Hindawi
Mobile Information Systems
Volume 2022, Article ID 5408470, 12 pages
https://doi.org/10.1155/2022/5408470

Research Article

Attribute-Based Policy Evaluation Using Constraints Specification Language and Conflict Detections

Wei Sun

School of Computer and Information Technology, Xinyang Normal University, Xinyang 464000, China

Correspondence should be addressed to Wei Sun; sunny810715@xynu.edu.cn

Received 29 June 2022; Accepted 22 August 2022; Published 5 September 2022

Academic Editor: Xingsi Xue

Attribute-based access control (ABAC) has attracted widespread interest and has become an ideal mechanism due to its flexibility characteristic and the powerful expressiveness for various security policies, such as the separation-of-duty constraint and cardinality constraint. The formulation of appropriate ABAC policies is critical for ensuring system security and robustness. However, conflicts occur frequently in existing state-of-the-art systems. Most conventional detection methods either lack the evaluation of the policy quality or consider no constraint. To resolve these problems, a novel method for the ABAC policy evaluation is proposed in this study. First, to meet diverse organizational requirements, we use the attribute-based constraints specification language to uniformly formulate and specify the conflict relations among attributes and present the satisfiability of conflict relations. Second, to comprehensively detect the conflict problems, we present the evaluation criteria for conflicts on attributes and rules and propose a novel algorithm for detecting conflicts. Last, we validate the effectiveness and efficiency of the proposal through experiments, which demonstrate that it not only improves the policy quality but also reduces the conflicting number and conflicting probability.

1. Introduction

With the high-speed development in high-performance computing and mobile-information technology, security has been considered as a fundamental requirement for the research fields such as the Internet of Things (IoT), smart contracts, blockchains, and the industrial information integration system [1]. There are large amounts of data storage and resource sharing in distributed and collaborative environments, and enterprises need to employ some means to ensure the integrity and confidentiality of information systems. As the main benchmark, the role-based access control (RBAC) model had been widely used for system implementation and management over the last few decades [2]. However, it is identity dependent and lacks flexibility and extendibility. As an alternative, attributes are employed to describe the features of entities. The attribute-based access control (ABAC) overcomes the limitations of RBAC, captures fine-grained access requirements, and becomes very attractive, particularly for large-scale distributed and collaborative systems [3]. It has gained much attention in both academia and industry [4] in recent years.

Actually, the ABAC policy rule is the combinational form of different attribute value pairs of subjects, objects, environments, and operations. The policy engineering [5, 6] is to find a suitable ABAC rule set, which is regarded as the most important step for implementing the ABAC mechanism. Xu and Stoller [7] were the first to study the ABAC policy mining problem from the given access control lists or matrices and proposed a bottom-up resolution (represented as the Xu-Stoller for simplicity). To reduce the scale of ABAC rules, Das et al. [8] used the Gini impurity and presented a policy mining method. Das et al. [9] also presented a visual method, called VisMAP, which mined ABAC policies based on a given authorization list.

The ABAC policy is very flexible and extendable. However, values of the attribute-expression conditions specified by different rules are partially identical. If the access decisions of the rules that have identical attribute values are inconsistent, then there exist conflicts among such rules, and

policy maintenance becomes difficult [10]. Therefore, how to detect the conflicts among policies has become an urgent problem to be resolved. Royer and De Oliveira [11] separated the existing conflict detection mechanisms for the eXtensible Access Control Markup Language (XACML) into three different types. The dynamic and testing detections depend on access requests, while the static detection was based on the rule set without generating requests. Jabal et al. [12] summarized the related research on static conflict detections and involved five different variants. St-Martin and Felty [13] converted the XACML policies into the Coq code that included the effect type and the SRAC that represented the "subject-resource-action-condition." Rezvani et al. [14] proposed a method to translate an XACML policy into the form of ASP programming, verified the properties of the policy by an analysis tool, and then, validated its effectiveness. Zheng and Xiao [15] visually specified ABAC rules, converted them into a set of binary sequences and presented a novel method for conflict detection. Shu et al. [16] proposed an optimized method to detect explicit conflicting rules from the given ABAC policy, which utilized the method of rule reduction to eliminate the redundancy of the rules and then adopted the method of binary search to improve the efficiency of the detections. To extend the detecting scope while improving the efficiency of the detections, based on the intersection of rule pairs, Liu et al. [17] proposed a novel approach for detecting both explicit and implicit conflicts. However, the existing methods do not consider the constraint requirements during the detecting processes.

To comprehensively capture the different organizational requirements while ensuring the ABAC system security and confidentiality, an important feature of the ABAC mechanism is to be able to specify and perform the cardinality constraint and the separation-of-duty constraint (SOD). These constraint policies are not relevant to the specific access control mechanism [18]. To implement several classical access control models, Jin et al. [19] presented a novel framework, called $ABAC_{\alpha}$, which specified constraints on the attribute-assignment relationships using the policy specification language. However, the constraints in ABACα were dependent on the specific events, and they became ineffective if the attribute assignments varied. To address this problem while uniformly specifying various types of ABAC constraints, Bijon et al. [20] proposed the attribute-based constraint specification language (ABCL). Based on the subject similarity, Helil and Rahman [21] used the ABAC constraint to check and determine the potential relationships between different entities. To further specify and verify the SoD constraints in ABAC systems, Jha et al. [22] presented a novel approach for analyzing the complexity of enforcing the SOD constraints. To verify whether a set of users could be replaced by another user set, Roy et al. [18] presented the employee-replacement problem with multiple constraints and then provided a scheme for solving the problem. Furthermore, to automatically derive ABAC rules from the conventional access control documents, Alohaly et al. [23] proposed a framework for policy extraction using natural language processing techniques. However, most conventional methods only focus on specifying or formulating constraints on attributes, which do not consider the influence of constraints on the ABAC rules and lack the evaluation of the policy quality. Thus, conflict problems among ABAC rules arise frequently while using the existing research methods.

To resolve the abovementioned problems, this study proposes the attribute-based policy evaluation using constraints specification language and conflict detections (ABPE_CSL&CD). To sum up, the main contributions of this work are as follows:

(1) To flexibly suit organizational requirements, while ensuring system security, we use the ABCL to uniformly formulate and specify the conflict relations among attributes and propose the satisfiability of conflict relations. We take the reconstructed ratio as the evaluation criterion and demonstrate the efficiency of the ABPE_CSL&CD using real datasets.

(2) To comprehensively detect the ABAC conflict problems, while ensuring the system robustness, we present the classification representations for conflicting rules and propose a novel method for conflict detection. We take the conflicting number and conflicting probability as the evaluation criteria and demonstrate the effectiveness of the ABPE_CSL&CD using synthetic datasets.

The rest of the article is organized as follows: Section 2 introduces some necessary preliminaries. Section 3 proposes a novel policy evaluation method and presents an algorithm for conflict detection. We present the experimental analysis in Section 4 and conclude the article and discuss future works in Section 5.

2. Preliminaries

In this section, some preliminaries are presented, including the basic components of the ABAC, basic components of the ABCL, and conflict problems in the ABAC.

2.1. Basic Components of ABAC. According to the ABACα [19], the ABAC model mainly consists of the following sets, relations, and functions:

(1) Sets S, O, and E represent all the subjects, objects, and environments in which the access control occurs, respectively. Set OP represents all the operations that are permitted or denied to be performed on the object resources. Sets SA, OA, and EA, respectively, represent the identifier names of the subjects, objects, and environments, which can be categorized into multi-valued and single-valued types. For instance, the *role* attribute is multi-valued as an employee may own more than one role in an organization, while the *id* attribute is single-valued as any employee in the organization has a unique identifier value.

