Detecting Conflicts in ABAC Policies with Rule-reduction and Binary-search Techniques

Cheng-chun Shu
Institute of Computing Technology (ICT)
Chinese Academic of Science (CAS)
Beijing, P. R. China
shuchengchun@software.ict.ac.cn

Erica Y. Yang and Alvaro E. Arenas
E-Science Centre
STFC Rutherford Appleton Laboratory
Didcot, Oxfordshire, U. K.
{*erica.yang, alvaro.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 distributed computing environments. However, ABAC policy rules are often complex making them prone to conflicts. 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.**

I. Introduction

Access control policies provide secure and controlled resource sharing in a variety of applications including database federation (e.g. [3]), Web services (e.g. [6]) and Grid systems (e.g. [5]). Recently attribute-based access control (ABAC) [2] policies are gaining popularity in open distributed environments. ABAC policies govern user requests based on the characteristics of requestors and resources rather than their identities.

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 restrictive to benefit from the collaboration and sharing of information and resources. The 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* [4] at a policy 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 comprehensible 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.

Our contributions are sumarised as follows. We formally introduce the notions of "semantically equivalent" policies and "statistically conflicting" rules. These notions are then used to improve the efficiency of conflict detection in ABAC policy rules. Our approach includes two techniques: rule reduction and binary search. Based on the notion 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. A variant of the classic binary search technique is applied to search conflicting policy rules upon the reduced rule set, resulting in a set of conflict-free rules.

II. ABAC Policy Model

The formal syntax and semantics of our policy model are based on those introduced by Bonatti [1] and Wimmer[6]. 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*.

A. 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 ith conjunction that uses the jth 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 $R_1, R_2 ..., R_n$.

B. 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 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: (*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 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 e. The set of decisions of applicable rules is returned as the result of rule evaluation.

It is possible that policies with different rules and/or rule specifications can sometimes produce the same set of decisions for a given evaluation context. Such "equivalent" policies are interesting because it suggests an intuitive approach to conduct conflict analysis upon a complex policy through replacing it with a "semantically equivalent but simpler" policy, leading to the improvement in the efficiency of detecting and eliminating conflicts. This is our starting point for searching efficient methods to detect conflicts among policy rules. We term such policies as semantically equivalent policies. The formal definition of semantically equivalent policies is given as follows.

*Definition 2.4: (*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 2.5: 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. $\forall e, \|P\|_e = \|P'\|_e$[1].

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 equivalent 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, without 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.

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*. An interesting category of *conflicting* rules in an ABAC policy is those rules who a) share a common set of attribute identifiers; and b) the attribute predicates of the rules with the same identifiers have intersected value sets. Such rules are common in a set of policy rules and the conflicts among them are detectable and removable prior to runtime. The definition of *statically-conflicting* rules is formally given as follows:

*Definition 2.6: (*Statically-conflicting rules) Two rules R_i and R_j are statically-conflicting if:

(1)Their decisions are distinct,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 $O(R_j)$), $Aid(ap) = Aid(ap') \Rightarrow V_{ap} \cap V_{ap'} \neq \phi$.

The formal definition of reduced policy rules is given as below.

*Definition 2.7: (*Reduced rules and policy) Two policy rules R_i and R_j are reduced if they satisfy one of the following conditions:

[1]The proof of this theorem corresponds to the proof of Theorem 2.10 in the document: http://epubs.cclrc.ac.uk/bitstream/3280/abac_conflicts.pdf.

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

Theorem 2.8: The original 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[2].

III. Detecting Statically-Conflicting Rules

This section describes an optimized method for detecting statistical conflicts among policy rules. The method achieves the efficiency by exploiting the redundant relationships among rules and avoiding unnecessary rule examinations as much as possible.

The optimized method consists of two techniques: rule reduction and conflict detection through a novel binary searching technique. The method does not detect directly conflicts on the original policy rules but instead on the reduced rules generated by the rule reduction technique. The reduced rules are semantically equivalent to the original rules, but policy analysis, such as conflict detection, can be performed more efficiently.

A. Rule Reduction

Rule reduction transforms the original rules into reduced rules that have no intersected attribute predicates. Given two rules that have redundancy between them, the reduction only transforms the attribute predicates with shared identifiers. 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.

The optimized method examines reduced rules *horizontally*: it checks one attribute identifier for comparing all the reduced rules' attribute predicates after 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.

