基于属性轻量级可重构的访问控制策略

谢绒娜[1]，李晖[1]，史国振[2]，郭云川[3]

（1. 西安电子科技大学网络与信息安全学院，陕西 西安 710071；
2. 北京电子科技学院电子与通信工程系，北京 100070；3. 中国科学院信息工程研究所，北京 100093）

摘 要：针对复杂网络环境下访问控制策略冗余与冲突检测、访问控制策略评估的效率面临的严峻挑战，提出了基于属性轻量级可重构的访问控制策略。以基于属性的访问控制策略为范例，根据访问控制策略中的操作类型、主体属性、客体属性和环境属性将基于属性的访问控制策略划分为多个不相交的原子访问控制规则，并通过与、或等逻辑关系构成的代数表达式，将原子访问控制规则重构出复杂访问控制策略；提出原子访问控制规则冗余与冲突检测方法，将复杂访问控制策略分解为等效的原子访问控制规则和代数表达式，通过对等效的原子访问控制规则和代数表达式进行冗余与冲突检测实现对复杂访问控制策略进行冗余与冲突检测；从时间复杂度和空间复杂度 2 个不同角度对等效转化的访问控制策略进行评估。结果表明，所提方法大大降低了访问控制策略的长度、数量和复杂度，提高了访问控制策略冗余与冲突检测的效率以及访问控制策略评估的效率。

关键词：轻量级；可重构；原子访问控制规则；代数表达式；等效转化

中图分类号：TP302

文献标识码：A

doi:10.11959/j.issn.1000−436x.2020035

Attribute-based lightweight reconfigurable access control policy

XIE Rongna[1], LI Hui[1], SHI Guozhen[2], GUO Yunchuan[3]

1. School of Cyber Engineering, Xidian University, Xi’an 710071, China
2. Department of Electronics and Communication Engineering, Beijing Electronic Science and Technology Institute , Beijing 100070, China
3. Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

Abstract: Aiming at the severe challenges of access control policy redundancy and conflict detection, the efficiency of access control policy evaluation in complex network environment, an attribute-based lightweight reconfigurable access control policy was proposed. Taking the attribute-based access control policy as an example, the attribute-based access control policy was divided into multiple disjoint atomic access control rules according to the operation type, subject attribute, object attribute, and environment attribute in the access control policy. Complex access control policies were constructed through atomic access control rules and an algebraic expression formed by AND, OR logical relationships. A method for redundancy and collision detection of atomic access control rules was proposed. A method was proposed for decompose a complex access control policy into equivalent atomic access control rules and an algebraic expression. The method for redundancy and collision detection of complex access control policies were proposed through redundancy and collision detection of equivalent atomic access control rules and algebraic expressions. From time complexity and space complexity, the efficiency of the equivalent transformation access control policy was evaluated. It showes that the reconstruction method for access control policy greatly reduces the number, size and complexity of access control policy, improves the efficiency of access control policy redundancy and collision detection, and the efficiency of access control evaluation.

Key words: lightweight, reconfigurable, atomic access control rule, algebraic expression, equivalent transformation

收稿日期：2019−11−18；**修回日期**：2019−12−16

通信作者：史国振，sgz1974@163.com

基金项目：国家重点研发计划基金资助项目（No.2017YFB0802705, No.2016QY06X1203）；国家自然科学基金资助项目（No.61672515）

Foundation Items: The National Key Research and Development Program of China (No.2017YFB0802705, No. 2016QY06X1203), The National Natural Science Foundation of China (No.61672515)

1　引言

访问控制策略用来保护系统和网络中敏感资源被合法访问和使用，创建正确的满足安全需求的访问控制策略是保证系统安全、网络安全的前提和基础。复杂网络环境中包含多个安全域和各种复杂的分布式信息系统，不同的安全域和系统千差万别，安全需求也不同。复杂的网络环境下，描述出正确的、满足安全需求的访问控制策略是保证系统安全、网络安全急需解决的问题。

随着系统复杂度的增加和安全需求的差异，访问控制策略的数量、长度和复杂度急剧增加，这势必造成访问控制策略冗余与冲突检测难度增加，访问控制策略评估的效率急剧下降。另一方面，在计算资源、存储资源和能耗受限的设备中，访问控制策略的长度、数量和复杂度都受到很大限制。如何生成轻量级、不存在冗余和冲突的访问控制策略是复杂网络环境中访问控制策略生成面临的挑战。