(2) Functions $atttype(att)$ and $range(att)$, respectively, represent the type and value domain for a specific entity attribute att, which can be formalized as

$$\forall \text{att} \in SA \cup OA \cup EA\text{: atttype(att)} \in \{\text{atomic, set}\}, \text{range(att)} = \left\{val_i^{\text{att}} | i \in Z^+\right\}. \quad (1)$$

For the sake of convenience, the environmental elements E and EA are not taken into account here.

(3) Relation $SSAV$ represents many-to-many assignments relationship of subjects and their attribute-expression conditions, for any user attribute att, which can be formalized as follows:

$$\forall \text{att} \in SA, SSAV\text{: } S \longrightarrow \begin{cases} \text{range(att)}, \text{if atttype(att)} = \text{atomic}, \\ 2^{\text{range(att)}}, \text{if atttype}(att) = \text{set}. \end{cases} \quad (2)$$

Similarly, $OOAV$ can be formalized as

$$\forall \text{att} \in OA, OOAV\text{: } O \longrightarrow \begin{cases} \text{range(att)}, \text{if atttype(att)} = \text{atomic}, \\ 2^{\text{range(att)}}, \text{if atttype(att)} = \text{set}. \end{cases} \quad (3)$$

Furthermore, the value set of attribute att assigned to any user or object entity en is denoted as function $val(en, att)$.

(4) Function $Assigned\_Entities_{EN,att}$ returns the entities with respect to a specific attribute value $attval$, which can be formalized as follows:

$$\forall att \in (SA \quad or \quad OA), \exists \text{attval} \in \text{range(att)}: \text{Assigned\_Entities}_{S\cup O,\text{att}}(\text{attval}) = \{en | \exists en \in (S \quad or \quad O), (\text{val}(en, \text{att}) = \text{attval} \wedge \text{atttype(att)} = \text{atomic}) \quad or (\text{attval} \in \text{val}(en, \text{att}) \wedge \text{atttype(att)} = \text{set})\}. \quad (4)$$

2.2. Basic Components of ABCL. The key component of the ABCL [20] is conflict relations, which are used to determine whether the policy conditions, such as mutually exclusive constraints or cardinality constraints, can be satisfied. Attribute-based conflicts can occur in several ways, in which two critical conflicting variants are considered:

(1) The single-attribute conflict is only applicable for the multi-valued attributes and can be formally expressed in formulation (5). In the formulation, set *Single_Conf_Set* contains various types of the constraint requirements, such as mutual exclusions cardinality constraints, precondition constraints, and so on. Each element of the *Single_Conf_Set* is a 2-tuple form, denoted as (*attval*, *limit*), where *attval* represents a set of conflicts existing in the single-attribute values, and *limit* represents the threshold value for satisfying the security constraint.

$$\forall \text{att} \in (SA \quad or \quad OA), \text{atttype(att)} = set: \text{Single\_Conf\_Set}_{S,O,\text{att}} = \{\text{avset}_1, \text{avset}_2, \ldots, \text{avset}_n\}, \text{where} \quad \text{avset}_i(att) = (\text{attval}, \text{limit}), \text{attval} \in 2^{\text{range(att)}}, and \quad 1 \le \text{limit} \le |\text{attval}|. \quad (5)$$

(2) The cross-attribute conflict is applicable for both the single-valued and multi-valued attributes and can be formally expressed in formulation (6). In the formulation, two different attribute sets are involved, which are represented as *Aattset* and *Rattset*, respectively. The values of one attribute in *Aattset* restrict those of the other one in *Rattset*. Further, set *Cross_Conf_Set* also contains the constraint specifications for different attributes as shown in formulation (6), where each element is a function, named $attfun_i$.

$$\forall (\text{Aattset, Rattset}) \subseteq (SA \quad or \quad OA), \forall att \in (\text{Aattset} \quad or \quad \text{Rattset}): Cross\_Conf\_Set_{S,O,\text{Aattset,Rattset}} = \{\text{attfun}_1, \text{attfun}_2, \ldots, \text{attfun}_n\}, where \quad attfun_i(att) = (\text{attval}, \text{limit}), 1 \le \text{limit} \le |attval|, \left(\text{attval} \in 2^{\text{range(att)}}, \text{atttype}(\text{att}) = \text{set}\right) or (\text{attval} \subseteq \text{range(att)}, \text{atttype}(\text{att}) = \text{atomic}). \quad (6)$$

The ABCL also presents two nondeterministic functions, written as $OE(X)$ and $AO(X)$. $OE(X)$ selects one element from the set X, while $AO(X)$ returns the other elements from X except for the element with $OE(X)$. Both of them are related to contexts. This is because there exists an equation $\{OE(X)\} \cup AO(X) = X$ for the given set X.

2.3. Conflict Problems in ABAC. Taking the ABAC policy specified by the XACML [11] as an example, there exist conflict problems among the rules. Several concepts such as attribute expression, policy rule, access request, and conflict are taken into account in this study, which are described as follows:

(1) Attribute expression *ap*: it can be formalized as a triple $ap = (\text{attribute identifier, operator, and attribute values})$, where the *operator* can take different comparison operators.

(2) Policy rule *ar*: it can be represented as a quadruple $ar_i = (S_i^{ap}, O_i^{ap}, op_i, d_i)$, where op_i represents the operating action on the object resources, d_i represents the positive or negative access decision, which takes only two possible values: *permitted* or *denied*. S_i^{ap} or O_i^{ap} is represented as the set of combinations of various attribute-expression conditions for the subjects or objects. The set of n different rules $ar_1, ar_2, \cdots, ar_n$ constructs an ABAC policy, denoted as $P = \{ar_1, ar_2, \ldots, ar_n\}$.

(3) Access request *req:* it can be formalized as a triple $\text{req} = (S_{\text{req}}^{ap}, O_{\text{req}}^{ap}, op_{\text{req}})$, which indicates that the subjects of S_{req}^{ap} request to perform the operation op_{req} on the objects of O_{req}^{ap}.

(4) Conflict: for the given rules *ari* and *arj,* which satisfy a single request *req* simultaneously while owning the identical attribute identifiers, if the intersections of the attribute-expression conditions of the rules have common attribute values, the operation sets overlap, but if the access decisions are distinct, then there exists a conflict between *ari* and *arj.*

3. Methodology

The proposed ABPE_CSL&CD is threefold: (1) the formulation and specification for conflict relations among attributes, (2) classification representations of conflicting rules, and (3) ABAC conflict detections. Specifically, based on the conventional policy-engineering method, we first construct an initial policy set that takes no conflicts into consideration. Subsequently, according to the requirement descriptions of usage scenarios, such as the banking businesses, we utilize the ABCL to formulate and specify the conflict relations in a single attribute and multiple attributes. Meanwhile, we categorize the conflicting rules into two classifications and present the evaluation criteria for conflicting rules and attribute-conflict relations. Last, we present a novel algorithm for detecting conflicts from the initial policy set and evaluate the performance of the proposal through experiments. The framework of the ABPE_CSL&CD is presented in Figure 1.

