Efficiently detect conflicts between attribute-based access control policy rules with rule reduction and binary-search techniques [*]

Cheng-chun Shu
Research Center for Grid and Service Computing
Institute of Comp. Tech.,CAS,China
No.6 Kexueyuan South Rd,Haidian, Beijing,100190
shuchengchun@software.ict.ac.cn

Erica Y. Yang and Alvaro Arenas
STFC e-Science Center
Rutherford Appleton Laboratory
Didcot,OX11 0QX,UK
erica.yang@stfc.ac.uk, a.e.arenas@stfc.ac.uk

Abstract

Attribute-based access control(ABAC) policies are effective and flexible in governing the access to information and resources in open computing environments. However, ABAC policy rules are often complex making them prone to conflicts. The complexity of dealing with ABAC rules arises from using multiple attributes to describe subjects and objects.

This paper proposes an optimized method to detect the conflicts between statistically conflicting rules *in an ABAC policy. This method includes two optimization techniques: rule reduction and binary-search. The first technique reduces the rules into a set of compact,* semantically equivalent rules *through removing redundant information among the rules. The binary-search technique is then applied to discover the conflicts among them. We also detail the algorithms used by these techniques to achieve the optimized performance. The time complexity for the proposed method is $O(nlgn)$, where n is the number of rules in a policy. This is achieved at a cost of less than two times runtime space increase. The experimental studies have shown that a) our method can detect statically-conflicting rules in near linear time cost proportional to the number of rules; and b) achieve good scalability, as shown, by efficiently detecting the conflicts in an ABAC policy containing over 20,000 rules.*

[*]This work was carried out for the EchoGRID IST project n^o04552,funded by the European Commission. The work of the first author was performed while visiting STFC e-Science Center, Rutherford Appleton Laboratory, UK.

1 Introduction

Access control policies provide secure and controlled service coordination and resource sharing in a variety of applications including database federation [9, 7], Grid systems [5, 12, 15, 18], Web services [21, 22]. Governed by the policies, access permissions are authorized only to legitimate users and any unprivileged requests are rejected. Recently attribute-based access control (ABAC) [4] policies are gaining popularity in open distributed environments such as Grids. ABAC policies govern user requests based on the characteristics of requestors and resources rather than their identies.

One important aspect of dealing with ABAC policies is to detect the conflicts among policy rules. Conflicting policy rules make different assertions for the same set of user requests. Without the conflicts being detected and resolved, the access control may be too open to guarantee the security of participating entities or be too restrcitive to benefit from the collaboration and sharing of information and resourcesThe conflicts between policy rules should be detected and eliminated before the rules are deployed at Policy Decision Points(PDPs) through conflict detection techniques. Some ABAC policies, such as XACML, use *rule combining algorithms*[14] at a rule level, to automatically resolve the conflicts among policy rules. However, the consequence of using these algorithms to override the access control decisions made by resource owners can be unpredictable when the number of rules being deployed at a PDP becomes too large to be comprehensable by resource administrators. This paper investigates the problem of automatically detect and remove the conflicts among ABAC policy rules and proposes techniques to efficiently resolve the problem.

Specifically, this paper focuses on *statically-conflicting* rules where conflicts among rules are detectable and can be removed without any evaluation being done against user requests. Informally, two *statically-conflicting* rules share the same set of subject and object attributes and there are intersections between their shared attribute predicates and action sets. However, they always give different decisions to a user request. An example of such is illustrated by rule 4 and 6 in Table 1.

Our contributions are sumarised as follows.

- We propose a novel method to detect and remove the conflicts among ABAC policy rules. This approach includes two techniques: rule reduction and binary search. Based on the notation of *semantically-equivalent policies*, rule reduction reduces the rules into a set of compact, semantically equivalent rules through detecting and removing the redundancy among policy rules. Upon the reduced rules, binary searching techniques can be applied to search conflicting policy rules.
- We prove that this method is optimized and show that the time complexity of the approach is $O(nlgn)$, where n is the number of rules in a policy, at cost of less than two times space increase in terms of the number of attribute predicates.
- Our experimental results have shown that the method can detect statically-conflicting rules in *nearly linear time* and is *scalable* to deal with policies with over 20,000 rules.

The remainder of this paper is organized as follows. Section 2 defines the policy model and its semantics. Section 3 formally defines conflicts between ABAC policy rules and *statically-conflicting* rules. Section 4 presents the optimized method of detecting statically-conflicting rules, and the implementation and analysis of the alogrithms. Experimental studies are presented in section 5, related work is discussed in section 6. Section 7 concludes the paper and outlines the future work.

2 ABAC Policy Model

The formal syntax and semantics of our policy model are based on those introduced by Bonatti [3] and Wimmer[21]. In the policy model, an ABAC policy comprises of rules, a rule contains four elements: subject,object, action and decision, and a decision specifies whether users (i.e., subjects) are allowed or denied (i.e., decisions) to perform actions over given resources (i.e., objects). The subject and object elements of a rule are defined as multiple attribute predicates, the action element as a set of action names that are allowed to be performed over objects, the decision element as one of the values *allow* or *deny*.

2.1 Model

The formal definitions of the ABAC policy model are given as follows.

Definition 2.1 (*Attribute predicate, multiple-attribute set)* . *An attribute predicate ap defines an attribute comparison of the form* (attribute-identifier $\circ$ constant). *The comparison operator* $\circ$ *is in* $\{<, \leq, =, >, \geq\}$. *A multiple-attribute set is represented by a disjunction of conjunctions of one or more attribute predicates in the form* $(ap_{1,1} \wedge ... \wedge ap_{1,m}) \vee ... \vee (ap_{k,1} \wedge ... \wedge ap_{k,m}))$, *where* $ap_{i,j}$ *is an attribute predicate.*

Definition 2.2 (*Rule and ABAC Policy)* . *A rule* $R = (S, O, A, d)$ *is a quadruple specifying that a set of actions A performed by a set of subjects S over a set of objects O is granted by decision d by the rule, where S and O are specified by a multiple-attribute set. A subject multiple-attribute set characterises a set of subjects S and is represented as* $(s_{1,1} \wedge ... \wedge s_{1,m}) \vee ... \vee (s_{k,1} \wedge ... \wedge s_{k,m})$, *where* $s_{i,j}$ *is an attribute predicate in the i-th conjunction that uses the j-th subject attribute-identifier. An object multiple-attribute set has a similar representation. A is a set of action names, denoted by* $a_1, a_2, ..., a_n$. *The decision d only takes two possible values: allow or deny, which represents a positive or a negative access decision respectively.*

A policy $P = \{R_1, R_2..., R_n\}$ *is made up of a set of rules* $R_1, R_2..., R_n$.

Compared with the policies specified with XACML, this policy model is a simplified one, which only supports five basic operations and attribute predicates containing attribute identifier and constant values.

2.2 Semantics

Access control policy systems guard the collaboration and sharing of information and resources by accepting or declining the user requests according to the decisions by the evaluation of policy rules. To evaluate

Table 1. Example of policy rules for an enterprise

#	Subjects	Objects	Actions	Decisions
1	dep=market,hr	fscope=maket,sales	write,read	Allow
2	dep=r&d	fscope=admin	write	Deny
3	dep=r&d,hr	fscope=market	read	Allow
4	$5 \leq workyear \leq 20$	fscope=special,norm	read	Allow
5	$workyear < 10$	fscope=norm	read	Allow
6	$workyear < 8$	fscope=special	write,read	Deny
7	$workyear > 10$	fscope=special	write,read	Allow

the policy rules, the evaluation context e is firstly extracted from the user request, which includes the subject and object sets of attribute-value pairs, and the action of the request.

Definition 2.3 (*Attribute Value Pair*) . *An attribute value pair $aid \circ val$ is a linked pair consisting of an attribute identifier, denoted by aid, and an alphanumerical value, denoted by val. The comparison operator $\circ$ is an element in the set of $\{<, \leq, =, >, \geq\}$*

Definition 2.4 (*Evaluation context*) . *An evaluation context e is defined as a triple $(S_{req}, O_{req}, a_{req})$, where S_{req} is a set of attribute-value pairs of a subject $(sa_1 = val_1), ..., (sa_p = val_p)$, O_{req} is a set of attribute-value pairs of an object $(oa_1 = oval_1), ..., (oa_q = val_q)$, and a_{req} is the action of a request.*

The subjects and objects in the evaluation context is modelled as sets of attribute-value pairs. Since most user requests have only one action and it is possible to decompose a complex request into multiple evaluation contexts, only a single action is modelled in the evaluation context.

All the rules in the ABAC policy are evaluated against with e. A rule is applicable to e if the subject's and object's attribute sets are within those specified by e and the requested action falls within the rule's action set of eThe set of decisions of applicable rules is returned as the result of rule evaluation.

Definition 2.5 (*Evaluation of Attribute Predicate(s)*) . *Given a given attribute-value pair $(aid = val)$, an attribute predicate ap is evaluated to be true if the attribute identifier of ap is same as aid, and val falls within the set specified by ap. This is represented as $\|ap\|_{aid=val} = true$. Otherwise, $\|ap\|_{aid=val} = false$. For a given set of attribute-value pairs $AV = \{(aid_1 = val_1), ..., (aid_p = val_p)\}$, the evaluation result of attribute predicate ap is true if there exists a attribute-value pair evaluates ap to true,i.e., $\|ap\|_{AV} = true \Leftrightarrow \exists(aid = val) \in AV, \|ap\|_{aid=val} = true$. Otherwise $\|ap\|_{AV} = false$.*

Given a set of attribute-value pair set $AV = (aid_1 = val_1, ..., aid_p = val_p)$, the evaluation result of a multiple-attribute set $S = ((s_{1,1} \wedge ... \wedge s_{1,m}) \vee ... \vee (s_{k,1} \wedge ... \wedge s_{k,m}))$ is defined as follows $\|S\|_{AV} = ((\|s_{1,1}\|_{AV} \wedge ... \wedge \|s_{1,m}\|_{AV}) \vee ... \vee (\|s_{k,1}\|_{AV} \wedge ... \wedge \|s_{k,m}\|_{AV}))$.

Given e, the evaluation result of a subject multiple-attribute set S, represented as $\|S\|_e$, is equal to $\|S\|_{S_{req}}$, i.e.,$\|S\|_e = \|S\|_{S_{req}}$. The evaluation result of a object multiple-attribute set O is given in similar way:$\|O\|_e = \|O\|_{O_{req}}$. The evalution result of an action set A, represented as $\|A\|_e$, is true if a_{req} is in A, i.e., $\|A\|_e = true \Leftrightarrow a_{req} \in A$

Definition 2.6 (*Applicable rule and Evaluation Result*) . *A rule $R = (S, O, A, d)$ is applicable in an evaluation context e if: $\|S\|_e \wedge \|O\|_e \wedge \|A\|_e = true$*

For a given evaluation context e, a policy $P = \{R_1, R_2..., R_n\}$ returns a set of decisions D as the evaluation results: $D = \|P\|_e = \{d | R_i = (S, O, A, d), \|S\|_e \wedge \|O\|_e \wedge \|A\|_e = true\}$.