在访问控制策略语言方面，研究人员先后提出了 Ponder、安全策略语言（SPL, security policy language）、可扩展的访问控制标记语言（XACML, extensible access control markup language）[1-3]。XACML 是一种采用 XML 格式的通用访问控制策略语言，它的通用性使各系统访问控制策略得到标准化，受到了学术界和工业界的广泛关注。然而，XACML 面临的最大挑战是策略部署和实施，其部署和实施的困难性导致 XACML 的应用受到一定的限制。为降低访问控制策略数量，文献[4-5]提出了访问控制策略组合方法，把 2 个不同的访问控制策略组合成一个等效的访问控制策略。文献[6]提出了基于主体、客体属性，将访问控制列表（ACL, access control list）生成等效的基于属性的访问控制策略，降低了访问控制策略的数量和复杂度。为提高访问控制策略评估效率，文献[7]通过把策略中复杂的逻辑表达式转换为决策树的方式来提高访问控制策略评估的效率。针对访问控制策略冲突问题，研究人员先后提出了基于有向无环图模型、概念格、策略决策树等方法实现访问控制策略的冲突检测。姚键等[8]提出了一种基于有向图模型的冲突检测机制，解决了冲突检测中形式化证明过于复杂的问题。李瑞轩等[9]提出了最小代价和字典编辑优选 2 种基于优先级的冲突消解方法，解决了策略非一致性冲突问题。但上述文献在轻量级可重构的访问控制策略生成、冗余与冲突检测、评估等方面并没有提出很好的解决方法。

针对上述问题，本文提出了基于属性轻量级可重构的访问控制策略。基于操作类型、主体、客体和环境属性将访问控制策略分解为多个不相交的原子访问控制规则，通过代数表达式对原子访问控制规则进行复合生成复杂访问控制策略，降低了访问控制策略数量、尺寸和复杂度，提高了访问控制策略冲突和冗余检测的效率以及访问控制策略评估的效率。

本文主要贡献如下。

1) 提出了根据访问控制策略中的操作类型、主体属性、客体属性和环境属性将基于属性的访问控制策略划分为多个不相交的原子访问控制规则，给出了基于属性的原子访问控制规则范式描述以及原子访问控制规则冗余与冲突检测的方法。

2) 提出了利用与、或等逻辑运算构造出代数表达式，通过原子访问控制规则和代数表达式重构出复杂访问控制策略的方法。

3) 提出了将复杂访问控制策略分解为等效的原子访问控制规则和代数表达式，通过对等效的原子访问控制规则和代数表达式进行冗余与冲突检测实现对复杂访问控制策略的冗余与冲突检测，提高了访问控制策略冗余与冲突检测的效率。

2　相关工作

访问控制策略语言定义了访问控制策略描述语言的语义和语法，文献[1]提出的 SPL 是一种事件驱动的策略语言。伦敦大学的 Policy 小组在 2000 年提出了一种面向对象的策略表示语言——Ponder[2]，与其他策略表示语言相比，Ponder 具有一定的通用性。在自主型访问控制模型中，一般使用访问控制列表和访问能力表来表述主体、客体与访问权限之间的对应关系。XACML[3]是采用 XML 格式的通用访问控制策略描述语言。在策略表达上，XACML 结构清晰且具有很大的灵活性，它将安全规则表示为主体、客体、行为和约束 4 个主要属性的属性值集合，通过逻辑操作把属性集合连接在一起实现对访问控制请求授权。同时 XACML 中包含了规则联合算法来解决安全策略中不同安全规则可能造成的冲突，以保证每个访问请求只得到一个最终授权结果。

基于属性的访问控制策略面临的最大挑战是策略部署和实施[10]。为提高基于属性访问控制策略部署实施的效率，基于属性的访问控制策略挖掘成为近期的一个研究热点，并取得了不少成果[6,11-12]。文献[6]提出了从 ACL 中自动挖掘基于属性访问控制策略的算法。该算法基于用户和权限的关系、用户和客体的属性值，通过用约束替换属性表达式中的连接词来建立关系的方法进行访问控制规则挖掘，并通过合并和简化方法来提高访问控制规则的质量。Iyer 等[11]给出了基于属性访问控制策略中肯定授权和否定授权规则的挖掘方法。Chakraborty 等[12]根据操作、主体和客体属性把基于属性的访问控制规则划分为多个不相交的规则集合。基于属性的访问控制策略的数量、长度同样也会影响访问控制策略实施的效率。Bonatti 等[13]提出了利用代数表达式来描述安全策略，以及把复杂访问控制策略形式化为代数的方法。针对访问控制策略复合问题，Rao 等[4]提出了通过 3 个二元操作和 2 个一元操作组成代数表达式的方式实现复杂访问控制策略细粒度的复合。文献[5]提出了通过 3 个二元操作把多个访问控制策略复合成一个丰富的访问控制策略，实现对于任意访问控制请求通过复合前后访问控制策略评估得到相同的访问控制结果，减少了访问控制策略的数量。针对使用 XACML 描述的访问控制策略评估效率的问题，文献[7]提出了 XACML 逻辑模型和决策图的方法，通过语义分析把策略中复杂的逻辑表达式转换为决策树来提高访问控制决策的效率。

