorithm

Each time a new SSoD (or SA) policy is generated by a policy administrator, the algorithm determines whether this policy is consistent with already existing policies. If the answer to the consistency checking problem is "yes", then the new SSoD (or SA) policy is allowed to be added to the access control system. Otherwise, it will be disallowed. Finally, the generated policies are consistent. We also add the following improvements that greatly reduce the running time.

(1) Removing SSoD and SA policies from our consideration which do not cause policy inconsistencies using "static pruning" technique.

(2) Reducing the number of access control states that need to be considered. Given an access control state ε, for each SA policy $f = sa\langle P, U, t\rangle$, ε satisfies f if and only if for each size-t set of users from U such that these users together possessing all permissions in P. One only needs to compute the set of permissions of each size-t subsets of U, and check whether it is a superset of P. There exist $C_{|U|}^{t}$ size-t user sets for U. If the return for the algorithm is "no", then we know that the state ε does not satisfy f, and thus need not to be considered. By Lemma 1, for the sake of "least privilege" principle, in order to ensure $sat_f(\varepsilon)$ being true, we let each permission $p \in P$ be assigned to only $(|U| + 1 - t)$ users in U. This can greatly reduce the number of access control states that should be taken into consideration.

(3)Reduction to SAT: Given an SSoD policy $e = ssod\langle P, U, k\rangle$ and an access control state ε, we have shown that determining whether $sat_e(\varepsilon)$ is true is coNP-complete problem [8]. Thus we can use the algorithms for SAT to solve this problem. The SAT solver we use is SAT4J [9]. The translation works are as follows. Given an SSoD policy $e = ssod\langle P, U, k\rangle$ and an access control state ε, 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 cover all the permissions in P. Then we have the following two kinds of constraints. For each $p \in P$, let $u_{i_1}, u_{i_2}, \ldots, u_{i_x}$ be the users who are authorized for p. We add the first constraint $v_{i_1} + v_{i_2} + \cdots + v_{i_x} \geq 1$, which ensures that all the permissions in P are covered by U'. There are $|P|$ such constraints. Then we add the second constraint $v_1 + v_2 + \ldots + v_n \leq k - 1 (n = |U|)$, 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 $sat_e(\varepsilon)$ is false; otherwise, $sat_e(\varepsilon)$ is true.

We assume that the order of the policies generating as $e_1, f_1, e_2, f_2, e_3, f_3, e_4, f_4, e_5, f_5$. Some of our experimental results are presented in Table 1. As we can see in Table 1, the SSoD and SA policies should be considered for improved algorithm is only 5. However, each time a new SSoD (or SA) policy is added, it should check whether the new policy is consistent with already existing policies in the access control system, and the total number of policies need be considered for straightforward algorithm is 1341. And the number of access control states should be considered for improved algorithm is only 324. The runtime for straightforward algorithm is 1810.4 s, but only 178.2 s for improved algorithm. The results above show that our improved algorithm solves policy inconsistencies more efficiently than straightforward algorithm. As policy inconsistencies are checked at compile time, which is not expected to happen frequently, relative slow running time may be acceptable in some situations.

5. Related work

We examine related work in four categories: safety analysis, utility analysis, policy conflicts, and policy inconsistencies.

Safety analysis has been the main research area in access control for several decades. Harrison et al. [10] formalized a simple safety analysis that determining whether an access control system can reach a state in which an unsafe access is allowed in the context of the well-known access matrix model. Following that, there have been various efforts in designing access control systems in which simple safety analysis is decidable or efficiently decidable, e.g., Li et al. [2] generalized safety analysis in the context of a trust management framework. They also studied the safety analysis in the context of role-based access control (RBAC), where they gave a precise definition of a family of safety analysis

Table 1 Comparisons between straightforward algorithm (SA) and improved algorithm (IA)

Policies		e_1	f_1	e_2	f_2	e_3	f_3	e_4	f_4	e_5	f_5	Total
Policies	**SA**	0	0	0	3	3	4	3	5	8	9	34
	IA	0	0	0	0	0	0	0	0	0	5	5
States	**SA**	0	0	0	9	9	9	9	9	648	648	1341
	IA	0	0	0	0	0	0	0	0	0	324	324
Runtime	**SA**	0	0	0	3.5	4.3	5.0	3.8	8.1	829.4	956.3	1810.4
	IA	0	0	0	0	0	0	0	0	0	178.2	178.2

problems in RBAC. It is more general than safety analysis that is studied in the literature [6]. SoD policy has been considered as a fundamental principle of information security; the concept of SoD can be traced back to 1975 when Saltzer and Schroeder [11] 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. It has been recognized that "one of RBAC's great advantages is that SoD rules can be implemented in a natural and efficient way" [12]. Various frameworks have been developed for specifying SoD in the context of access control. However, it should be noted that most existing approaches on SoD only consider authorization constraint sets with exact two elements. We employ the definition for SoD by our previous work [8], which considers the total number of available users as a limitation factor through referring to the Jason's work [13]. In general, the problem of deciding whether a term is satisfied by a set of users is NP-complete [14]. Therefore, it comes as no surprise that directly enforcing SSoD policies is intractable (coNP-complete) [4]. Li et al. [15] seek to enforce an SSoD constraint using SMER(statically mutually exclusive roles) constraints, but provide no analysis of the complexity of computing the set of all such constraints. Chen et al. [16] study some variations on the set cover problem, and show that the RSSoD generation problem is NP-hard.