Based on the above definitions, policies with different number of rules or rule specifications may always make the same set of decisions for any given evaluation contexts. These policies are interesting because a simplified policy may found to give the same evaluation results but more efficiently for an equivalent, complex policy. Therefore, it is possible to make an efficient analysis of a given policy. The polcies are called semantically equivalent policies. Based on the basic operations of multiple-attribute set, the formal definiation of semantically equivalent policies is given as follows

Definition 2.7 (*Operations of multiple-attribute set*) *A multiple-attribute set $S' = (ap'_1 \wedge ... \wedge ap'_p)$ is a subset of another multiple-attribute set $S = (ap_1 \wedge ... \wedge ap_q)$, denoted as $S' \subseteq S$ if the following conditions are satisfied:*

*(1)The two sets have the same set of attribute identifiers, i.e., $\forall ap'$ **in** S',$\exists ap$ **in** S such that $Aid(ap) = Aid(ap')$, and $\forall ap$ **in** S,$\exists ap'$ **in** S' such that $Aid(ap') = Aid(ap)$.*

(2)The value set of attribute predicate $ap^{'}$ in $S^{'}$ is a subset of the value set of attribute predicate ap in S with the same attribute identifier, i.e., $Aid(ap^{'}) = Aid(ap)$, $V_{ap'} \subseteq V_{ap}$.

A multiple-attribute set S_s is a subset of the intersection of two multiple-attribute sets S' and S if S_s is the subset of both sets S' and S,i.e., $S_s \subseteq S' \cap S \iff (S_s \subseteq S) \wedge (S_s \subseteq S')$.

A multiple-attribute set S_s is a subset of the union of two multiple-attribute sets S' and S if S_s is the subset of either set S' or set S,i.e., $S_s \subseteq S' \cup S \iff (S_s \subseteq S) \vee (S_s \subseteq S')$.

Theorem 2.8 *The multiple-attribute set S_s is a subset of multiple-attribute set S. For any evaluation context e such that $\|S_s\|_e = true$, then $\|S\|_e = true$.*

Definition 2.9 (*Semantically Equivalent policies) Two policies $P = \{R_1, R_2..., R_n\}$ and $P^{'} = \{R_1^{'}, R_2^{'}..., R_n^{'}\}$ are semantically equivalent if for a given tuple (S,O,A), any decision specified by one policy can also be found in another, i.e. $\forall(S,O,A,d), (\exists R_i = (S_i, O_i, A_i, d_i) \subseteq P, S_i \cap S \neq \phi \wedge O_i \cap O \neq \phi \wedge A_i \cap A \neq \phi \wedge d_i = d) \Leftrightarrow (\exists R_j = (S_j, O_j, A_j, d_j) \subseteq P^{'}, S_j \cap S \neq \phi \wedge O_j \cap O \neq \phi \wedge A_j \cap A \neq \phi \wedge d_j = d)$.*

Theorem 2.10 *If two policies $P = \{R_1, R_2..., R_n\}$ and $P^{'} = \{R_1^{'}, R_2^{'}..., R_n^{'}\}$ are semantically equivalent, then any evaluation context e can be evaluated to the same set of decisions,i.e. $\forall e, \|P\|_e = \|P'\|_e$.*

Proof 1 *We can prove $\|P\|_e = \|P'\|_e$ in two parts: $\forall d \in \|P\|_e \Rightarrow d \in \|P'\|_e$ and $\forall d \in \|P'\|_e \Rightarrow d \in \|P\|_e$. Because they are symmetry, we can prove only one of them. Let us prove the first part. For a given evaluation context $e = (S_{req}, O_{req}, a_{req})$, if $d \in \|P\|_e$, then $\exists R_i = (S_i, O_i, A_i, d_i)$ such that $\|S_i\|_e = \|O_i\|_e = \|A_i\|_e = true$. $\|S_i\|_e = true$ means $\forall ap_j$ in S_i, there exists an subject attribute-value pair $(sa_j = val_j) \in S_{req}$ such that $\|ap_j\|_{sa_j=val_j} = true$. Make a new subject multiple-attribute set S_r which is a conjunction of all attribute predicates from the attribute-value pairs, i.e., $S_r = ((sa_1 = val_1) \wedge ... \wedge (sa_p = val_p))$. Obviously $\|S_r\|_e = true$, thus $S_r \cap S_i \neq \phi$. Similiarly we can make a multiple-attribute set O_r such that $O_r \cap O_i \neq \phi$. $\|A_i\|_e = true$ means $a_{req} \in A_i$. Therefore the tuple $(S_r, O_r, \{a_r eq\}, d)$ is specified by $R_i \in P$, according to the definition there exists $R_j = (S_j, O_j, A_j, d) \in P'$ such that $S_r \cap S_j \neq \phi$,$O_r \cap O_j \neq \phi$ and $A_r \cap A_j \neq \phi$. Because S_r contains only attribute predicates with operator "=",$S_r \cap S_j = S_r$, which means $S_r \subseteq S_j$. From $\|S_r\|_e = true$, we have $\|S_j\|_e = true$. Similarly $\|O_j\|_e = true$, and $\|A_j\|_e = true$. Therefore rule R_j is applicable for the given evaluation context e and has the decision d, which implies $d \in \|P'\|_e$.*

Semantically, a given rule $R = (S, O, A, d), S = ((s_{1,1} \wedge ... \wedge s_{1,m}) \vee ... \vee (s_{k,1} \wedge ... \wedge s_{k,m})), O = ((o_{1,1} \wedge ... \wedge o_{1,n}) \vee ... \vee (o_{l,1} \wedge ... \wedge o_{l,n}))$ is equaivalent to a set of rules $\{R_{i,j} | R_{i,j} = (S_i, O_j, A, d), S_i = (s_{i,1} \wedge ... \wedge s_{i,m}), O_j = (o_{j,n} \wedge ... \wedge o_{j,n})\}$,$1 \leq i \leq k, 1 \leq j \leq n$. The conflicts among rules $R_{i,j}$ can be analyzed more easily than for the rule R, and the results of R can be synthesized by the individual results of rules $R_{i,j}$. Therefore, witout loss of generality, our discussion of conflicts is limited to the rules of form $R_{i,j}$, i.e., the multiple-attribute sets contain only conjunctions of attribute predicates for a single subject and a single object in the following sections. For the convenience of discussion, notations are summerized in table 2.

3 Conflicts between ABAC policy rules

An ABAC policy contains a set of rules, each of which specifies the positive or negative decision for the user requests. For the flexiblity of rule specification, different rules may be applicable to the same set of user requests. Therefore, the access control result of a given user request may depend on the decisions specified by multiple applicable rules. Problems arise if the decisions of the applicable rules are not same, i.e., for a given user request some of the rules assert positive while others assert negative. When the problem happens, the decision of the user request may be resolved automatically by the policy system according to the predefined mechnisms, e.g. the least priviledge principle or combination algorithms[14]. However, both resolution methods may make too rigid or too loose access control that introduces obstacles or threats to the collaboration and sharing of resources in distributed environment.

The problem that a single user request is applied to both positive (allow) and negative (deny) decisions by an ABAC policy is called a *conflict*. The root of a *conflict* is that some of the applicable rules for the given user request are *incompatible*. In this paper, we divide the relationships of two given ABAC policy rules into two categories: *compatible* and *incompatible*. For the same set of user requests(or evaluation contexts), *compatible* rules are policy rules that they give the same decisions or are never applicable at the same time, while *incompatible* rules may give distinct decisions and lead to conflicts. The formal definition of the conflicts of the rules are given as follow.

Definition 3.1 (*Conflicts between ABAC policy rules and conflicting rules) A conflict between rules R_i and*

Table 2. Notations

Symbols	Meaning
$S(R)$,$O(R)$,$A(R)$,$d(R)$	The subject,object multiple-attribute sets, action set and decision of the rule R
$(sa = val)$	an attribute-value pair with attribute identifier sa and value val
$Aid(ap)$	The attribute identifier of the given attribute predicate ap
ap in S	ap is one of the attribute predicates of multiple-attribute set $S = (ap_1 \wedge ... \wedge ap_p)$
$\lVert S\rVert_e$	Evaluate multiple-attribute set S by the evaluation context e
$\lVert ap\rVert_e$, $\lVert ap\rVert_{sa=val}$	Evaluate attribute predicate ap by the evaluation context e, attribute-value pair $(sa = val)$
V_{ap}	the value set of attribute predicate ap
$\{(sa_1 = val_1), ..., (sa_p = val_p)\}$	a set of p attribute-value pair

R_j of an ABAC policy P occurs if there exists an evaluation context e such that:

(1)The two rules are applicable for the given evaluation context,i.e., $\|S(R_i)\|_e \wedge \|O(R_i)\|_e \wedge \|A(R_i)\|_e = \|S(R_j)\|_e \wedge \|O(R_j)\|_e \wedge \|A(R_j)\|_e = true$,and

(2)the two rules' decisions are different,i.e.,$d(R_i) \neq d(R_j)$.

Two rules are conflicting if there is any conflict that occurs between the them.

3.1 Non-conflicting rules

Not any rules conflict. Given a ABAC policy $P = \{R_1, R_2..., R_n\}$, the relationships between policy rules, according to whether they conflict, can be divided into two categories: *non-conflicting* and *conflicting*. Two rules are *non-conflicting* if they never conflict, i.e., they never give distinct decisions for any evaluation contexts, while they are *conflicting* if they may lead to conflicts under some evaluation contexts. Two cases make two rules *non-conflicting* which breaks either of the two conflicting conditions: the rules either specify the same (positive or negative) decision or can not be applicable for any evaluation context at the same time. For example in Table 1 , rules 1 and 3 are non-conflicting because they all make positive ("allow") decision, rules 2 and 3 are so because their action elements do not overlap. The formal definition of non-conflicting rules is given as follows.

Definition 3.2 (*Non-conflicting rules*) *Two rules R_i and R_j are non-conflicting if either of the following conditions holds:*

(1)The rules' decisions are the same, i.e. $d(R_i) = d(R_j)$. Or

(2)The decisions are different, but no evaluation context e to which the two rules can be applicable at the same time, i.e., $d(R_i) \neq d(R_j)$, and $\neg\exists e$ such that $\|S(R_i)\|_e = \|S(R_j)\|_e = \|O(R_i)\|_e = \|O(R_j)\|_e = \|A(R_i)\|_e = \|A(R_j)\|_e = true$

Two rules R_i and R_j can not be applicable for any evaluation contexts e at the same time if they meet one of the following conditions:

(1)Their action sets do not overlap, i.e.,$A(R_i) \cap A(R_j) = \phi$. Or,

(2)The subject multiple-attribute sets of the two rules can never be applicable to any evaluation context at the same time. The condition can be satisfied when both the two rules have subject attribute predicates with a special attribute identifier and the attribute predicates have non-overlapped value sets. The attribute identifier sa is special that any evaluation context e contains only one attribute-value pair $sa = val$ with the identifier, i.e., $\forall(sa' = val') \in e, Aid(sa') = Aid(sa) \Leftrightarrow val = val'$. That attribute predicates ap_i in $S(R_i)$ and ap_j in $S(R_j)$ have the same attribute identifier and their value sets do not overlap is formally defined as $Aid(ap_i) = Aid(ap_j)$ and $V_{ap_i} \cap V_{ap_j} = \phi$. Or,

(3)The object multiple-attribute sets of the two rules can never be applicable to any evaluation context at the same time. The condition can be held when both the two rules have object attribute predicates with a special attribute identifier and the attribute predicates have non-overlapped value sets, as defined similiarly with previous condition.

Any evaluation context can not be applied by the rules without overlapped action sets, which can be reasoned as follows. Suppose there exists an evaluation context $e = (S_{req}, O_{req}, a_{req})$ such that $\|A(R_i)\|_e = \|A(R_j)\|_e = true$, then $a_{req} \in A(R_i)$ and $a_{req} \in A(R_j)$, which means the two action sets overlap because $a_{req} \in A(R_i) \cap A(R_j) \neq \phi$, contradicting that the two rules have the non-overlapped action sets.