3.1. Formulation and Specification for Conflict Relations among Attributes. In this section, an extensive case study in the banking-domain scenario is presented, which utilizes the standard ABCL to formulate various conflict relations among attributes, in order to express the business requirements and ensure system security.

3.1.1. Requirement Descriptions for the Banking Businesses. First, the corresponding attribute characteristics of subjects and objects are presented in Tables 1 and 2, respectively. Each subject is a user who is assigned attributes *id* and *style*. The attribute *role* represents the job responsibility. *trustness_level* and *work_year* are the other two attributes of the subjects, of which the values can be denoted using the comparison operation expressions, such as $trustness\_level \geq 11$, and $work\_year \geq 8$. The descriptions of object features are omitted owing to the space limitation.

According to Tables 1 and 2, the organizational requirements with various constraints are stated as follows:

Const 1: any user can obtain 5 *benefit businesses* at most

Const 2: any user cannot have both the *cashier* and *accountant* roles

Const 3: any user cannot obtain both the bf_1 and bf_2*benefit businesses*

Const 4: any user can together obtain 5 *loan* and *card businesses* at most

Const 5: if the *trustness_level* value of a user is less than 5, then this user cannot obtain more than 1 benefit from $\{bf_1, bf_2, and\ bf_3\}$

Const 6: the number of users obtaining the *house loan business* is no more than 12

Const 7: any two users cannot own the same *id* value

Const 8: if a user has the *house* and *car loan businesse*s and also has more than one *card business*, then this user cannot obtain any *benefit business*

3.1.2. ABCL Specifications for the Security Requirements. Next, the standard ABCL is used to specify the security requirements mentioned above, including the declarations and initializations of different conflict relations in a single attribute and multiple attributes, in which *SMERole* represents a related set of mutually exclusive constraints towards the *role* attribute values, and *OMEBenefit* is another conflict set towards the *benefit business* attribute. Similarly, *SOMETB* and *SOMETWB* represent mutually exclusive conflict sets of the *trustness_level* with *benefit business* and the *trustness_level* and *work_year* with *benefit business* by the *Cross_Conf_Set* variant, respectively. Here, assume that the number of attributes, which are assigned to an entity for satisfying each constraint relation from *Single _Conf_Set* or *Cross_Conf_Set* is less than the threshold *limit*.

(1) Declaration and initialization for *Single_Conf_Set*

(a) $\text{Single\_Conf\_Set}_{S,\ \text{role}}$ SMERole:SMERole = $\{\text{avset}_1\}\text{avset}_1(\text{role}) = (\{\text{cashier, accountant }\}, 2)$

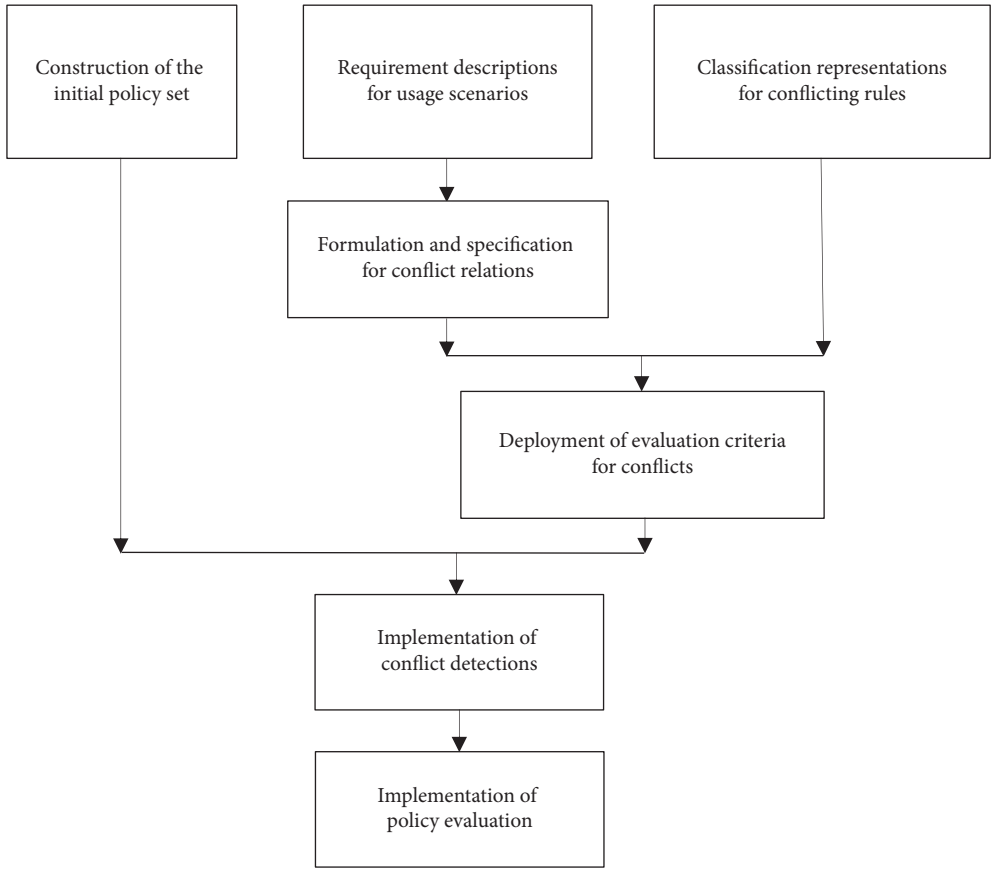

Figure 1: The framework of the ABPE_CSL&CD.

Table 1: Attribute characteristics of subjects.

SA	Type of *SA*	Range of *SA*
Id	*atomic*	$\{id_1, id_2, \ldots id_{20}\}$
Style	*atomic*	{*client and banking employee*}
Role	*set*	{*customer, general employee, cashier, accountant, junior, assistant manager, and manager*}
trustness_level	*atomic*	{1, 2, . . ., 15}
work_year	*atomic*	{1, 2, . . ., 30}

Table 2: Attribute characteristics of objects.

OA	Type of *OA*	Range of *OA*
General business	*atomic*	{*deposit and withdrawal*}
Loan business	*set*	{*house, car, and education*}
Benefit business	*set*	$\{bf_1, bf_2, \ldots, bf_{10}\}$
Card business	*set*	$\{Card_1, card_2, \ldots, card_{12}\}$

(b) Single_Conf_Set$_{O,\ \text{benefit business}}$ OMEBenefit: OMEBenefit = {avset$_1$}avset$_1$(benefit business)=({bf$_1$, bf$_2$}, 2)

(2) Declaration and initialization for *Cross_Conf_Set*

(a) Cross_Conf_Set$_{S,\ O,\ \text{trustness_level},\ \text{benefit business}}$ SOMETB:SOMETB = {attfun$_1$, attfun$_2$} attfun$_1$(trustness_level)=("≤5", 2), attfun$_1$(benefit business)=({bf$_1$, bf$_2$, bf$_3$}, 2) attfun$_2$(trustness_level)=("≤5", 1), attfun$_2$(benefit business)=({bf$_1$, bf$_2$, bf$_3$, bf$_4$, bf$_5$}, 1)

(b) Cross_Conf_Set$_{O,\ \{\text{loan business, ccard business}\},\ \text{benefit business}}$ OMELCB:OMELCB = {attfun1}attfun1(loan business)=({house, car}, 2), attfun1(ccard business)=({card1, card2, . . ., card12}, 2), attfun$_1$(benefit business)=({bf$_1$, bf$_2$, . . ., bf$_{10}$}, 1)

Then, the conflict relations and functions are used to express the security requirements in the ABCL form as follows:

Spec 1: $|\text{benefit\_business}(OE(S))| \leq 5$

Spec 2: $|\text{role}(OE(S)) \cap OE(\text{SMERole}).\text{avset}| < OE(SMERole)$. limit.