B. Binary Search

The second step of the optimized method is to detect the conflicting rules over the reduced rules. The detection needs to compare the attribute predicates of both subject and object multiple-attribute sets. 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 upon the classic binary search technique to efficiently identify 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.

C. A Combined Algorithm for Conflict Detection

Algorithm 1 gives a combined algorithm for detecting *statically-conflicting* rules in an ABAC policy. 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. The algorithm also uses *binary_search* to improve the efficiency of searching intersected attribute predicates (Line 17). For every attribute predicate of rule R_i, the rule sets of its intersected attribute predicate are kept in the set $rule\_set$ (Line 20). The conflicting rules are generated 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).

IV. Implementation and Experiments

We have implemented the algorithms for naive and optimized methods using Java. The implementation of the optimized method comprises of three techniques: rule reduction, binary-search and bitmap-based set operations. In bitmap-based set, every rule in the given ABAC policy 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

[2]The proof of this theorem corresponds to the proof of Theorem 4.2 in the document: http://epubs.cclrc.ac.uk/bitstream/3280/abac_conflicts.pdf.

Algorithm 1: efficient_conflict_detection(P)

```
Input: $P$ = $\{R_1$,$R_2$...,$R_n\}$
Output: the *statically-conflicting* rules $conflicting$ in the
        given policy
begin
    $conflicting = \{\}$;
    sort the rules in policy $P$ in descended order based on
    the number of a rule's attribute predicates;
    for $R_i$ in $P$ do
        $s\_rule\_set = \{\}$;
        for $act$ in $A(R_i)$ do
            $rule\_set$=actionMap.get($act$);
            $s\_rule\_set += rule\_set$;
            $rule\_set.add(R_i)$;
        end
        $aplist$ = $S(R_i).aplist$ ;
        $aplist.addAll(O(R_i).aplist)$;
        $reslist = \{\}$;
        $rule\_sets = \{\}$;
        for $ap$ in $aplist$ do
            $node$=$attributeIdentiferNodes$.get($Aid(ap)$);
            $binary\_search$($aps$,$ap$,$node.list$,0,
            $node.list.length$-1);
            $rule\_set = \{\}$;
            for $apt$ in $aps$ do
                $rule\_set$ += $apt.rule\_set$;
                $reslist$.addAll(reduce($apt$,$ap$));
            end
            $rule\_sets += rule\_set$;
        end
        for $rs$ in $rule\_sets$ do
            $s\_rule\_set = s\_rule\_set \cap rs$;
        end
        for $r$ in $s\_rule\_set$ do
            if $r.decision != R_i.decision$ then
                $conflicting += \{r, R_i\}$;
            end
        end
        for $ap$ in $reslist$ do
            $ap.rule\_set.add(R_i)$;
        end
    end
    return $conflicting$;
end
```

of the attributes; and (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 settings: (1) The actions of a rule are also uniformly picked based on the settings (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. We use *#attr* to denote the number of attribute predicates per multiple-attribute set. The results of the experiments consistently show that the optimized method outperforms the naive method in detecting statistically conflicting rules. This is particularly evident when a policy consists of more than 10,000 rules. The time cost of conflict detection increases squarely with the number of rules using the naive method. But the same cost only increases linearly using the optimized method. For example when *#attr=3*, the time cost of using the naive method is about 40 seconds for 10,000 rules, which grows to around 160 seconds for 20,000 rules. In contrast, for the same *#attr*, the time cost of using the optimized method increases from 5.3 seconds to around 12.3 seconds, which is about one magnitude faster than that of the naive method.

V. Conclusions

ABAC policy rules are subject to conflicts that make contradicting decisions for the same user requests. This paper formally defines the statistical conflicts between ABAC policy rules, which is a category of conflicting rules in which conflicts can be detected and removed prior to runtime. Our paper focuses on enhancing the efficiency of detecting and removing such conflicts among the rules in an ABAC policy. We have produced a prototype implementation of a conflict detection tool based on the conflict detection method we propose in this paper. The preliminary experimental results are encouraging. The results show that the optimized method is *efficient* in that the time cost of detecting statistically conflicting rules increases linearly as the number of rules increases. The method is also *scalable* because the efficiency remains the same even when the number of rules hits 20,000.

Acknowledgment

This work is partially funded by the European Commission under FP6 projects EchoGRID, grant No 04552, and XtreemOS, grant No 033576.

References

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

[2] 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.

[3] 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.

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

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

[6] 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.