Another condition that the rules can not be applicable for any evaluation context at the same time is examplified by rules 6 and 7 in table 1, where any evaluation context can only contain one attribute-value pair with attribute *workyear* and the attribute predicates (*workyear* < 8 and *workyear* > 10) in subject multiple-attribute sets have no overlaped value sets.

The reasoning can be given as follows: suppose there exists an evaluation context $e = (S_{req}, O_{req}, a_{req})$ is applied by the rules R_i and R_j satisfying such a condition, then $\|S(R_i)\|_e = \|S(R_j)\|_e = true$, which implies that there exists an attribute-value pair $(sa = val) \in S_{req}$ such that the attribute predicates ap_i and ap_j with the special attribute identifier sa in rules R_i and R_j have $\|ap_i\|_{sa=val} = \|ap_j\|_{sa=val} = true$. The evaluation of the attribute predicates to true also means that $val \in V_{ap_i} \cap V_{ap_j} \neq \phi$, which contradict the condition $V_{ap_i} \cap V_{ap_j} = \phi$.

3.2 Conflicting rules

Conflicting rules are those may give different decisions for some evaluation contexts and lead to conflicts. *Conflicting* rules not only have different decisions and overlapped action sets, but also their subject and object multiple-attribute sets can be evaluated to true by some evaluation contexts. The formal definition of *conflicting* rules are given in definition 3.1.

To eliminate conflicts in the ABAC policy, all the *conflicting* rules should be first found out and revised into *non-conflicting* ones or adjust them based on the predefined conflict resolution mechanisms. Particularly, it is desirable to detect and remove *conflicting* rules before they are put into enforcement. However, it is difficult and overkill to capture all the *conflicting* rules between ABAC policy rules before runtime. In fact, any two rules that do not determine to be non-conflicting may cause conflicts under some evaluation contexts, though their attribute predicates seem to be quite irrelevant. For example in table 1 rule 1 $(dep = market|hr, fscope = market|sales, write|read, allow)$ and rule 6 $(workyear < 8, fscope = special, write|read, deny)$ are applicable for the evaluation context $e = (\{(dep = market), (workyear = 6)\}, \{(fscope = special), fscope = market)\}, read)$ and lead to conflicting decisions, and therefore they are *conflicting*. The conflicts between the two rules are special that they occur only under a small set of special evaluation contexts and the rules are not impractical to be revised into *non-conflicting* ones because the revision may affect a majority of user requests that the rules are specified to goven on purpose.

A more interesting category of *conflicting* rules in an ABAC policy is those rules share attribute identifiers and the attribute predicates of two rules with same identifiers have intersected value sets. Especially, the rules called *statically-conflicting* that one rule shares all attribute identifiers with another rules, and the attribute predicates of shared identifiers have intersected value sets. *Statically-conflicting* rules are common in a set of policy rules and result in conflict more frequently than other *conflicting* rules. In addition, they are detectable and the conflicts caused the rules are removable before runtime. One example of *conflicting* rules is rule $(dep = hr \wedge workyear > 10, fscope = special, read\|write, allow)$ and rule $(workyear > 2, fscope, read, deny)$. The definition of *statically-conflicting* rules is formally given as follows:

Definition 3.3 (*Statically-conflicting rules) Two rules R_i and R_j are statically-conflicting if:*

(1)Their decisons are dinstinct,i.e.,$d(R_i) \neq d(R_j)$.

(2)Their action sets overlap, i.e., $A(R_i) \cap A(R_j) \neq \phi$.

(3)One of the rules shares all attribute identifiers with other rule. Without loss of generality, suppose rule R_i shares all attribute predicates with rule R_j. Formally the two rules satisfy $\forall ap$ in $S(R_i)$(or $O(R_i)$),$\exists ap'$ in $S(R_j)$(or $O(R_j)$) such that $Aid(ap) = Aid(ap')$.

(4)The attribute predicates with shared identifiers have intersected value sets, i.e., $\forall ap$ in $S(R_i)$(or $O(R_i)$) and $\forall ap'$ in $S(R_j)$(or $O(R_j)$), $Aid(ap) = Aid(ap') \Rightarrow V_{ap} \cap V_{ap'} \neq \phi$.

Theorem 3.4 *Statically-conflicting rules R_i and R_j cause conflicts.*

Proof 2 *To prove that the two rules cause conflicts, the evaluation contexts that can be applied by both the two rules should be found. Without loss of generality, suppose rule R_i shares all its attribute predicates with rule R_j.*

Firstly we make an evaluation context $e = (S_{req}, O_{req}, a_{req})$ satisfying the four following conditions:

(1)The requested action is in the intersection of the two rules' action sets, i.e. $a_{req} \in A(R_i) \cap A(R_j)$. and,

(2)For each of the shared subject attribute identifier sa, there exists a subject attribute-value pair $(sa = val)$ in S_{req} that evaluates both attribute predicates ap in $S(R_i)$ and ap' in $S(R_j)$ with the identifier sa to true, i.e., $\forall ap$ in $S(R_i)$ and ap' in $S(R_j)$ and $Aid(ap) = Aid(ap') = sa$, $\exists (sa = val) \in S_{req}$ such that $\|ap\|_{(sa=val)} = \|ap'\|_{(sa=val)} = true$. and

(3)For each of the shared object attribute identifier oa, there exists an object attribute-value pair $(oa = oval)$ in O_{req} that evaluates both attribute predicates oap in $O(R_i)$ and oap' in $O(R_j)$ with the identifier oa to true,i.e., $\forall oap$ in $O(R_i)$ and oap' in $O(R_j)$ and $Aid(oap) = Aid(oap') = oa$, $\exists (oa = oval) \in O_{req}$ such that $\|oap\|_{(oa=oval)} = \|oap'\|_{(oa=oval)} = true$. and,

(4)Other subject and object attribute predicates with no shared attribute identifiers are evaluated to true by the evaluation context e, i.e., $\forall ap''$ in $S(R_j)$(or $O(R_j)$), $\|ap''\|_e = true$.

Then both rule R_i and rule R_j can be applicable to such an evaluation context e: (a)Rule R_i is applicable for such an evaluation context e satisfying the above conditions because (1)$a_{req} \in A(R_i) \cap A(R_j) \Rightarrow a_{req} \in A(R_i) \Rightarrow \|A(R_i)\|_e = \|A(R_i)\|_{a_{req}} = true$. (2)For each subject attribute predicates ap, since $\exists(sa = val) \in S_{req}$ such that $\|ap\|_{(sa=val)} = true$, thus $\|S_{R_i}\|_e = \|S_{R_i}\|_{S_{req}} = true$ (3)For each object attribute predicates oap, since $\exists(oa = oval) \in O_{req}$ such that $\|oap\|_{(oa=oval)} = true$, thus $\|O_{R_i}\|_e = \|O_{R_i}\|_{O_{req}} = true$.

(b)Rule R_j is applicable to such an evaluation context e satisfying the above conditions because (1)$a_{req} \in A(R_i) \cap A(R_j) \Rightarrow a_{req} \in A(R_j) \Rightarrow \|A(R_j)\|_e = \|A(R_j)\|_{a_{req}} = true$.(2)Both rule R_j's subject element $S(R_j)$ and object elements $O(R_j)$ are evaluated to true by the evaluation context e because all its subject and object attribute predicates are evaluated to true by e: (i)The subject(or object) attribute predicates with shared attribute identifiers are evaluated to true since for each ap there exists an attribute-value pair $(sa = val)$ in S_{req}(or O_{req}) such that $\|ap\|_{(sa=val)} = true$;(ii)The subject(or object) attribute predicates ap without shared attribute identifiers are evaluated to true by the evaluation context e according the fourth condition of the evaluation context e.

Because $d(R_i) \neq d(R_j)$ and $\|R_i\|_e = \|R_j\|_e = true$, the two rules cause conflicts by one of such evaluation contexts e.

In practice, *statically-conflicting* rules are commonly seen in policy rules in the form that one rule with less attribute predicates governing the general case and other rules with more attribute predicates for the special cases. For example, rules $R_i = (workyear < 10, fscope = special, read, deny)$ and $R_j = (workyear < 10 \wedge dep = r\&d, fscope = special, read, allow)$ are *statically-conflicting* rules for the general case and the special cases respecitvely. Many conflicts caused by these *statically-conflict* rules are not real problems under predefined conflict resolution mechanisms, but analysis of the rules are worthy in order to verify the arrangement of rules are reasonable to be automatically resolved properly (E.g., put the special rules first under first-applicable resolution mechanisms).

In addition to *statically-conflicting* rules, there are other ABAC policy rules may result in conflicts, for example the conflicts occured between rule 1 and rule 6 in table 1. However the conflicts caused by *statically-conflicting* rules are special in two ways. Firstly, the root of conflicts is special. The conflicts of *statically-incompatible* rules are caused by the potential mistakes in rule specifications; the conflicts of other rules are caused by the special evaluation contexts with attribute-value pairs satisfying multiple rules by accident. Secondly, the conflicts of *statically-incompatible* rules are detectable before runtime by analyzing the relationship between rules, and the real conflicts can be removed by revising the elements (e.g., the subject, object, action and decision) of rules; the conflicts of other rules are detectable only when the special evaluation contexts accidently occur at runtime, and are not removable from rules themselves because the real root of conflicts are the special user requests. For the system administrators, the conflicts of *statically-conflicting* rules are *explicit* while the conflicts of other rules are *implicit*.

4 Efficent detection of statically-conflicting rules

The conflicts between the policy rules give the conflicted evaluation contexts opposite decisions, which are both specified by system administrators and but only one of the access control decisions can be taken. Generally both of the decisions are possible to be choosed as the evaluation result, depending on which automatical resolution mechanism is applied. However, any resoultion mechanisms, whether the least priviledge principle or deny override, may undermine the security of collaboration and sharing by returning a wrong decision as the evaluation result. In particular, the wrongly resolved decisions may lead to a too loose or too tight access control. Conflict detection is necessary because it provides valuable information for specifying and correcting rules so that the evaluation results are always correctly returned to meet the security requirements.

Since *statically-conflicting* rules frequently result in conflicts in a given set of ABAC policy rules, the conflict detection in this paper focuses on the subset of the *conflicting* rules. The task of conflict detection detects all possible *statically-confliciting* rules in the given set of ABAC policy rules, which helps system administrators revise the pairs of rules to *non-conflicting* ones or adjust them based on the automatical resolution mechanisms.

The naive method of detecting conflicts is obvious. The method takes brute-force strategy to compare any pair of rules to check whether they are *statically-conflicting* by testing whether the rules satisfy the four conditions given in the definitio. The problem of the method is its complexity in time and does not scale well to deal with the large number of rules with complex multiple-attribute sets. In fact, the rules in an ABAC policy for open distributed environment may number thousands [23] and the multiple-attribute sets of each rule may contain complex several attribute predicates. Therefore, more efficient methods of conflict dection are needed for *statically-conflicting* ABAC policy rules. In this section, we first present and analyze the native method, and then propose, based on the lessons, the optimized method that efficiently detects the *statically-conflicting* rules.

4.1 Naive algorithm

Algorithm 1 gives the details of the naive detection of *statically-conflicting* rules. Based on the definition 3.3, the algorithm is straight-forward that for each of the rule pairs in the given ABAC policy it examines whether they are *statically-conflicting*, i.e., whether they satisfy the four defined conditions at the same time(Line 5). The function *MAS_shared_contains* checks whether one of the input multiple-attribute sets shares all attribute identifiers with another set and the attribute predicates with shared identifiers have intersected value sets. Lines 13-18 check for the sharing of attribute identifiers, lines 19-34 for the overlapped of value sets. For each pair of rules, the time complexity of *MAS_shared_contains* is $O(N_a)$ where $N_a = 1 + |A(R_i)| + |A(R_j)| +$ total number of the two rules' attribute predicates. Since the total number of rule pairs is $\binom{n}{2} = n(n+1)/2$, the time complexity of algorithm 1 is $O(n^2 * N_a)$.