Spec 3: $|\text{benefit\_business}(OE(S)) \cap O\ E(OME\text{Benefit}).\text{avset}| < OE(\text{OMEBenefit}).\text{limit}$.

Spec 4: $|\text{loan\_business}(OE(S)) \cap ccar\,d\_business(OE(S))| \leq 5$.

Spec 5: $|\text{trustness\_level}(OE(S)) \cap OE(\text{SOMETB})(\text{trustness\_level}).\text{avset}| < OE(SOMETB)(trustness\_level).limit \Rightarrow |bene\,f\,it\_business(OE(S)) \cap OE(\text{SOMETB})(\text{benefitbusiness}).\text{avset}| < OE(SOMETB)(\text{benefitbusiness}).limit$.

Spec 6: $|\text{Assigned\_Entities}_{S\cup O,\text{loan business}}(\text{house})| \leq 12$.

Spec 7: $id(OE(S)) \neq id(OE(AO(S)))$.

Spec 8: $(|\text{loan\_business}(OE(S)) \cap OE(\text{OMELCB})(\text{loanbusiness}).\text{avset}| \geq OE((OMELCB)(\text{loanbusiness}).limit) \wedge (|\text{ccard\_business}(OE(S)) \cap OE(\text{OMELCB})(\text{ccardbusiness}).\text{avset}| \geq OE(OMELCB)(\text{ccardbusiness}).limit) \Rightarrow |\text{benefit\_business}(OE(S)) \cap OE(\text{OMELCB})(\text{benefitbusiness}).\text{avset}| < OE(OMELCB)(\text{benefitbusiness}).\text{limit}$.

3.2. Classification Representations of Conflicting Rules. According to the description of conflicts, if multiple rules satisfy the same access request, while their access decisions are different, then a conflict occurs among these rules. The conflicting rules can be directly compared and are easy to be detected by implementing the static analysis or using the existing tool before the system runs. They are referred to as explicit conflicting rules and are commonly seen in the policy set, such as ar_5 and ar_6, as shown in Table 3. Furthermore, other rules that seem to be not directly comparable, which are called implicit ones, and they still exist. For instance, implicit conflicts occur between ar_2 and ar_4 when the access request is ({*role* = {*general employee*}, *trustness_level* > 12}, {*loan business* = {*house*}, *card business* = {$card_2$}}, *apply for*). Similarly, ar_3 and ar_5 become implicit conflicting rules for a given request such as ({*role* = {*general employee*}, *trustness_level* > 8, 10 < *work_year* < 20}, {*benefit business* = {bf_1}}, *apply for*). Note that, it is difficult to statistically determine the implicit conflicts since such conflicts can only be detected for the coming access request during the system running.

It has been shown that any two rules can be compared by using the method of attribute complementation [17]. The distinction between the *e*xplicit and implicit conflicting rules just lies in the different representations , while the nature of both is the same. According to whether or not the conflicts happen, all the rules in the policy can be categorized into two classifications as follows:

(1) Probable-conflicting rules ar *and* ar'*need* meet all of the following conditions, where op*()*, d*()*, att*()*, and ap*()*, respectively, represent the corresponding functions or actions with respect to their prefixes:

$$\begin{aligned}
&(a) op(ar) \cap op(ar') \neq \varnothing,\\
&(b) d(ar) \neq d(ar'),\\
&(c)\ \begin{array}{l}\forall \text{att} \in (\text{att}(S,ar), att(O,ar)), \exists \text{att}' \in (att(S,ar'), \text{att}(O,ar')):\\ \text{att} = \text{att}',\end{array}\\
&(d) \forall ap \in (ap(S,ar), ap(O,ar)), \exists ap' \in (ap(S,ar'), ap(O,ar')):\ (\text{att}(ap) = \text{att}(ap')), (ap \cap ap' \neq \varnothing).
\end{aligned} \tag{7}$$

(2) Nonconflicting rules *ar* and *ar*' need meet one of the following conditions:

$$\begin{aligned}
&(a).\ d(ar) = d(ar'),\\
&(b).\ op(ar) \cap op(ar') = \varnothing,\\
&(c). \exists ap \in (ap(S,ar)\ or\ \ ap(O,ar)), \exists ap' \in (ap(S,ar')or\ \ ap(O,ar')):\\
&(att(ap) = att(ap'))\ and\ (ap \cap ap' = \varnothing).
\end{aligned} \tag{8}$$

3.3. Novel Method of Conflict Detections. Different rules may satisfy the same access request because of the flexibility and powerful expressiveness of the ABAC mechanism. The rules with both the *permitted* and *denied* values of the access decision, however, can cause contradictory access results. Therefore, it is necessary to study how to detect and resolve the existing conflict problems.

To comprehensively detect the conflict problems, we first propose the definitions of conflict-relation satisfiability and conflict probability as follows.

Definition 1. Conflict-relation satisfiability

The satisfiability of conflict relations is a function *satisfied*(*ar*, *conf*), which is used to check whether or not the

Table 3: Description of ABAC rules.

Rule	Attribute condition of the subject	Attribute condition of the object	Operation	Decision
ar_1	*role* = {*cashier, junior*}	*general business* = {*deposit, withdrawal*}	*perform, withdraw*	*permitted*
ar_2	*role* = {*general employee, manager*}	*loan business* = {*house*}	*apply for*	*denied*
ar_3	*role* = {*general employee*} and *work_year* < 20	*benefit business* = {bf_1, bf_3}	*apply for*	*denied*
ar_4	*trustness_level* > 12	*card business* = {$card_2$}	*apply for,withdraw*	*permitted*
ar_5	*trustness_level* > 8 and *work_year* > 10	*benefit business* = {bf_1, bf_2}	*apply for,withdraw*	*permitted*
ar_6	*trustness_level* < 10 and *work_year* < 15	*benefit business* = {bf_1, bf_2}	*apply for*	*denied*

rule *ar* could satisfy the specific conflict relation *conf*, where $ar \in P$, $conf \in Conf\_Set$. $Conf\_Set = Single\_Conf\_Set \cup Cross\_Conf\_Set$, which contains all the conflict relations between a single attribute and multiple attributes using the ABCL specification method. It can be formalized as

$$\forall ar \in P, \exists \text{conf} \in \text{Cons\_Set}:$$
$$\text{satisfied}(ar, \text{conf}) = \begin{cases} \text{true, if conf is satisfied} \\ \text{false, if conf is not satisfied} \end{cases}. \quad (9)$$

Definition 2. Conflict probability

Given any two conflicting rules ar_i and ar_j and suppose that there are n common attributes $att_1, att_2, \ldots, att_n$ existing in both the rules. If the access request is uncertain, then the conflict probability between ar_i and ar_j is denoted as

$$\text{Prob}\left(ar_i, ar_j\right) = P^C\left(\text{att}_1\right) \times P^C\left(\text{att}_2\right) \times \cdots \times P^C\left(\text{att}_n\right) = \prod_{att} P^C(\text{att}), \quad (10)$$

where $P^C(\text{att}) = \text{sim}(ap_i, ap_j) = |ap_i \cap ap_j| / |ap_i \cup ap_j|$; ap_i and ap_j are the attribute expressions of the common attribute *att* in ar_i and ar_j, respectively; the Jaccard coefficient $sim(ap_i, ap_j)$ of statistics, which aims to identify sample clusters, is used to measure the similarity between ap_i and ap_j.