Safety policy is mostly viewed as a tool for restricting access. An equally important aspect of access control is to enable access. We introduce the notion of utility policies in this paper, which state properties about enabling access in access control. Li et al. introduces the related concept of availability policies in [2,6], which discriminates whether a user always possesses certain permissions across state changes. A similar concept is resiliency policy [3], which requires an access control system to be resilient to the absence of users. Following the preliminary version of this paper, Wang and Li [17] studied resiliency in workflow authorization systems. They proposed three levels of resiliency in workflow systems, namely, static resiliency, decremental resiliency and dynamic resiliency. Unlike the work by Li et al., the availability policy in [4] is a high-level requirement, and it is expressed in terms of restrictions on permission set and user set. AS shown in Theorem 2, SA policy is strict type of availability policy. Such policies are particularly useful when evaluating whether the access control configuration of a system is ready for emergency response. When an emergency such as a natural diaster or a terrorist attack occurs, an organization may need any teams of employees to respond to the emergency.

Policy-based authorization systems are becoming more common as information systems become larger and more complex. The overall authorization policy may be defined by different entities, which may produce conflicting authorization decisions. Arbitrary rules can be used to resolve Policy conflicts, but typically a generic resolution method is defined, such as first rule wins in firewalls or denials take precedence in ASL [18]. However, resolution of policy conflicts by manual intervention of policy administrator is a slow and ad hoc process and provides no guarantee on the optimality of the resulting interoperation system. Gong et al. [19] have investigated interoperation of systems employing multilevel access control policies. They have proposed several optimization techniques for resolution of interoperation conflicts. Ferrari and Thuraisingham have identified that several conflict resolution strategies may be useful depending on the domain [20]. In the current systems, rules and policy combination algorithms are defined on a static basis during policy composition, which is not desirable in dynamic systems with fast changing environments. Apurva Mohan et al. [21] propose a framework that supports the need for changing the rule and policy combination algorithms dynamically based on contextual information and also eliminates the need to recompose policies. The resolution for policy inconsistencies differs from policy conflicts that is resolved at compile-time. That means it is a static conflict resolution which is independent of access control system environments.

Policy inconsistencies may arise between safety and utility policies due to their opposite objectives. And in many cases, it is desirable for access control system to have both of safety and utility policies. Li et al. [4] attempts to address the problem of consistency checking for safety and availability in the context of access control. Based on the consistency checking method, it can help the policy administrator to specify reasonable access control policies without policy inconsistencies. However, this approach has its own shortcomings, the computing cost is usually unacceptable, and it does not consider optimization on tradeoff between safety and utility. In this paper, we provide a formal examination of policy inconsistencies resolution for safety and utility policies, especially for the coexistence of static separation-of-duty (SSoD) policies and strict availability (SA) policies. The experimental results show the validity of our approach. The resolution for policy inconsistencies is very important for policy administrators to specify reasonable access control policies when both safety and utility policies coexists.

6. Conclusion and future work

In this paper, we handled policy inconsistency of safety and utility policies based on the safety-utility tradeoff in the context of access control. We formally defined the policy inconsistency for the coexistence of safety policies and utility policies, and some key formal properties that resolved policy inconsistencies. We first reduced the complexity of reasoning about policy inconsistencies by *static pruning* and *MIC sets*; we then presented a systematic method for measuring safety loss and utility loss; Finally, we evaluated the safety-utility tradeoff, and presented two prioritized-based approaches to deal with policy inconsistencies. Our work can help the policy administrators to specify reasonable access control policies.

In the future research, we intend to address the policy inconsistencies by modifying policies rather than removing policies. It is difficult because there may be many choices, and to find the best choice is a challenging work. Continuing from Example 6, removing the SSoD policy $e_1 = ssod<\{order, goods, invoice\}, \{alice, bob, carl\}, 2>$, or the SA policy $f_3 = sa<\{order, goods\}, \{alice, bob, carl\}, 2>$ can both resolve the policy inconsistency. Assuming that we modify e_1 as $e_1' = ssod\langle\{order, goods, invoice\}, \{alice, bob\}, 2\rangle$, or modify f_3 as $f_3' = sa\langle\{order, goods\}, \{alice, bob\}, 2\rangle$. Then the policy inconsistency also can be resolved, and both of the safety loss and utility loss is lesser than removing e_1 or f_3.