Based on the time complexity, the cost of naive algorithm can be high for large number of policy rules and complex multiple-attribute sets with many attribute predicates. The reason behind the high cost of naive algorithm is that the detection of conflicting rules for every rule comprises of examinations with any other rules in the ABAC policy, most of which are usefuless, especially when the number of rules in the given policy is large. The rule examinations are essentially *vertical* and brute-force. By *Vertical* examinations, the naive detection does not explore and make good use of the relationships (e.g., redundancy among rules) among rules, assuming rules are totally irrevalent. By brute-force examiniations, naive method spends a large portion of the execution time on examining the unrelated rules, especially when the ABAC policy has thousands of rules with multiple-attribute sets of complex attribute predicates. In fact, the related rules, conflicting or applicable, for a given rule are only a tiny part

Algorithm 1: naive_static_conflict(P)

```
   Input: $P = \{R_1,R_2...,R_n\}$
   Output: the pairs of statically-conflicting rules
           in the policy $P$
1  begin
2      ret={};
3      for $i \leftarrow 1$ to $n-1$ do
4          for $j \leftarrow i+1$ to $n$ do
5              if $d(R_i) \neq d(R_j)$ and
               $A(R_i) \cap A(R_j) \neq \phi$ and
               $MAS\_shared\_contains(S(R_i), S(R_j) =$
               $MAS\_shared\_contains(O(R_i), O(R_j)$
               and
               $MAS\_shared\_contains(O(R_i), O(R_j)! =$
               $0)$ then
6                  ret +={($R_i$,$R_j$)};
7              end
8          end
9      end
10     return ret;
11     MAS_shared_contains($S_i$,$S_j$) Input: Two
               multiple-attribute-sets $S_i$ and $S_j$
       Output: return 1 if $S_i$ shares all attribute
               identifiers with $S_j$ and their value
               sets with shared identifers intersect,
               return -1 if $S_j$ shares all attribute
               identifiers with $S_i$ and their value
               sets with shared identifers intersect,
               otherwise return 0
12     begin
13         $attr\_ids_i$ = attribute_identifers($S_i$) ;
14         $attr\_ids_j$ = attribute_identifers($S_j$) ;
15         $attr\_ids = attr\_ids_i \cap attr\_ids_i$;
16         if $attr\_ids \neq attr\_ids_i$ and
           $attr\_ids \neq attr\_ids_j$ then
17             return 0;
18         end
19         for $ap_i$ in $S_i$ do
20             ret = false;
21             $S_t = (\forall ap_j$ in $S_j$ and
               $Aid(ap_i) = Aid(ap_j)$ );
22             if IsEmpty($S_t$) then
23                 continue;
24             end
25             for $ap_j \in S_t$ do
26                 if $V_{ap_j} \cap V_{ap_i} \neq \phi$ then
27                     ret = true;
28                     break;
29                 end
30             end
31             if ret = false then
32                 return 0;
33             end
34         end
35         if $attr\_ids \neq attr\_ids_i$ then
36             return 1;
37         end
38         return -1;
39     end
```

of the whole set of policy rules.

4.2 Optimized method

Naive detection of *statically-conflicting* rules is less efficient because of the *vertical* and *brute-force* examinations. This paper proposes an optimized method that performs well even for conflict detection for ten thousands rules. The method achieves efficient conflict detection by exploring the relationships among rules and avoiding unecessary rule examinations as much as possible.

The optimized method detects conflicting rules for a target rule in two steps: rule reduction and binary-search based conflict detection. The method doesn't detect directly conflicts on the original policy rules but instead on the reduced rules generated by rule reduction. The reduced rules are semantically equivalent to the original rules, but they can be performed policy analysis such as conflict detection more efficiently. The idea of rule reduction is to explore and make good use of the redundancy among rules. Rule reduction transforms the redundant orginal rules into reduced rules, whose attribute predicates with same identifiers either identical or non-intersected. For example the orginal rules $R_i = (workyear < 10, fscope = norm, read, allow)$ and $R_j = (workyear < 8, fscope = special, write \| read, allow)$ is by rule reduction turned into three reduced rules $R_a = (workyear < 8, fscope = norm, read, allow)$, $R_b = (8 <= workyear < 10, fscope = norm, read, allow)$ and $R_c = (workyear < 8, fscope = special, write \| read, allow)$. The optimized method examines reduced rules *horizontally* that it checks one attribute identifier for comparing all the reduced rules' attribute predicates afters another, rather than one rule by another. By organizing the identical attribute predicates (e.g. rule R_a's attribute predicate $workyear < 8$ and rule R_c's attribute predicate $workyear < 8$) of different reduced rules into one attribute predicate(e.g. attribute predicate $workyear < 8$), the optimized method examines every attribute predicate no more than once, even it is shared by multiple rules. Furthermore, the multiple-attribute sets can be compared efficiently since their attribute predicates with same identifiers are made non-intersected. For this reason, the reduced rules are structured into multiple attribute predicate lists. Every list presents a list of attribute predicates with a given attribute identifier and an attribute predicate point to a list of original rules that can be reduced into rules containing such an attribute predicate, as shown in figure 1.

The second step of the optimized method is detect

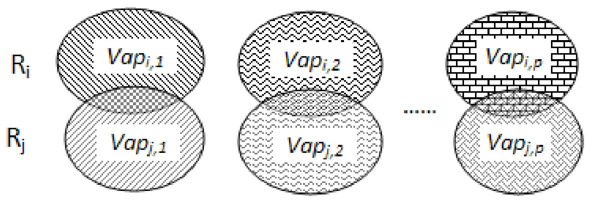


Figure 1. Example of structured reduced rules

the conflicting rules over the reduced rules. The detection needs to compare the attribute predicates of both subject and object multiple-attribute sets. The optimized method uses binary-search to avoid the brute-force comparisons of the rules' attribute predicates. Instead of comparing the list of attribute predicates with a given attribute identifier one by one, the optimized method sorts the attribute predicates and bases upons binary search to efficiently find those attribute predicates intersect with the target rule. The sorting of the list of attribute predicates is possible because the attribute predicates in the list do not intersect with each other, which is guaranteed by the rule reduction step. For a given attribute predicate of the target rule, binary-search of the intersected attribute predicates in the list firstly compares the given attribute predicate with the middle one in the list, then halves the search scope by using the low half or the high half as the new list of attribute predicates for searching, until the intersected middle attribute predicate is found or is determined to be none. Since binary-search halves the search scope and compares only the middle attribute predicates, the total number of comparisons of attribute predicates is supposed decreased exponentially. Therefore, binary search based conflict detection scales well for detecting conflicting rules among a large number of rules.

4.2.1 Rule reduction

The motivation of the technique is that among the given ABAC policy rules there is redundancy: many rules govern access control decisions for the same set of subjects and/or objects. For example in table 1, the access control decisions of subjects with *workyear* no less than 5 and less than 8 are governed by rules 4,5 and 6. The redundancy of multiple-attribute sets forces the naive methods to examine rules in brute-force way, every rules should be examined at least once to prevent missing a single conflict. Intuitively, the subjects that can not satisfy rule 4's subject attribute predicate with attribute identifier *workyear* can never satisfy the subject attribute predicates with the same attribute identifier of rules 5 and 6, thus the examiniations of rules 5 and 6 could be avoided. Rule reduction turns the given ABAC policy rules into a reduced set of policy rules that are compact but semantically equivalent to the given rules which make the conflict detection give the same outputs. Since the reduced rules are compact and structured, the unnecessary rule examinations can be thus minimized that only a subset of the reduced rules are examined and no more once. The formal definition of reduced policy rules is given as below.

Definition 4.1 (*Reduced rules and policy*) . *Two policy rules R_i and R_j are reduced if they satisfy one of the following conditions:*

(1)All the shared subject and object attribute identifiers have the value sets of the corresponding attribute predicates identical, i.e., $\forall ap_i$,ap_j such that ((ap_i in $S(R_i)$) $\wedge$ (ap_j in $S(R_j)$)) or ((ap_i in $O(R_i)$) $\wedge$ (ap_j in $O(R_j)$)), and $Aid(ap_i) = Aid(ap_j)$, they must hold $V(ap_i) = V(ap_j)$. Or

(2)At least one of their shared subject or object attribute identifier has the value sets of the corresponding attribute predicates do not intersect, i.e., $\exists ap_i$, ap_j such that ((ap_i in $S(R_i)$) $\wedge$ (ap_j in $S(R_j)$)) or ((ap_i in $O(R_i)$) $\wedge$ (ap_j in $O(R_j)$)), $Aid(ap_i) = Aid(ap_j)$, they must hold $V(ap_i) \cap V(ap_j) = \phi$.

If any pair of rules in an ABAC Policy $P = \{R_1, R_2..., R_n\}$ are reduced, then the policy is a reduced policy.

For example in table 1, rules 5 and 6 are reduced because the attribute predicates with the shared attribute identifier *fscope* have value sets do not intersect, but rules 4 and 5 are not reduced because both value sets of attribute predicates with shared attribute identifiers (i.e.,*workyear* and *fscope*) interset, and thus the policy in the example is not reduced.

Rule reduction is to transform the original rules into reduced rules that have no intersected attribute predicates. Given two rules that have redundancy, the reduction needs only transform the attribute predicates with shared identifiers. This is because an attribute identifier only existing in one of the two rules has the attribute predicates trivially non-intersected (because $V_{ap} \cap \phi = \phi$). For one of the shared identifiers, the two rules' attribute predicates are reduced into several non-intersected ones. Suppose two rules R_i and R_j are not reduced, and the attribute predicates with shared identifiers $\{aid_1, aid_2,...,aid_p\}$ are $ap_{i,1}, ap_{i,2}, ...ap_{i,p}$ in rule R_i and $ap_{j,1}, ap_{j,2}, ...ap_{j,p}$ in rule R_j, as showed in figure **??**. For each of the shared attribute identifiers aid_k, their value sets are made up of three subsets that do not intersect: $V_{ap_{i,k}} - V_{ap_{i,k}} \cap V_{ap_{j,k}}$, $V_{ap_{i,k}} \cap V_{ap_{j,k}}$ and $V_{ap_{j,k}} - V_{ap_{i,k}} \cap V_{ap_{j,k}}$. Suppose the attribute predicates corresponding to value sets are apr_i, apr_{ij},and apr_j, according to the above decomposition of intersected value sets, the attribute predicates with a shared attribute identifier can be reduced into non-intersected attribute predicates apr_i, apr_{ij},and apr_j. In particular, attribute predicate ap_i is reduced into attribute predicates apr_i and apr_{ij}, and attribute predicate ap_j into attribute predicates apr_{ij} and apr_j. For example, intersected attribute predicates $ap_i = (5 \leq workyear \leq 20)$ and $ap_j = (workyear < 10)$ can be reduced into three non-intersected attribute predicates $apr_i = (workyear < 5)$,$apr_{ij} = (5 \leq workyear < 10)$,$apr_j = (10 \leq workyear \leq 20)$. The attribute predicate $workyear < 10$ is reduced into attribute predicates $workyear < 5$ and $5 \leq workyear < 10$, and attribute predicate $5 \leq workyear \leq 20$ into attribute predicates $5 \leq workyear < 10$ and $10 \leq workyear \leq 20$.