According to the classification representation for the conflicting rules, as well as Definitions 1 and 2, we take the initial policy rules constructed by the conventional policy-engineering method as input and present the process of the conflict detections in Algorithm 1.

In the algorithm, we first create and initialize a temporary policy set P' and a result set *CP* of *conflicting-rule pairs* in lines 1 and 2. Next, for each rule *ar* in P', we check whether or not *ar* could satisfy a specific conflict relation *conf* (lines 3–7). If the *satisfied* function returns *false*, which indicates that some constraint is not yet satisfied, then rule *ar* is removed from the candidate policy set. Then, based on the descriptions of the probable-conflicting rules and non-conflicting ones, (lines 8–21) examine and calculate the conflict probability of each rule pair (ar_i and ar_j), in order to detect the *conflicting-rule pairs*, which provides quantitative evaluation criteria for resolving the conflicts and formulating the policies.

4. Experimental Analysis

Experiments are carried out in order to evaluate the performance of the satisfiability of the conflict relations as well as the conflict detections. All the experiments are compiled and run under the Java environment.

4.1. Performance Evaluations for the Satisfiability of Conflict Relations. We employ the real-world and public-available datasets from research [1], in order to construct initial ABAC policies using the Xu-Stoller [7] or VisMAP [9] method, while studying the formulation of the conflict relations, such as the SOD constraints and cardinality constraints. We utilize the Rel-SAT model counter [24] to generate constraint policies from the given SOD constraints.

To simulate the actual scenarios while meeting the security requirements, we implement the experiments in the initial policy-engineering system and adopt the same experimental setup with research [1], including the number of users and the scale of the policy. Figure 2 presents the performances of the proposal using different policy sets.

Two conclusions can be observed in Figure 2. First, as the number of users increases, the execution time does not vary obviously and tends to grow linearly for each given policy. Second, if the number of users remains unchanged, the execution time varies obviously as the scale of the policy changes. Specifically, when the scale of the policy is set at 20, the execution time is always close to 0.04 s, which remains almost stable. However, if the number of users remains constant and is set at 40, the time varies from 0.02 s to 0.04 s. This is because the larger the scale of the policy is, the more verification time for the SOD constraints generated by the Rel-SAT tool will be.

To further demonstrate the efficiency of the conflict-relation satisfiability, we present the following evaluation measure:

Reconstructed ratio (*RR*): It is used to quantitatively evaluate the satisfiability for the constraints during the formulation of conflict-relation sets, which can be denoted as: $RR = |SSAV'|/|SSAV|$, where the $SSAV'$ represents the relationship of the reconstructed attribute assignments that can satisfy the conflict relations, and the $SSAV$ represents the relationship of the initial attribute assignments.

Then, we take the cardinality constraint and SOD constraints as inputs, repeatedly implement the experiments on the real-world datasets, such as University and Healthcare, and output the median values of the experimental results as shown in Figure 3.

Figure 3 shows that the reconstructed ratio *RR* of attributes varies as the threshold of the cardinality constraint varies, where the scale number of the *Conf_set* respectively takes 100, 200, 300, and 400, and the number of attributes in the $SSAV$ constraint set is fixed. It can be observed that the

Input: the set *Conf_Set* of conflict relations and the initial policy set $P = \{ar_1, ar_2, \ldots, ar_n\}$, where $ar_i = (S_i^{ap}, O_i^{ap}, op_i, d_i)$.
Output: the result *set* CP *of conflicting-rule pairs, such as* (ar_i and ar_j) *and the conflict probability* Prob(ar_i and ar_j).

```
(1) Initialize CP = ∅;
(2) Create and initialize a temporary policy set P' = P;
(3) for each ar in P' do
(4)   if ∃conf ∈ Conf_Set: satisfied(ar, conf) = = false then
(5)     P' = P'\{ar};
(6)   end if
(7) end for
(8) for each rule pair (ar_i, ar_j) in P' do
(9)   if (d(ar_i)! = d(ar_j))∧(op(ar_i)∩op(ar_j)! = ∅) then
(10)    for each ap in S_i^ap or O_i^ap of ar_i do
(11)      for each ap' in S_j^ap or O_j^ap of ar_j do
(12)        if (att(ap) = = att(ap'))∧(ap∩ap' = = ∅) then
(13)          continue;
(14)        else
(15)          CP=CP∪{(ar_i, ar_j)};
(16)          calculate Prob(ar_i, ar_j);
(17)        end if
(18)      end for
(19)    end for
(20) end if
(21) end for
```

Algorithm 1: Novel method of conflict detection.

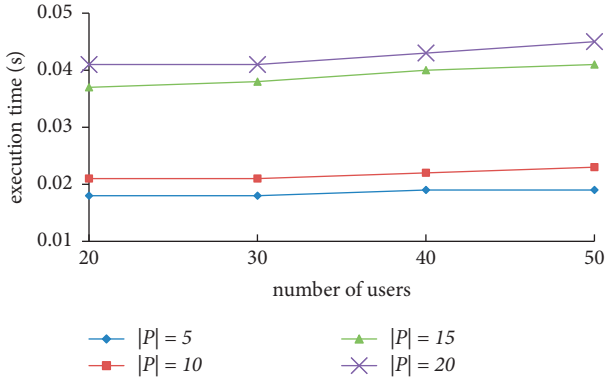

Figure 2: Execution time using different policies.

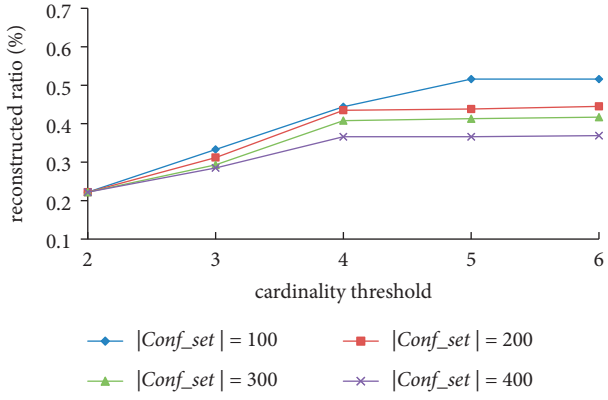

Figure 3: Evaluations of satisfiability using different constraints.

reconstructed ratio first increases significantly and then varies slightly as the constraint threshold increases. This is because the reconstructed ratio is positively correlative to the cardinality constraint, while the saturation will be present when the threshold of the cardinality constraint reaches a certain value. It can be also observed that the reconstructed ratio decreases with the increasing number of cardinality constraints, which is because of the restriction of the constraints on the attribute assignments.

4.2. Performance Evaluations for the Conflict Detections. Next, we use the Shu method [16] and Liu method [17], in order to obtain the synthetic datasets. Specifically, we assume that all the attributes belong to the natural-number type and the value ranges change from 1 to 100 since different attributes can take different values. The number of attribute conditions in each access request follows a normal distribution, and the total number of attributes is less than 40. The value of any attribute-expression condition varies between an upper bound and a low bound and follows a uniform distribution. According to the actual functional requirements of the business organization, we first design a generator to automatically generate various access requests, as well as, 30 different policy sets that are separated into 5 groups for the usage scenario, as shown in Table 4.