Acknowledgements
This work is supported by National Natural Science Foundation of China under Grant 60873225, Zhejiang Province Education Foundation under Grant No.201120897.

Author details
[1]College of Mathematics-Physical and Information Engineering, Zhejiang Normal University, Jinhua, Zhejiang, China [2]College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei, China [3]Department of Computer Science, College of Engineering, Qatar University, Doha, Qatar

Competing interests
The authors declare that they have no competing interests.

Received: 14 November 2010 Accepted: 18 September 2011
Published: 18 September 2011

References

1. DD Clark, DR Wilson, A comparison of commercial and military computer security policies, in *Proceedings of 8th IEEE Symposium on Security and Privacy (SP)*, (IEEE Computer Society Press, Oakland, California, USA), pp. 184–195 (April 1987)
2. N Li, JC Mitchell, WH Winsborough, Beyond proof-of-compliance: security analysis in trust management. J ACM. **52**(3), 474–514 (2005). doi:10.1145/1066100.1066103
3. N Li, MV Tripunitara, Q Wang, Resiliency policies in access control. ACM Trans Inf Syst Secur. **12**(4), 113–137 (2009)
4. R Li, J Lu, Z Lu, X Ma, Consistency checking of safety and availability in access control. IEICE Trans Inf Syst Soc. **E93-D**(3), 491–502 (2010). doi:10.1587/transinf.E93.D.491
5. S Benferhat, R El Baida, F Cuppens, A stratification-based approach for handling conflicts in access control, in *Proceedings of the 8th Symposium on Access Control Models and Technologies*, (Villa Gallia, Como, Italy), pp. 189–195 (June 2003)
6. N Li, MV Tripunitara, Security analysis in role-based access control. ACM Trans Info Sys Secur. **9**(4), 391–420 (2006). doi:10.1145/1187441.1187442
7. D Dubois, J Lang, H Prade, Possibilistic logic, in *Handbook of Logic in Artifical Intelligence and Logic Programming*, vol. 3. (Oxford University Press, Oxford, 1994), pp. 439–513
8. J Lu, R Li, Z Lu, J Hu, X Ma, Specification and enforcement of static separation-of-duty policies in usage control, in *Proceeding 12th Information Security Conference (ISC)*, (Pisa, Italy), pp. 403–410 (September 2009)
9. D Le Berre, (project leader), SAT4J: A satisfiability library for Java. http://www.sat4j.org/ (January 2006)
10. MA Harrison, WL Ruzzo, JD Ullman, Protection in operating systems. Commun ACM. **19**(8), 461–471 (1976). doi:10.1145/360303.360333
11. JH Saltzer, MD Schroeder, The protection of information in computer systems. Proceed IEEE. **63**(9), 1278–1308 (2005)
12. R Sandhu, E Coyne, H Feinstein, C Youman, Role-based access control models. Computer. **29**(2), 38–47 (1996). doi:10.1109/2.485845
13. J Crampton, Specifying and enforcing constraints in role-based access control, in *Proceedings 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, (Villa Gallia, Como, Italy), pp. 43–50 (June 2003)
14. N Li, Q Wang, Beyond separation of duty: an algebra for specifying high-level security policies. J ACM. **55**(3), 1–46 (2008)
15. N Li, MV Tripunitara, Z Bizri, On mutually exclusive roles and separation-of-duty. ACM Trans Info Syst Secur. **10**(2), 231–272 (2007)
16. L Chen, J Crampton, Set covering problems in role-based access control, in *Proceedings of 14th European Symposium on Research in Computer Security*, (Saint-Malo, France), pp. 689–704 (September 2009)
17. Q Wang, N Li, Satisfiability and resiliency in workflow systems, in *Proceedings 12th European Symposium on Research in Computer Security*, (Dresden, Germany), pp. 90–105 (September 2007)
18. S Jajodia, P Samarati, VS Subrahmanian, A logical language for expressing authorizations, in *Proceedings of 18th IEEE Symposium on Security and Privacy (SP)*, (Oakland, California, USA), pp. 31–42 (May 1997)
19. L Gong, X Qian, Computational issues in secure interoperation. IEEE Trans Soft Eng. **22**(1), 14–23 (1996)
20. E Ferrari, B Thuraisingham, Secure database systems, in *Advanced Databases: Technology and Design*, ed. by Diaz O, Piattini M (Artech House, London, 2000)
21. A Mohan, DM Blough, An attribute-based authorization policy framework with dynamic conflict resolution, in *Proceedings of 9th Symposium on Identity and Trust on the Internet*, (New York, NY, USA), pp. 37–50 (2010)

doi:10.1186/1687-1499-2011-101
Cite this article as: Lu *et al.*: Inconsistency resolving of safety and utility in access control. *EURASIP Journal on Wireless Communications and Networking* 2011 **2011**:101.