The two rules are transformed into reduced rules by reducing every attribute predicate with shared attribute identifiers to non-intersected attribute predicates and then making reduced rules using the reduced attribute predicates. Given the reduced attribute predicates with shared attribute identifer aid_k are $apr_{1,k}$,$apr_{2,k}$,$apr_{3,k}$, corresponding to value sets of $V_{ap_{i,k}} - V_{ap_{i,k}} \cap V_{ap_{j,k}}$, $V_{ap_{i,k}} \cap V_{ap_{j,k}}$ and $V_{ap_{j,k}} - V_{ap_{i,k}} \cap V_{ap_{j,k}}$, the original rule R_i can be reduced into 2^p reduced rules for p shared attribute identifiers, which replace attribute predicates $ap_{i,k}$ of R_i with reduced attribute predicate either $apr_{1,k}$ or $apr_{2,k}$. For example, rules $R_i = (5 \leq workyear \leq 20, fscope = market \| sales, read, allow)$ and $R_j = (workyear < 10, fscope = sales \| admin, read, allow)$ are reduced to 4 rules $R_1^{'} = (10 \leq workyear \leq 20, fscope = market, read, allow)$, $R_2^{'} = (5 \leq workyear < 10, fscope = market, read, allow)$,$R_3^{'} = (10 \leq workyear \leq 20, fscope = sales, read, allow)$ and $R_4^{'} = (5 \leq workyear < 10, fscope = sales, read, allow)$. Similarly the original rule R_j can be reduced into 2^p reduced rules for p shared attribute identifiers, which replace attribute predicates $ap_{j,k}$ of R_j with reduced attribute predicate either $apr_{2,k}$ or $apr_{3,k}$. For example, the original rule R_j also be reduced into 4 rules $R_1^{''} = (workyear < 5, fscope = admin, read, allow)$, $R_2^{''} = (5 \leq workyear < 10, fscope = admin, read, allow)$,$R_3^{''} = (workyear < 5, fscope = sales, read, allow)$ and $R_4^{''} = (5 \leq workyear < 10, fscope = sales, read, allow)$.

The reduced rules have the attribute predicates with shared attribute identifiers equal or non-intersected, which is ensured by the method that the attribute predicates are transformed. The reduced rules are also semantically equivalent to the original rules, as stated in the following theroem.

Theorem 4.2 *The orignal policy $P = \{R_i, R_j\}$ and the reduced rules $P^{'} = \{R^{'}_1, R^{'}_2 ..., R^{'}_k\}, 1 \leq k \leq 2^{p+1}$ are semantically equivalent.*

Proof 3 *(1)prove any tuple (S, O, A, d) specified by the original policy P also is specified by the reduced policy $P^{'}$. Given the tuple (S, O, A, d), without loss of generality suppose rule R_i satisfies $S(R_i) \cap S \neq \phi \wedge O(R_i) \cap O \neq \phi \wedge A(R_i) \cap A \neq \phi \wedge d(R_i) = d$. According to the rule reduction, the attribute predicate $ap_{i,k}$ with shared attribute identifier of $S(R_i)$ is divided into non-intersected attribute predicates $apr_{1,k}$ and $apr_{2,k}$,i.e. $V_{ap_{i,k}} = V_{apr_{1,k}} \cup V_{apr_{2,k}}$. Since $S \cap S(R_i) \neq \phi$, there exists an attribute predicate $V_{apr_{s,k}}, s = 1$ or 2 should have its value set intersected with V_{apt_k}, the value set of attribute predicate apt_k of S. Otherwise $V_{ap_{i,k}} \cap V_{apt_k} = (V_{apr_{1,k}} \cup V_{apr_{2,k}}) \cap V_{apt_k} = (V_{apr_{1,k}} \cap V_{apt_k}) \cup (V_{apr_{2,k}} \cap V_{apt_k}) = \phi$, contradicating with $S \cap S(R_i) \neq \phi$. Similarly for every object attribute predicate apt_k with shared object attribute identifier of O, there exists $V_{apr_{s,k}}, s = 1$ or 2 should intersect with V_{apt_k}. Therefore, the required reduced rule $R^{'}$ can be found by substituting rule R_i's subject or object attribute predicate $ap_{i,k}$ with shared attribute identifier by an attribute predicate $apr_{s,k}, s = 1$ or 2, and keeping other attribute predicates, action sets and decisions untouched. According to definition, the reduced rule $R^{'}$ satisfies $S(R^{'} \cap S \neq \phi \wedge O(R^{'} \cap O \neq \phi$. Based on the making of rule $R^{'}$, we have $A(R^{'} \cap A = A(R_i \cap A \neq \phi \wedge d(R^{'}) = d(R_i) = d)$.*

(2)Prove any tuple (S, O, A, d) specified by the reduced policy $P^{'}$ is also specified by the original policy P. Given the tuple (S, O, A, d), suppose the reduced rule $R^{'}$ satisfies $S(R^{'}) \cap S \neq \phi \wedge O(R^{'}) \cap O \neq \phi \wedge A(R^{'}) \cap A \neq \phi \wedge d(R^{'}) = d$. Without loss of generality, suppose rule $R^{'}$ is generated by transforming rule $R_i \in P$. According to the rule reduction, we have $S(R^{'}) \subseteq S(R_i) \wedge O(R^{'}) \subseteq O(R_i) \wedge A(R^{'}) = A(R_i) \wedge d(R^{'}) = d(R_i)$. According to the definition 2.7, the value set of any subject or object attribute predicate $apr_{s,k}, s = 1$ or 2 of rule $R^{'}$ is the subset of the value set of attribute predicate $ap_{i,k}$ with same identifier of rule R_i, i.e. $V_{apr_{s,k}} \subseteq V_{ap_{i,k}}$. Therefore $V_{apr_{s,k}} \cap V_{apt_k} \neq \phi \Rightarrow V_{ap_{i,k}} \cap V_{apt_k} \neq \phi$, which further implies $S(R^{'}) \cap S \neq \phi \Rightarrow S(R_i) \cap S \neq \phi$ and $O(R^{'}) \cap O \neq \phi \Rightarrow O(R_i) \cap O \neq \phi$. Furthermore, $A(R_i) \cap A = A(R^{'}) \cap A \neq \phi$ and $d(R_i) = d(R^{'}) = d$.

According to definition 2.9, (1) and (2) implies that policy P is equivalent to policy $P^{'}$.

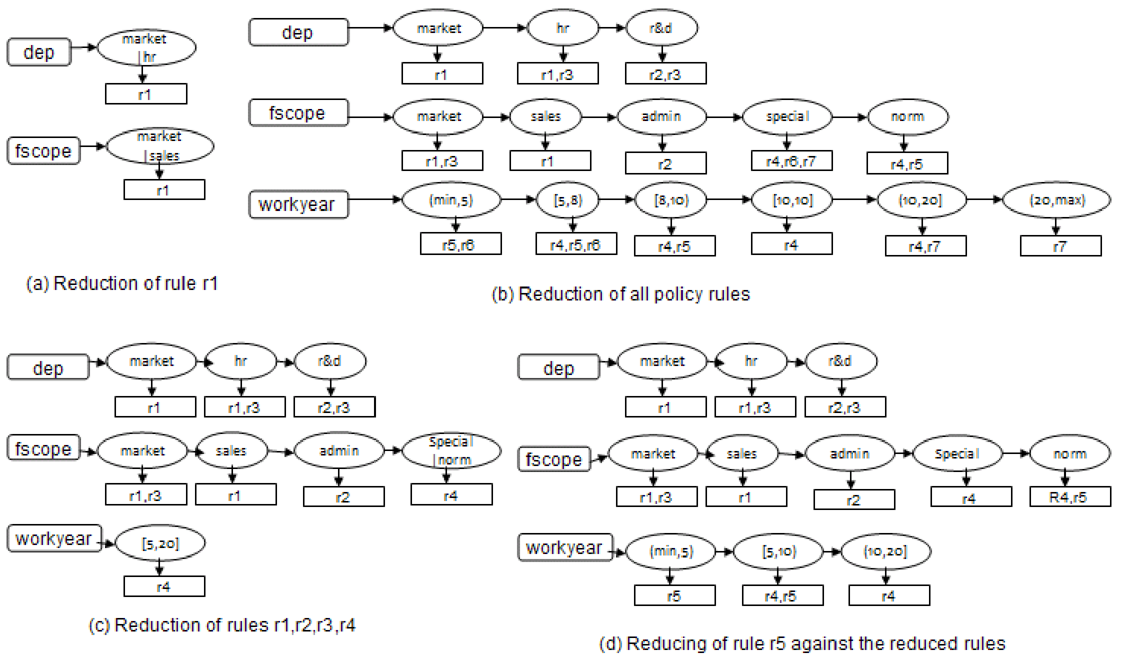
The basic principle of rule reduction is to divide the rules' intersected attribute predicates into several non-intersected attribute predicates and make the reduced rules by substituting the original rules' attribute predicates of shared identifiers with the generated attribute predicates by the division. Given a set of policy $P = \{R_1, R_2 ..., R_n\}$ with more than 2 rules, the rules are not directly reduced on basis of pairs of two rules because the pairs of rules of reduction is $n(n-1)/2$, high for large number of rules, and the generated rules from two different pairs may be not reduced too so that the reduction may require many loops before it ends. Instead, we follow the basic principle of the rule reduction and reducing the given set of rules in policy $P = \{R_1, R_2 ..., R_n\}$ on basis of rules by reducing a rule after another against the already reduced rules *horizontally*, utile all the rules being reduced. The whole reduction process comprises of n loops. The generated rules after the last reduction of rule R_n are reduced that they have attribute predicates with same identifiers identical or non-intersected. Every loop of the reduction is to reduce the rule $R_i (1 \leq i \leq n)$ against the already reduced rules. The reduction of a rule is performed on the basis of attribute predicates.

As illustrated in figure 2, the reduced rules are represented as nodes of attribute identifiers which have a list of non-intersected attribute predicates. Every attribute predicate in the list has a rule set containing the original rules one of whose attribute predicates are divided into this attribute predicates. For example in figure 2.b, rule $R_6 = (workyear < 8, fscope = special, write \| read, deny)$ has its attribute predicate $workyear < 8$ divided into two attribute predicate $(min, 5)$ and $[5, 8]$ in the list of attribute identifier node $workyear$. The representation of reduced rules presents the redundancy relationship among rules in it that the rules in the rule set of every attribute predicate show that they share the attribute predicate. Since the attribute predicates in a list do not intersect, brute-force search of intersecting attribute predicate with the given rule can be avoid.

Figures 2.b is the final result of reduction of ABAC policy rules given in table 1. The details of reducing rules in a given policy $P = \{R_1, R_2 ..., R_n\}$ is given as follows.

(1)(**Reudction of rule** R_1) As shown in figure 2.a, for each attribute predicate ap of rule R_1, make an attribute identifier node with $Aid(ap)$ as the key, and attribute predicate ap as only item in the node's attribute predicate list. The attribute predicate ap adds R_1 into its rule set.

(2)(**Reduction of rule** $R_i, 2 \leq i \leq n$) For each attribute predicate ap of rule R_i, if the attribute identifier node with $Aid(ap)$ as the key does not exist, make it and add attribute predicate ap to the node's attribute



(a) Reduction of rule r1

(b) Reduction of all policy rules

(c) Reduction of rules r1,r2,r3,r4

(d) Reducing of rule r5 against the reduced rules

Figure 2. Example of reduction of policy rules

predicate list; otherwise, find in the node's list the intersected attribute predicate with *ap* and directly add *ap* to the list if none is found or divide the found attribute predicates against *ap* and add the generated attribute predictes back into the list. Figures 2.c and figure 2.d illustrate the representations before and after reduction rule R_5 in table 1.