To demonstrate the effectiveness of the proposal in the phase of conflict detection, we propose two evaluation metrics as follows:

(1) Average conflicting number $Avg\_N(CP)$: it can be denoted as $Avg\_N(CP) = 1/M \sum_{i=1}^{M} N(ar_i)$, where

Table 4: Descriptions of the generated ABAC rule set.

Group	Policy set	Maximal length of the rule ($\|RL\|$)	Policy scale
1	P_1–P_6	3	100, 200, 400, 700, 1000, 1500
2	P_7–P_{12}	7	100, 200, 400, 700, 1000, 1500
3	P_{13}–P_{18}	11	100, 200, 400, 700, 1000, 1500
4	P_{19}–P_{24}	15	100, 200, 400, 700, 1000, 1500
5	P_{25}–P_{30}	20	100, 200, 400, 700, 1000, 1500

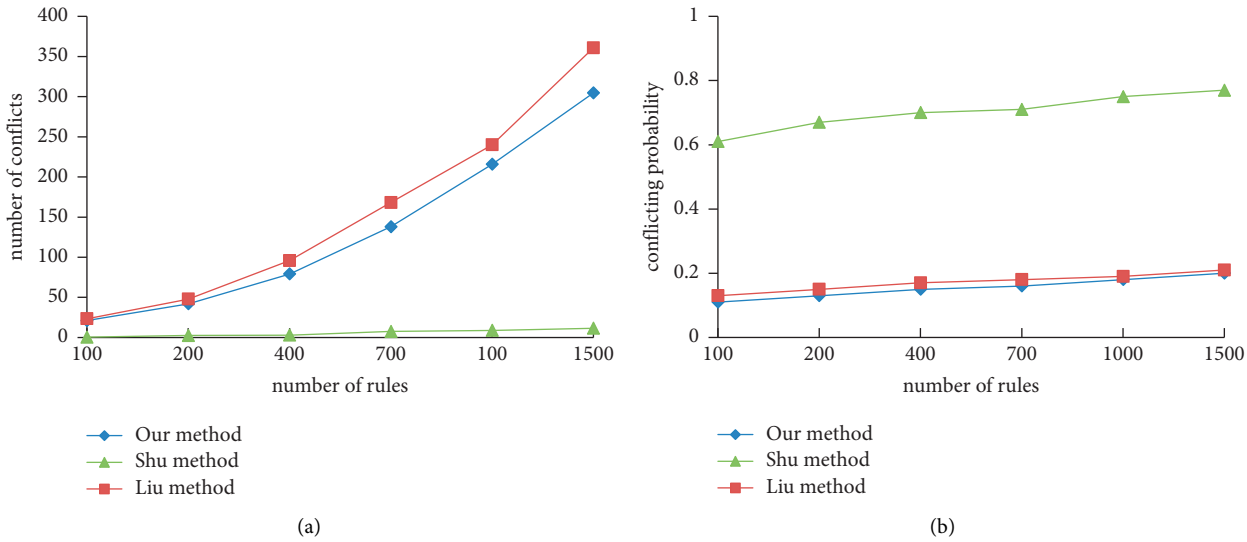

Figure 4: Performance comparisons when $|RL| = 3$. (a) Conflict number. (b) Conflict probability.

$N(ar_i)$ represents the conflicting number of rule ar_i, and M represents the scale number of set CP

(2) Average conflicting probability $Avg\_P(CP)$: it can be denoted as $Avg\_P(CP) = 1/M \sum_{i=1}^{M} (1 - \prod_{(ar_i, ar_j) \in CP} (1 - Prob(ar_i, ar_j))$, where $\text{Prob}(ar_i, ar_j)$ represents the conflict probability between ar_i and ar_j, and M represents the scale number of set CP

We implement experiments on the generated policy sets, calculate $Avg\_N(CP)$ and $Avg\_P(CP)$, and compare the performance of our method with the results of the Liu method and Shu method as shown in Figures 4–8.

Figure 4(a) shows that the average conflict number for P_1–P_6 policies vary with the increasing number of rules when the maximal length of the rule is set at 3. Specifically, the average conflicting number using our method varies from 21.17 to 304.63, which increases remarkably as the number of rules increases from 100 to 1500. Similarly, the result of the Liu method also increases significantly from 23.29 to 360.78 as the number of rules varies. However, the result of the Shu method tends to grow linearly and varies gradually from 0.45 to 11.5, which is a very small proportion of the actual conflicting rules. Obviously, the conflicting number using the Shu method is far less than those of the other two methods, which is not applicable to the actual requirements of organizations. This is because the Shu method only considers the attribute expressions and rules with the same identifiers. The rules with different attribute identifiers, however, are not taken into account using such a method. In fact, most conflicts occur among the implicit conflicting rules, which tend to be ignored and are not easy to be detected. To resolve this issue, both the Liu method and our method perform better and can detect the implicit conflicting results, and the conflicting number using our method is less than that of the Liu method. This is because the probable-conflicting rules violating the attribute-conflict relations are first removed using our method, before actually detecting conflicts. Furthermore, for the values of $|RL|$ taking 7, 11, 15, and 20, the varying tendencies of the average conflicting rules are presented in other figures, which are similar to that of figure 4(a). Notice that the results of both the Liu method and our method decrease as the number of the attribute-expression conditions increases. Thus, it is suggested to choose as many attribute conditions as possible, in order to formulate more flexible policies while reducing the number of probable conflicts.

Figure 4(b) shows the average conflict probability for the first 6 policies in *Group* 1. It is observed that using our method and the Liu method, the conflicting probability is always lower than 0.2 as the number of rules varies, while the result of the Shu method exceeds 0.6. With an increase in the length of the attribute-expression conditions, the average conflict probability decreases remarkably as shown in the figures. For instance, its value remains around 10^{-2}, 10^{-3}, 10^{-5}, and 10^{-7} when $|RL|$ takes 7, 11, 15, and 20, respectively. Both our method and the Liu method perform better than

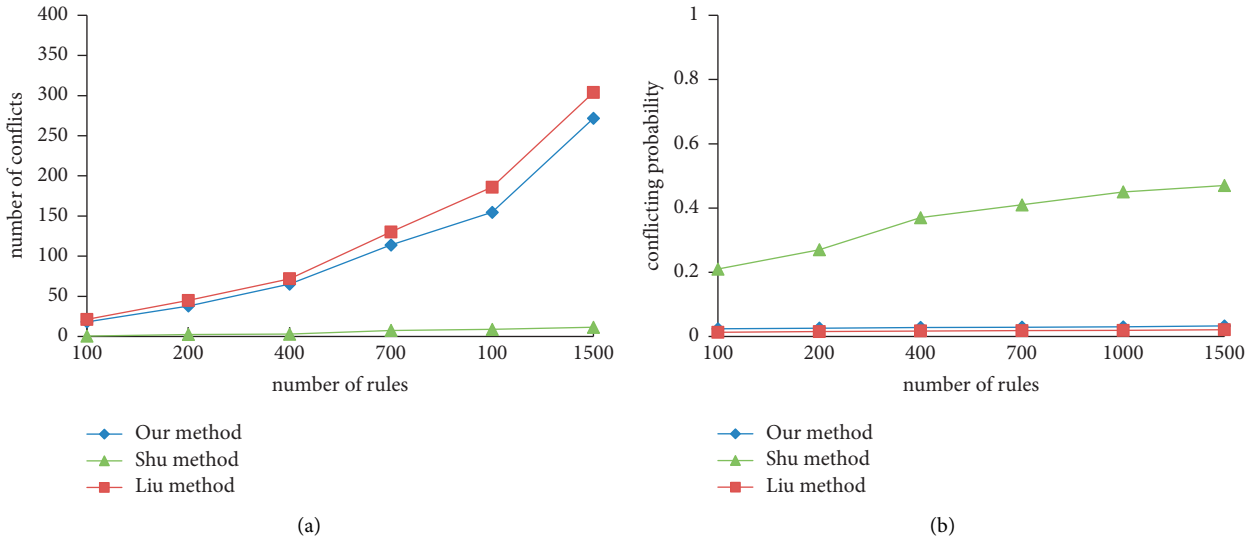

Figure 5: Performance comparisons when $|RL| = 7$. (a) Conflict number. (b) Conflict probability.