The dection of all *statically-conflicting* rules among the given ABAC policy $P = \{R_1, R_2, ...R_n\}$ can be performed by suming up all conflicting rules with rules $R_i, 1 \leq i \leq n$. For a given rule R_i, only rules $R_1, R_2, ...R_{i-1}$ are required to be examined for conflicts. This is due to by summing up all the examinations every pair of two rules is checked once and only once for possible conflicting. With reduction, the detection of conflicting rules of rule R_i can be performed when the rule is reducing. The reduction of rule R_i is based on the represention result of rules $R_1, R_2, ...R_{i-1}$. For each attribute predicate of rule R_i finds and reduces the intersected attribute predicates in the list of corresponding attribute identifier node and gets the original rules in rule set of intersected attribute predicates as intermediate results of examination. After all of rule R_i's attribute predicates are processed, the possible conflicting rules are concluded by summerizing all the results and examining the original rules action sets and decisions. For example in figure 2, the conflicting rules with rule R_5 is found when the rule is reduced as shown in figure 2.d where it is reduced against the reduction result of rules $R_1, R_2, ...R_4$. The figure shows rule R_5's attribute predicate with identifer *workyear* intersects with the attribute predicate of original rule R_4, and the attribute predicate with identifier *fscope* intersects with attribute predicate of R_4 too. Furtherly, the action sets of rule R_4 and R_5 are examined to intersect with each other, but their decisions are the same, therefore none of rules $R_1, R_2, ..., R_4$ are *statically-conflicting* with rule R_5.

However, some improvements should be made for efficiency reasons. Firstly, rule reduction does not consider the rules' action sets, which can be examined efficiently when detecting *statically-conflicting* rules. Secondly, the intermediate results can not be summerized efficiently when the rules have different number of subject or object attribute predicates. For example given a rule $r_i = (dep = market \wedge rank < 4, fscope = market, write, deny)$ have the intermediate result $\{r_3\}$ for attribute predicate $dep = market$, $\{r_3, r_2\}$ for attribute predicate $rank < 4$ and $\{r_3, r_2\}$ for attribute predicate $fscope = market$. According to definition of *statically-conflicting* rules, rule r_3 is conflicting with rule r_i if their action sets intersect and decisions are different, but the conflicting relationship between rules r_2 and r_i further depends on the number of subject and object attribute predicate of rule r_2: they are conflicting only when rule r_2 has only one subject attribute

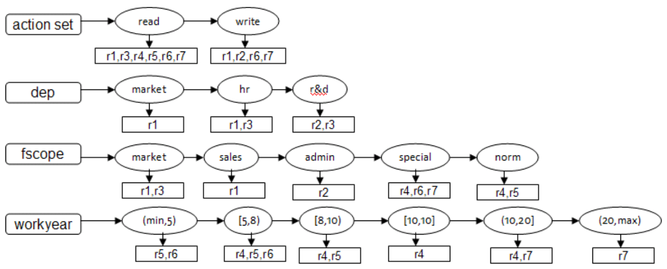


Figure 3. Improved rule reduction of rules in table 1

predicate and one object attribute predicate.

The first improvement is to include the rules' action sets in the reduction representation. Since a rule's action set comprises of a set of action names, the action set can be divided into several action names which are placed into the hash table to facilitate the searching of interaction of action sets. The hash table of action sets takes an action name as the key and the set of rules that contain the action name as the value of the key. As illustrated in figure 3, the action name *read* is hashed to the value $\{R_1, R_2, R_6, R_7\}$.

The second improvement is to sort the policy rules based on the number of attribute predicates , and the sorted rules are input to the rule reduction in order. The larger number of attribute predicates a rule has, the earlier it is reduced. The summerization of conflicting rules based on the intermediate results is now easier: When reducing a given rule, the rules that exist in all the intermediate results for every attribute predicates should be the *statically-conflicting* rules with the given rule. This is because the given rule has all its attribute predicates intersected with the rules' attribute predicates with same identifiers, and the sorting assures the rules have larger or equal number of subject or object attribute predicates as the given rule.

Algorithm 2 gives the details of turning a set of ABAC policy rules into the reduced rules represented in the attribute identifiers nodes. Firstly, the policy rules are sorted based on the rules' numbers of attribute predicates (Line 2) in order to ease the summary of the conflicting rules. Then each of the sorted policy rules is reduced one by one(Lines 3-21). The first step of a rule's reduction is to reduce the action sets with hash table(Lines 4-7), each action name hashing into a rule set. The second step is to reduce the rule's subject and object attribute predicates(Lines 8-20). Lines 8-9 collect the attribute predicates of the rule. Every attribute predicate is reduced against the list of attribute predicates of the attribute identifer node by finding intersected attribute predicates (Lines 12-13) and dividing them into non-intersected ones (Lines 14-16). The last step of a rule's reduction is to add the rule into the rule sets of attribute predicates (Lines 18-20).

The time complexity of the algorithm can be analyzed as follows. The sorting of the n policy rules can be done in $nlgn$ time with comparison-based algorithms. The time cost for the reduction of rule $R_i (1 \leq i \leq n)$ is the sum of reduction cost of action set, subject and object attribute predicate. Due to the hashing nature of action set reduction, the cost depends on the number of action sets, i.e., the cost is $O(|A(R_i)|)$. For every subject or object attribute predicate of rule R_i, the time cost sums up the cost of finding and reducing the intersected attribute predicates, and updating the rule sets for the reduced attribute predicates of rule R_i. Generally the finding cost depends on both the number of attribute predicates in the list and the number of intersected attribute predicates. Suppose the cost for rule R_i's attribute predicate $ap_{i,k}$ to reduce against a list of attribute predicates of size $s_{i,k}$ and get $t_{i,k}$ intersected attribute predicates is $O(f(s_{i,k}, t_{i,k}))$. The dividing cost is proportional to the number of intersected attribute predicates,i.e., $O(t_{i,k})$. The updating cost of the rule sets is determined by the total number of intersected attribute predicates, i.e., $O(\sum_k t_{i,k})$. In summary, the time complexity of the algorithm is $O(nlgn) + \sum_{i=1}^{n}(O(|A(R_i)|) + \sum_k(O(f(s_{i,k}, t_{i,k})) + O(t_{i,k}) + O(t_{i,k})))$. The time complexity largely depends on the efficiency of finding intersected attribute predicates because in practice both the number of intersected attribute predicates $O(t_{i,k})$ and the size of action sets $O(|A(R_i)|)$ can be assumed to be less than a big enough constant. The optimized method utilizes efficient binary-search to lower the time complexity so that the algorithm scales to even large number of policy rules.

The space complexity of the algorithm depends on how many attribute predicates are added for dividing intersected attribute predicates into non-intersected ones. Though the exact number of added attribute predicates depends on the given rules in the policy P, the number can not grow more than two times because the dividing only changes the size of value sets of attribute predicates, not the end points of value sets. For example, attribute predicates $5 \leq workyear \leq 8$ and $7 \leq workyear \leq 10$ share the endpoints $5, 7, 8, 10$ with their reduced attribute predicates $5 \leq workyear < 7$, $7 \leq workyear \leq 8$ and $8 < workyear \leq 10$. The dividing of attribute predicates can be analyzed in two cases for the increasing number of attribute predicates: the dividing results in point value sets of attribute predicates or not. If no point value sets are generated, given m endpoints, the division can result in $m-1$ interval value sets from $m/2$ attribute predicates due to 2 points to form a value set. In this case the maximum growth of the number of attribute predicates is $(m-1)/(m/2) < 2$. In another case suppose the dividing results in p point value sets and and $m-1$ interval value sets, then the minimum number of attribute predicates generating the interval value sets is $m/2$, and the p point value sets are generated for at least other $p/2$ attribute predicates share endpoints with the same $m/2$ attribute predicates. Thus, the growth ratio of attribute predicates in this case is $(m-1+p)/(m/2+p/2) < 2$. Since the number of added attribute predicates after dividing grows less than two times, the algorithm is efficient in space cost that makes itself scalable for large number of policy rules.

Algorithm 2: rule_reduction(P)

```
   Input: $P = \{R_1,R_2...,R_n\}$
   Output: the reduced represenation
           $attributeIdentiferNodes$
 1 begin
 2     sort the rules in policy $P$ in descended order
       based on the number of a rule's attribute
       predicates;
 3     for $R_i$ in $P$ do
 4         for $act$ in $A(R_i)$ do
 5             $rule\_set$=actionMap.get($act$);
 6             $rule\_set.add(R_i)$;
 7         end
 8         $aplist = S(R_i).aplist$ ;
 9         $aplist.addAll(O(R_i).aplist)$;
10         $reslist = \{\}$;
11         for $ap$ in $aplist$ do
12             $node$=$attributeIdentiferNodes$.get($Aid(ap)$);
13             $aps$=findIntersectedAP($ap$,$node.list$,0,
               $node.list.length$-1);
14             for $apt$ in $aps$ do
15                 $reslist$.addAll(reduce($apt$,$ap$));
16             end
17         end
18         for $ap$ in $reslist$ do
19             $ap.rule\_set.add(R_i)$;
20         end
21     end
22     return $attributeIdentifierNodes$;
23 end
```

non-intersected attribute predicates, continuous. Binary search attribute predicates.

4.2.2 Binary search and its complexity

The optimized method uses binary serch to improve the efficiency of searching intersected attribute predicates, and futher the efficiency of conflict detection. As discussed before, search opertions are common both in rule reduction and in conflict detection and the time cost of search operations has significant impact on their complexity. Expensive search operations lead to high cost of the algorithms, and cheaper search operations make both rule reduction and conflict detection more efficient. Binary search decreases the time cost of these search operations, which finds in a given list those attribute predicates whose value sets intersect with a given attribute predicate.

Given a list of attribute predicates ap_list and a target attribute predicte ap, the naive search operation is to examine every attribute predicates one by one which results in a time complxity of $O(|ap\_list|)$. Generally two techniques that both improve search operations are binary search and hash. Hash is more efficient than binary search because it finds in a list an object in constant time by mapping directly the key to the location of the qualified object. However, hash technique is difficult to be applied to the search operations of the optimized method because the hash function, which maps the given attribute predicate(e.g. the attribute predicate $4 < workyear < 6$) and its multiple intersected, not identical, attribute predicates(e.g., the attributes $workyear < 5$ and $5 < workyear < 8$) into the same location, is hard to design.

Binary search is a recursive search procedure that halves the list of objects in the middle after comparing the given object with the middle object util the qualified object is found or the lenth of list reaches zero. Binary search assumes that the objects in the given list are comparable and sortable. Generally the attribute predicates are not comparable for two reasons: (1)an attribute predicate can predicate on categorical value, e.g., $dep = market$;(2)two attribute predicates may have intersected value sets, e.g. $2 < workyear < 8$ and $workyear > 6$. Fortunately, the solution exists that represents every value set of attribute predicates as an interval because our policy model consider only five basic operators in attribute predicates: (1) A numberical attribute predicate directly represents the lower and upper boundary of the value set as the lower and upper boundary of the interval, e.g., the value set of $5 \leq workyear \leq 20$ is represented as the interval $[5, 20]$; (2)A categorical attribute predicate (e.g. the attribute predicate $dep = market$) can be represented as an interval if we give every categorical value of a specified identifier a distinct index, e.g., if the index of value $market$ is 1, the value set of $dep = market$ is de-

Algorithm 3: binary_search(res,ap,ap_list,l,h)

```
Input: The result of searching $res$,the target attribute predicate $ap$, a list of sorted
       attribute predicates $ap\_list$, and the lowest position $l$ and highest position $h$ in
       the list to search
Output: return true if attribute predicates are found, otherwise return false
begin
    $Low = ap\_list[l]$;$High$=$ap\_list[h]$;
    if $Low > ap$ then
        $res.insertpoint$=$l$;
        return false;
    end
    else if $High < ap$ then
        $res.insertpoint$=$h$;
        return false;
    end
    $m = (l + h)/2$;$Mid = ap\_list[m]$;
    if $Mid > ap$ then
        return binary_search($res$,$ap\_list$,$l$,$m - 1$);
    end
    else if $Mid < ap$ then
        return binary_search($res$,$ap\_list$,$m + 1$,$h$);
    end
    $res.ap\_list.insertAt(0, Mid)$;$pos$=$m$ ;
    while $--pos >= l$ do
        $Mid = ap\_list[pos]$;
        if not intersected(Mid,ap) then
            break;
        end
        $res.ap\_list.insertAt(0, Mid)$;
    end
    while $++pos \leq h$ do
        $Mid = ap\_list[pos]$;
        if not intersected(Mid,ap) then
            break;
        end
        $res.ap\_list.insertAtTail(Mid)$;
    end
    return true;
end
```

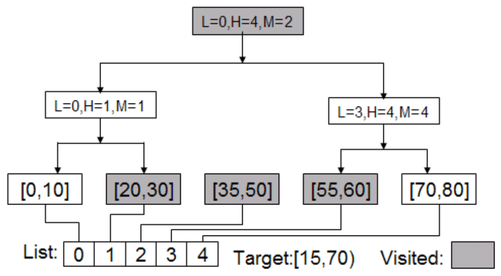


Figure 4. Example of binary search

noted by the interval $[1, 1]$. In addition, the attribute predicates in the list are reduced to have no intersected value sets so that they are comparable and sortable based on their interval representation. The comparison between two non-intersected attribute predicates is formally given as follows.

Definition 4.3 (*Comparison between two attribute predicates) Attribute predicate ap_i is greater than attribute predicate ap_j, denoted as $ap_i > ap_j$, if either of the following conditions holds:*

(1) the lower boundary of $V(ap_i)$ is greater than the upper boundary of $V(ap_j)$, i.e. $V(ap_i).low > V(ap_j).high$;

(2) the lower boundary of $V(ap_i)$ is equal to the upper boundary of $V(ap_j)$, but one of the boundary flags are open, i.e., $V(ap_i).low = V(ap_j).high$, $\neg(V(ap_i).low\_flag == V(ap_j).high\_flag == CLOSE)$;

Two attribute predicates ap_i and ap_j are equal, denoted as $ap_i = ap_j$, if $(V(ap_i).low = V(ap_j).low \wedge (V(ap_i).low\ _flag = V(ap_j).low\_flag \wedge V(ap_i).high = V(ap_j).high) \wedge V(ap_i).high\_flag = V(ap_j).high\_flag)$.

Two attribute predicates ap_i and ap_j are comparable if one of the following conditions holds:

(1) $ap_i > ap_j$, or (2) $ap_j > ap_i$, or (3) $ap_j = ap_i$

Different from general binary search technique, binary search in the optimized method gets in the list one or more attribute predicates whose value sets intersect with the given attribute predicates as a result. For example in figure 4, binary search returns three attribute predicates $[20, 30]$,$[35, 50]$ and $[55, 60]$ for the given attribute predicate $[15, 70)$. Furthermore, the returned attribute predicates are always adjacent in the list due to the given attribute predicate is in nature one continuous interval.

Algorithm 3 gives the details of binary search. Given a list of sorted attribute predicates ap_list and an attribute predicte ap, and the scope of the list to search defined by the lowest l and highest position h, the steps of the algorithm can be summarized as follows: (1)it tests whether even the attribute predicate at position l is greater than the given attribute predicate ap (Line 3-6), or the given attribute predicate ap is greater than the one at position h (Line 7-10) by comparing, and sets the insert position if they do and returns immediately without finding any attribute predicates; (2)Otherwise it picks up the attribute predicate at middle position m and compares it with the given attribute predicates ap(Line 12-17). If the given attribute predicate ap is less than the middle one Mid, it sets new searching scope to $[l, m-1]$ and calls the algorithm resursively and returns immediately(Line 12-14). If the middle attribute predicate Mid is less than the given attribute predicate ap, it sets new searching scope to $[m+1, h]$ and calls the algorithm resursively and returns immediately (Line 15-17);(3)Otherwise the middle attribute predicate Mid intersects with the given attribute predicate ap, and inserts the middle attribute predicate as one of the intersected attribute predicates(Line 18);(4)Since the attribute predicates adjacent to the middle one are also possible to intersect with the given predicates, it also looks backward and foreward to search other instersected attribute predicates as results(Line 19-37). The algorithm returns a list of sorted intersected attribute predicates if they exist or returns the insert position for the given attribute predicate to be inserted in the list as sorted.

Like the general binary search technique, Algorithm 3 halves the searching scope in the middle after comparing with the middle attribute predicates. The time cost to find the first intersected middle attribute predicate depends on where the intersected attribute predicates positioned in the list. For example in figure 4, one of the intersected attribute predicates $[35, 50]$ happens to at middle position 3, so it is found in the first call to binary_search(res,ap,ap_list,0,$ap\_list.size()-1$), and other answers at found by looking backward and foreward around the middle attribute predicates. Generally, the algorithm finds the first answer in time complexity of $O(lgn)$ for n attribute predicates in the list, and other answers in time complexity $O(m-1)$ for m total intersected attribute predicates. Therefore, the overall time complexity for the algorithm is $O(lgn+m-1)$,also $O(lgn+m-1)$ for m intersected attribute predicates in a list of n sorted attribute predicates.

4.2.3 Algorithm of efficient conflict detection and complexity

Algorithm 4 gives the details of detecting *statically-confliting* rules in a given an ABAC policy $P = \{R_1, R_2..., R_n\}$. The conflicting rules of rule R_i are found using the representation of the already reduced rules $R_1, R_2, ..., R_{i-1}$ when the rule is being reduced. Therefore, the algorithm is based on algorithm 2 added with the detection of conflicting rules. the algorithm also uses *binary_search* to improve the efficiency of searching intersected attribute predicates in the list(Line 17). For every attribute predicate of rule R_i, the rule sets of its intersected attribute predicate are kept in set *rule_set* (Line 20). And the conflicting rules are summarized by intersecting all the rule sets in *rule_sets* which contains the rule sets of every attribute predicates of rule R_i. The rules left in the intersection result *s_rule_set* with different decisions with rule R_i are *statically-conflicting* rules(Lines 27-30) due to rule R_i shares all its ation sets, and attribute predicates with these rules but specifying different decisions.

The time complexity of the algorithm is the time complexity of algorithm 2 plus the time complexity of extra detection of conflicting rules. With the efficient implementation of set operations such as bitmap-based set, the operations of rule sets can be completed in the constant time $O(1)$. Moreover, *binary_search*-based searching of intersected attribute predicates is efficient in time complexity of $O(lgn+m-1)$. Therefore, the time complexity of the algorithm is $O(nlgn) + \sum_{i=1}^{n}(O(|A(R_i)|) + \sum_k(O(lg(s_{i,k}) + t_{i,k} - 1)) + O(t_{i,k}) + O(t_{i,k}) + O(1)))$. Because the size of list $s_{i,k}$ are less than $2*n$ for n rules, in the given policy P, and suppose the number of intersected attribute predicates $t_{i,k}$ is less than a big enough constant $O(1)$, the time complexity is less than $O(nlgn) + n*N_k*(lgn+O(1)+3*O(1)) = O(N_k*nlgn) = O(nlgn)$. Therefore the time complexity of the algorithm is considered as $O(nlgn)$.

5 Experiments

In order to evaluate the efficiency of the optimized method, we implemented both naive and optimized methods, and compared them with rules produced by an attribute-based policy rule generator. Our experiments focus on the time cost of detecting all the *statically-conflicting* rules in a given ABAC policy using different methods. Efficiency of the methods are examined under different number of rules and different number of attribute predicates of the rules.

5.1 Experiment Setup

The algorithms of naive and efficient detection of conflicts are both developed in Java language. Naive method is implemented in direct manner according to

Algorithm 4: efficient_conflict_detection(P)

```
Input: $P = \{R_1, R_2 ..., R_n\}$
Output: the statically-conflicting rules
        $conflicting$ in the given policy
1  begin
2      $conflicting = \{\}$;
3      sort the rules in policy $P$ in descended order
       based on the number of a rule's attribute
       predicates;
4      for $R_i$ in $P$ do
5          $s\_rule\_set = \{\}$;
6          for $act$ in $A(R_i)$ do
7              $rule\_set$=actionMap.get($act$);
8              $s\_rule\_set += rule\_set$;
9              $rule\_set.add(R_i)$;
10         end
11         $aplist = S(R_i).aplist$ ;
12         $aplist.addAll(O(R_i).aplist)$;
13         $reslist = \{\}$;
14         $rule\_sets = \{\}$;
15         for $ap$ in $aplist$ do
16             $node$=$attributeIdentiferNodes$.get($Aid(ap)$);
17             $binary\_search(aps, ap, node.list, 0,$
               $node.list.length$-1);
18             $rule\_set = \{\}$;
19             for $apt$ in $aps$ do
20                 $rule\_set$ += $apt.rule\_set$;
                   $reslist$.addAll(reduce($apt$,$ap$));
21             end
22             $rule\_sets += rule\_set$;
23         end
24         for $rs$ in $rule\_sets$ do
25             $s\_rule\_set = s\_rule\_set \cap rs$;
26         end
27         for $r$ in $s\_rule\_set$ do
28             if $r.decision != R_i.decision$ then
29                 $conflicting += \{r, R_i\}$;
30             end
31         end
32         for $ap$ in $reslist$ do
33             $ap.rule\_set.add(R_i)$;
34         end
35     end
36     return $conflicting$;
37 end
```

algorithm 1. The optimized method implementation comprises of rule reduction, binary-search and bitmap-based set operations. In bitmap-based set, every rule in the given ABAC policy $P = \{R_1, R_2, ...R_n\}$ is indexed from 0 to $n-1$, and each rule set of the reduced attribute predicate is represented as a bitmap of an array of integers where a bit is set for the corresponding rule in the rule set and is clear otherwise. The basic set operations such as union and intersection can be completed in constant time.

We built a rule generator which can produce a ABAC policy with a given number of rules using the predefined settings: (1) the set of attribute identifiers and the value ranges of the attributes; (2) the action names. A multiple-attribute set of a rule contains several attribute predicates that the identifiers and the value sets are picked uniformly at random according to the predefined setting (1). The actions of a rule are also uniformly picked based on setting (2). The decision of a rule is generated by randomly choosing the value *allow* or *deny*.

The experiments are conducted on a Pentium M 1.7GHz, 1 Gb RAM machine.

5.2 Results

In the experiments, three different groups of ABAC policy rules with 1, 3 and 5 attribute predicates in a multiple-attribute set respectively are used. Figure 5 shows the results of the experiments, where *#attr* denotes the number of attribute predicates per multiple-attribute set.

The results of all three groups of experiments show that the optimized method outperforms the naive method, especially when the policy has more than 10000 rules. In particular, the time cost of conflict detection increases squarely with the number of rules using the naive method, but increases linearly using the optimized method. For example when *#attr=3*, the cost using naive method is about 40000 milliseconds for 10000 rules and grows to about 160000 milliseconds for 20000 rules. While for the same value of *#attr* the cost using optimized method increases from 5258 milliseconds to 12298 milliseconds. Consider that the number of pairs of rules for detecting conflicts grows squarely with *#attr*, the optimized method takes less time to process each pair of rules for a larger number of rules.