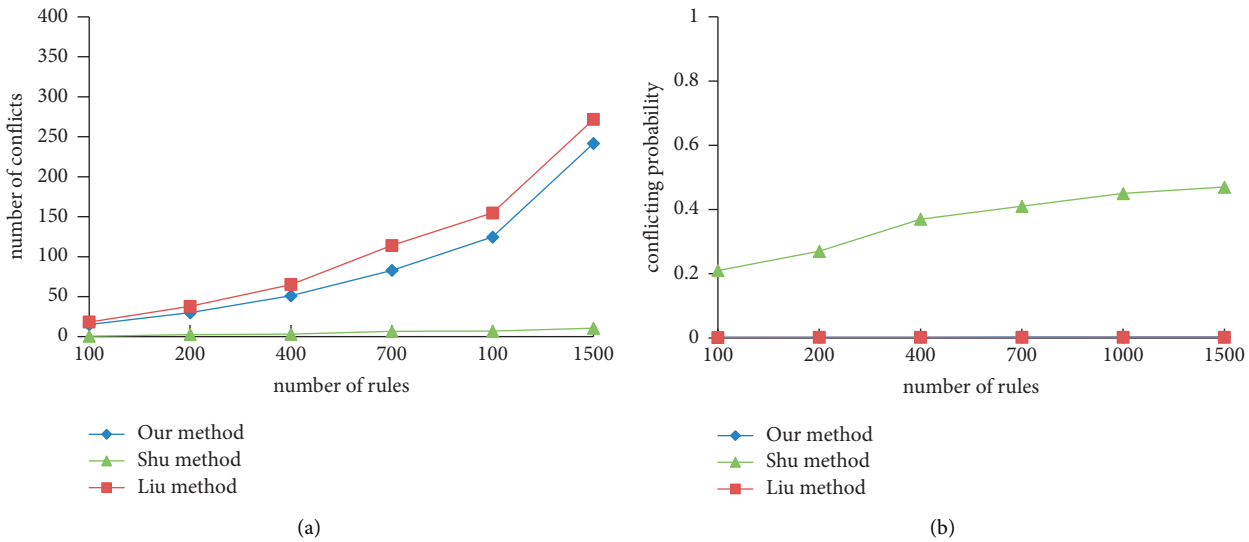

Figure 6: Performance comparisons when $|RL| = 11$. (a) Conflict number. (b) Conflict probability.

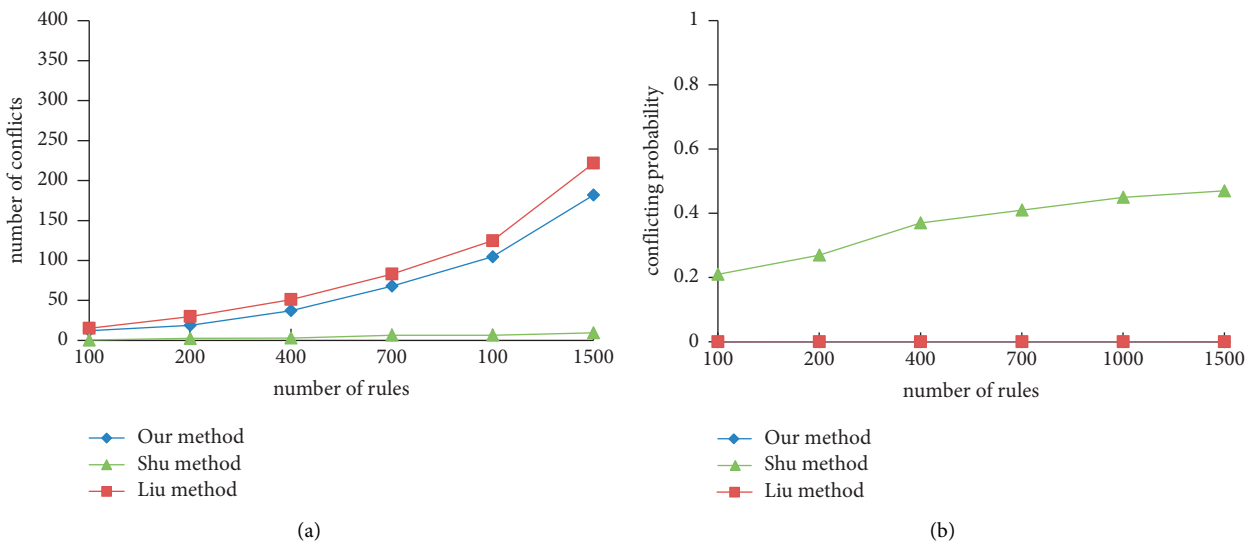

Figure 7: Performance comparisons when $|RL| = 15$. (a) Conflict number. (b) Conflict probability.

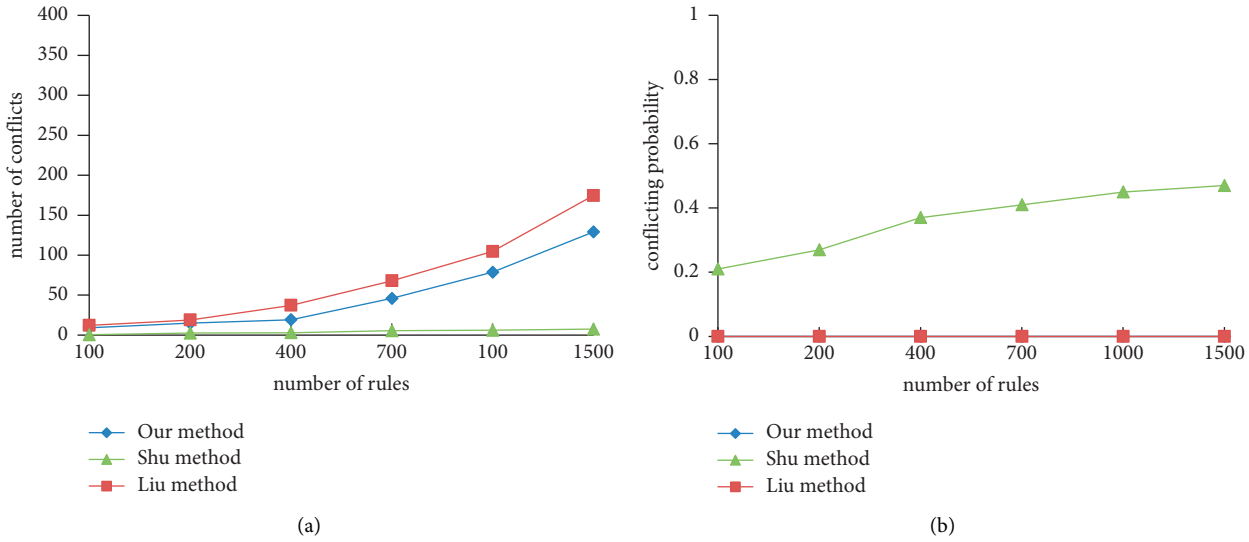

FIGURE 8: Performance comparisons when $|RL| = 20$. (a) Conflict number. (b) Conflict probability.

the Shu method from the viewpoint of conflicting probability. Furthermore, it is observed that the varying length of the attribute-expression conditions has a greater effect than that of the policy scale. Thus, it is also advised to use as many attribute conditions as possible, in order to improve the policy quality while reducing the conflicting probability.