Compare the results of different groups of experiment, larger *#attr* results in smaller time cost of conflict detection. For example for 10000 rules, the optimized method takes 5448 milliseconds when *#attr=1* and takes 3746 milliseconds when *#attr=5*. The reason behind is that the same number of policy rules produced by the rule generator have more *statically-conflicting* rules when *#attr* is smaller and the high conflicting ratio policy raises the time cost of conflict detection. When the conflicting ratio is constant, larger *#attr* should cost more time to detect the conflicting rules in the given policy.

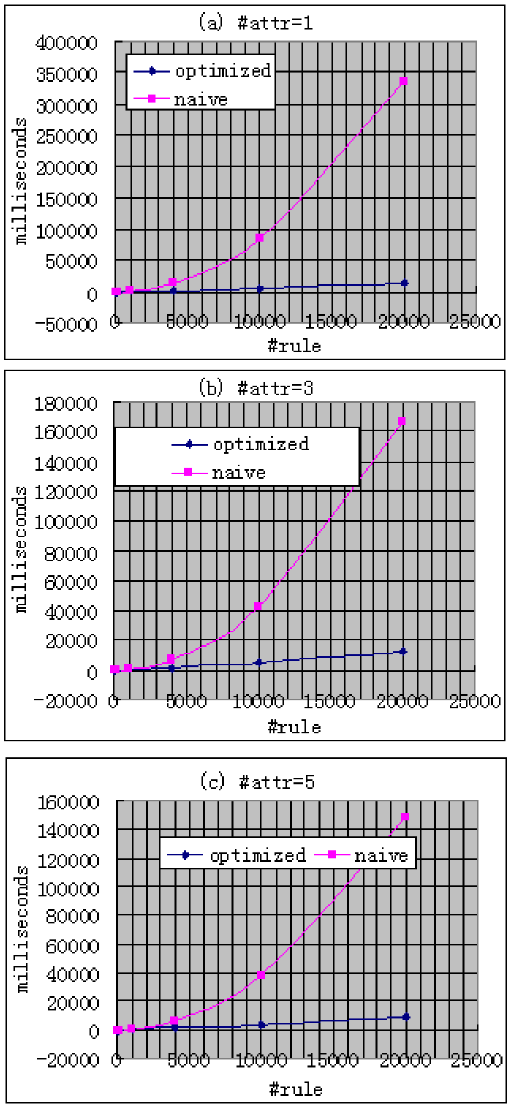


Figure 5. Time costs of conflict detection

6 Related work

Conflicts of policy have been studied extensively in previous research work [7, 9, 17]. However, they mainly addresses the cyclic inheritance and separation of duties(SOD) conflicts of hierachical policies(e.g., RBAC and LBAC). Two recent approaches proposed to analyze conflicts between policy rules are model-checking[8] based and logical reasoning based[1]. But both approaches are not computationally complex. Model-checking based approach explodes in the size of policy representation nodes for a large scale policies. Logical reasoning based approach [1] models the problem of conflict detection as five categories of boolean expressions and presents algorithms to examine if the satifiable solutions exist in the policy rules. Though the algorithms are applicable for a variety of policy constraints, they are expensive and only determine the existence of conflicts not the exact conflicting rules. Since a policy contains a large number of rules [23] and contains many attributes[8], conflict of policy rules should be analyzed efficiently in terms of time cost.

Previous work on conflict detection and resolution have mainly been performed in the context of Role-based Access control(RBAC) and Lattice-based access control(LBAC) systems. Mediator-based approaches are proposed in [9, 17]. [9] charaterizes the security and autonomy principles, and investigates the theoretical complexity of security interoperations that has proved the general case and optimization of secure interoperation are intractable. [17] presents an policy integration algorithms for construting a global non-conflicting multiple-domain collaborating policy. [17] formulates the secure interoperation as an optimization problem of an objective of maximazing interoperability without violating security of collaborating domains when two principles of interoperation can be be guarateed simultaneously, and employs the Integer-based approach for optimal resolution of such conflicts. [16] proposes a mediator-free approach for making conflictless access control decisions across multiple domains using both basic and extended path links rules. [7] addresses the conflicts in establishing secure interoperation among LBAC systems. Since the conflicts in these work, such as cyclic inheritance and separation of duties, are specific to hierachical policies (e.g. RBAC and LBAC), their approaches are not applicable in attribute-based access control policies. [1] dicusses conflicts as one of the rule interactions as domainance and coverage and provides algorithms like domain elimination, linear inequalities and solution trees to detect the existences of conflicts among the policy rules. Whereas the algorithms are applicable to more general access control policy rules, they can not find the exact conflicting rules thus be less helpful for conflicts elimination.

[21] specially provides solutions to consolidating access control of composite applications based on the policies of sub-applications in context of application interoperation. Our ABAC policy model is partially based on the policy model in [21]. The policy that doesn't fulfill the Least priviledge (LP) and Maximum set of subjects (MS) criteria is considered to be conflicting, which coincides with security and autonomy principles[9] in RBAC. This definition ignores the conflicting rules in the policies of sub-applications that give different decisions for the same user requests. [21] use the notation of *reduced policies* for policies that can be efficiently consolidated. The subject elements of all rules in a reduced policy are equal and the object and action elements specify disjoint priveges. This paper not only proposes more general definition of reduced policies for detecting *statically-conflicting* rules, but also presents the algorithm of reducing ABAC policy rules based on notation of *semantically-equivalence*. Besides conflict detection, the proposed rule reduction in this paper is applicable to efficient analysis of policy rules such as policy dominance and similarity.

7 Conclusions

ABAC policy rules are subject to conflicts that make contradicting decisions for the same user requests. This paper formally defines the conflicts between ABAC policy rules. Because of the nature of attribute-based rule specification with *multiple-attribute sets*, the policy rules are special that two seemingly unrelated rules may conflict which make the conflicts undetectable and removable prior to runtime. The paper focuses on *statically-conflicting* rules and investigates how to efficient detect all *statically-conflicting* rules in a given ABAC policy. The main contributions of the paper include:

(1)Formal definition of conflicts between ABAC policy rules and the notation of *statically-conflicting* rules whose conflicts are detectable and removed without evaluation any user requests.

(2)Two techniques that make the conflict detection in a given policy more efficient: rule reduction and binary search. Rule reduction transforms the policy rules into a set of compact, semantically equivalent reduced rules that simplify the analysis of statically-conflicting rules. Binary search technique is used to search m intersected attribute predicates in a list of n non-intersected attribute predicates with time complexity $O(lgn + m - 1)$.

(3)The comprehensive alogrithm of the optimized method to detect all *statically-conflicting* rules in a given ABAC policy. The time complexity of the algorithm is $O(nlgn)$, and the experiments have shown it can handle complex ABAC policies with over 20,000 rules in nearly linear time.

References

[1] D. Agrawal, J. Giles, K. Lee, and J. Lobo. Policy Ratification. In *Proc. the 6th POLICY*, pages 223-232, June 06-08, 2005, Stockholm, Sweden.

[2] R. Alfieri, R. Cecchini, V. Ciaschini et al. VOMS: an Authorization System for Virtual Organizations. In *Proc. 1st European Across Grid Conference*, Santiago de Compostela, Feb. 13-14, 2003.

[3] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. An Algebra for Composing Access Control Policies. *ACM Transactions on Information and System Security*, 5(1),pages 1–35, 2002.

[4] P. Bonatti, and P. Samarati. A unified framework for regulating access and information release on the web. *Journal of Computer Security*,10(3),pages 241–272,2002.

[5] D.W. Chadwick, and O. Otenko. The PERMIS X.509 Role Based Privilege Management Infrastructure. In *Proc. the 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, June 2002.

[6] J. Chomicki, J. Lobo, and S. Naqvi. Conflict resolution using logic programming. *IEEE Transaction on Data and Knowledge Engineering*, 15(1), pages 244-249, 2003.

[7] S. Dawson, S. Qian, and P. Samarati. Providing Security and Interoperation of Heterogeneous Systems. In *Proc. 14th International Conference on Information Security (SEC98)*, Vienna-Budapest, Aug. 31-Sept. 2, 1998.

[8] K. Fisler, S. Krishnamurthi, L.A. Meyerovich, and M. C. Tschantz. Verification and Change-Impact Analysis of Access-Control Policies. In *Proc.the*

27th international conference on Software engineering (ICSE'05),Pages 196-205,St. Louis, Missouri, USA, May 15-21, 2005.

[9] L. Gong, and X. Qian. Computational Issues in Secure Interoperation. *IEEE Transactions on Software Engineering*, 22(1),pages 43–52, 1996.

[10] H. Li, D. Groep, L. Wolters, J. Templon. Job Failure Analysis and Its Implications in a Large-scale Production Grid. In *Proc. 2nd IEEE International Conference on e-Science and Grid Computing(eScience 2006)*,Amsterdam, The Netherlands, Dec 4 - 6, 2006.

[11] E. Lipu, and M. Sloman. Conflicts in policy-based distributed system management. *IEEE Transaction on Software Engineering*, 25(6), pages 852-869, Nov. 1999.

[12] M. Lorch, and D. Kafura. The PRIMA Grid Authorization System. *Journal of Grid Computing*, 2(3), pages 279-298, 2004.

[13] P. Mazzoleni, E. Bertino, and B. Crispo. XACML Policy Integration Algorithm. In *Proc. the 11th ACM symposium on Access control models and technologies*,Pages 219-227,Lake Tahoe, California,USA,June 07 - 09, 2006 .

[14] Security service technical committee. eXtendible Access Control Markup Language Committee specification 2.0. 2005.

[15] L. Pearlman, V. Welch, I. Foster, K. Kesselman, S. Tuecke. A community authorization service for group colaboration. In *IEEE Workshop on Policies for Distributed Systems and Networks*, 2002.

[16] M. Shehab, E. Bertino, and A. Ghafoor. Secure Collaboration in Mediator-Free Environments. In *CCS'05*, November 7-11,2005, Alexandria, Virginia, USA.

[17] B. Shafiq, J. B. D. Joshi, E. Bertino, and A. Ghafoor. Secure Interoperation in a Multidomain Environment Employing RBAC Policies. *IEEE Transactions On Knowledge and Data Engineering*, 17(11), pages 1557–1577, 2005.

[18] M. Thompson, A. Essuaru and S. Mudumbai. Certificate-based Authorization Policy in a PKI Environment. *ACM Transactions on Information and System Security(TISSEC)*, 6(4), pages 566-588, 2003.

[19] H. wang, S. Jha, M. Livny, and P. D. McDaniel. Security Policy Reconciliation in Distributed Computing Environments. In *Proc. the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, Page 137-148, 2004, New York, USA,June 07 - 09.

[20] L. Wang, D. Wijesekera, and S. Jajodia. A Logic-based Framework for Attribute based Access Control. In *Proc. 11th ACM Conference on Computer and Communications Security(CCS'04)*, Washington, DC, USA, October 25-29, 2004.

[21] M. Wimmer, A. Kemper, M. Rits, and V. Lotz. Consolidating the Access Control of Composite Applications and Workflows. In *Data and Applications Security 2006*, LNCS 4127, pages 44-59,2006.

[22] E. Yuan, J. Tong. Attributed Based Access Control (ACBC) for Web Services. In *Proc. the IEEE International Conference on Web Services (ICWS05)*, Pages 561-569, Orlando, Florida, USA,July 11-15, 2005.

[23] A. Schaad, J. Moffett, and J. Jacob. The role-based access control system of a European bank: a case study and discussion. In *Symposium on Access Control Models and Technologies*, pages 3-9, Chantilly, Virginia, United States, 2001.