5. Conclusions

A novel policy evaluation method, called ABPE_CSL&CD, was proposed in this study. According to the requirement descriptions of usage scenarios, we first utilized the attribute-based constraints specification language to formulate and specify the conflict relations among attributes, proposed the satisfiability of conflict relations, and categorized the conflicting rules into two classifications. Then, we presented the evaluation criteria on conflicting rules and attribute-conflict relations and proposed a novel algorithm for detecting conflicts. As a result, the proposed method flexibly suited the organizational requirements and comprehensively detected the ABAC conflict problems. The experiments on the real and synthetic datasets demonstrated that they could address the stated problems of improving the policy quality while reducing the conflicting number and conflicting probability. Our future work will focus on studying how to implement the ABPE_CSL&CD in other system scenarios such as the IoT, blockchain, and wireless sensor networks.

Data Availability

All the underlying data used to support the results of the study are included within the article.

Conflicts of Interest

The author declares no conflicts of interest.

Acknowledgments

This work was supported by the Key Scientific Research Project of Henan Province University.

References

[1] W. Sun, H. Su, and H. Xie, "Policy-engineering optimization with visual representation and separation-of-duty constraints in attribute-based access control," *Future Internet*, vol. 12, no. 10, p. 164, 2020.

[2] W. Sun, "Hybrid role-engineering optimization with multiple cardinality constraints using natural language processing and integer linear programming techniques," *Mobile Information Systems*, vol. 23, p. 1, 2022.

[3] S. Chakraborty, R. Sandhu, and R. Krishnan, "On the Feasibility of Attribute-Based Access Control Policy Mining," in *Proceedings of the 20th IEEE International Conference On Information Reuse And Integration For Data Science*, pp. 245–252, Los Angeles, CA, USA, August 2019.

[4] D. Servos and S. L. Osborn, "Current research and open problems in attribute-based access control," *ACM Computing Surveys*, vol. 49, no. 4, pp. 1–45, 2017.

[5] M. Narouei, H. Khanpour, H. Takabi, N. Parde, and R. D. Nielsen, "Towards a top-down policy engineering framework for attribute-based access control," in *proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, pp. 103–114, Indianapolis, IN, USA, June 2017.

[6] D. Mocanu, F. Turkmen, and A. Liotta, "Towards ABAC Policy Mining from Logs with Deep Learning," in *Proceedings of the 18th International Multiconference*, pp. 124–128, Ljubljana, Slovenia, October, 2015.

[7] Z. Xu and S. D. Stoller, "Mining attribute-based access control policies," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 533–545, 2015.

[8] S. Das, S. Sural, J. Vaidya, and V. Atluri, "Poster: using gini impurity to mine attribute-based access control policies with environment attributes," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pp. 213–215, Indianapolis, IN, USA, 2018 June.

[9] S. Das, S. Sural, J. Vaidya, V. Atluri, and G. Rigoll, "VisMAP: Visual Mining of Attribute-Based Access Control Policies," in *Proceedings of the 15th International Conference on Information Systems Security*, pp. 16–20, Hyderabad, India, December, 2019.

[10] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, pp. 852–869, 1999.

[11] J. C. Royer and A. S. De Oliveira, "AAL and static conflict detection in policy," *International Conference on Cryptology and Network Security*, pp. 367–382, Springer, Heidelberg, Germany, 2016.

[12] A. A. Jabal, M. Davari, E. Bertino et al., "Methods and tools for policy analysis," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–35, 2019.

[13] M. St-Martin and A. P. Felty, "A Verified Algorithm for Detecting Conflicts in XACML Access Control Rules," in *proceedings of the 5th ACM SIGPLAN Conference On Certified Programs And Proofs*, pp. 166–175, ACM, St. Petersburg FL USA, January 2016.

[14] M. Rezvani, D. Rajaratnam, A. Ignjatovic, M. Pagnucco, and S. Jha, "Analyzing XACML policies using answer set programming," *International Journal of Information Security*, vol. 18, no. 4, pp. 465–479, 2019.

[15] G. Zheng and Y. Xiao, "A Research on Conflicts Detection in ABAC Policy," in *7th International Conference On Computer Science And Network Technology*, pp. 408–412, IEEE, Dalian, China, October 2019.

[16] C. chun Shu, E. Y. Yang, and A. E. Arenas, "Detecting conflicts in ABAC policies with rule reduction and binary-search techniques," in *Proceedings of the 2009 IEEE International Symposium on Policies for Distributed Systems and Networks*, pp. 182–185, IEEE, London, UK, July 2009.

[17] G. Liu, W. Pei, Y. Tian, C. Liu, and S. Li, "A novel conflict detection method for ABAC security policies," *Journal of Industrial Information Integration*, vol. 22, Article ID 100200, 2021.

[18] A. Roy, S. Sural, A. K. Majumdar, J. Vaidya, and V. Atluri, "Enabling workforce optimization in constrained attribute based access control systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1901–1913, 2021.

[19] X. Jin, R. Krishnan, and R. Sandhu, "A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC," in *26th Annual IFIP WG 11.3 Conference On Data And Applications Security And Privacy XXVI*, pp. 41–55, Paris, France, July 2012.

[20] K. Z. Bijon, R. Krishnan, and R. Sandhu, "Towards an attribute based constraints specification language," in *Proceedings of the 2013 International Conference on Social Computing*, pp. 108–113, Washington, DC, USA, September, 2013.

[21] N. Helil and K. Rahman, "Attribute based access control constraint based on subject similarity," in *Proceedings of the 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications*, pp. 226–229, IEEE, Ottawa, ON, Canada, September 2014.

[22] S. Jha, S. Sural, V. Atluri, and J. Vaidya, "Specification and verification of separation of duty constraints in attribute-based access control," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 897–911, 2018.

[23] M. Alohaly, H. Takabi, and E. Blanco, "Towards an Automated Extraction of ABAC Constraints from Natural Language Policies," in *Proceedings of the 34th IFIP TC 11 International Conference On ICT Systems Security And Privacy Protection*, pp. 105–119, Lisbon, Portugal, June, 2019.

[24] J. B. Roberto and S. C. Robert, "Using CSP Look-Back Techniques to Solve Real-World SAT Instances," in *Proceedings of the 14th National Conference On Artificial Intelligence And Ninth Innovative Applications Of Artificial Intelligence Conference*, pp. 27–31, Providence, Rhode Island, USA, July, 1997.