在策略冲突检测和消解方面，不少文献也取得了许多成果。Lupu 等[14]研究了 Ponder 语言中策略形式的一致性问题，在进行策略冲突检测时，将策略分解为主体、客体和行为 3 个要素。当 2 条策略的某个要素的作用域相互覆盖时，就会引起符号冲突。姚健等[8]研究了分布式系统中元素之间的关系，并统一抽象成有向无环图模型，提出了一种应用该模型检测分布式系统中安全策略冲突的定量方法。针对 XACML 描述的各种复杂访问控制策略不同规则之间存在的冲突问题，St-Martin 等[15]提出了冲突检测算法并进行了正确性证明。

上述文献对于如何生成轻量级可重构的访问控制策略，以及重构后策略的冗余与冲突检测并没有给出很好的解决方案。

3 基于属性访问控制策略生成与检测

3.1 基于属性访问控制策略

基于属性访问控制策略根据主体、客体和环境属性进行访问授权。

定义 1 S 为主体 s 集合，AS 为主体属性 atts 集合，VS 为主体属性值 values 集合，values 包括单值属性值、多值属性值和区间属性值。

主体属性对可以描述为二元组<atts, values>。atts=SAE(s)表示主体属性的表达式，简称主体属性，values=e(atts)表示主体属性 atts 的属性值。比如，主体 s_1 的部门（department）是“部门 A”，那么 $atts_1$=SAE(s_1)=“department”，$values_1$=e($atts_1$)=“部门 A”。主体 s_1 的属性对可以表示为<department,“部门 A”>。

主体属性值除了单值外还包括多值和区间，比如不同场景下主体 s_1 的角色（role）不同，主体 s_1 的第二个属性 role，$atts_2$=SAE(s_1)=“role”, $values_2$=e($atts_2$)={“administrator”, “teacher”, “father”}。主体 s_1 的角色属性对可以表示为<role, {“administrator”, “teacher”, “father”}>。主体属性值还包括区间情况，比如主体 s_1 的上班时间 t 为 8:00—17:00，主体 s_1 的第三个属性对可以表示为<t, 8:00≤t≤17:00}>或者<t, [8:00,17:00]>。

按照同样的方法可以定义客体属性和环境属性。

定义 2 O 为客体 o 集合，AO 为客体属性 atto 集合，VO 为客体属性值 valueo 集合，valueo 包括单值属性值、多值属性值和区间属性值。

客体属性对可以描述为二元组<atto, valueo>。atto=OAE(o)表示客体属性的表达式，valueo=e(atto)表示客体属性 atto 的值。

定义 3 C 为环境 c 集合，AC 为环境属性 attc 集合，VC 为环境属性值 valuec 集合，valuec 包括单值属性值、多值属性值和区间属性值。

环境属性对可以描述为二元组<attc,valuec>。attc=CAE(c)表示环境属性的表达式，valuec= e(attc)表示环境属性 attc 的值。

基于属性的访问控制策略 policy 是一个八元组<OP, AS, AO, AC, VS, VO,VC,Rules>。其中，OP 是操作 op 的集合，Rules 是规则 rule 的集合。基于属性的访问控制规则可以表示为<effect,<att,value>, op>。effect 表示该条规则是肯定授权还是否定授权，effect={“permit”, “deny”}，permit 表示肯

定授权，deny 表示否定授权。<att,value>表示属性、属性值对。例如，规则 rule=<permit,department={“A”, “B”}∧location=“D://”,read>表示部门 A 或者部门 B 的员工可以阅读 D 盘上的文件。

访问控制规则是基于属性访问控制策略最核心的部分，基于属性的访问控制规则的巴科斯范式（BNF, Backus-Naur form）描述表示如下。

Rule::= Rule∨Rule| Ruleexp
Ruleexp::=<effect, attexp,op>
effect::= “permit”|“deny”
attexp::=attexp| anyofattexp
anyofattexp::=anyofattexp|allofattexp
allofattexp::=allofattexp∧atomicattexp
atomicattexp::=<att,value>

3.2　基于属性的原子访问控制规则

针对基于属性访问控制策略存在的冗余与冲突检测困难、访问控制请求评估效率低等问题，本文提出了基于属性的原子访问控制规则（atomicRule）的概念，以下简称原子规则。根据访问控制策略中的访问控制规则包含的操作类型、主体属性、客体属性、环境属性等将访问控制策略划分为多个不相交最小原子规则，然后通过代数表达式将原子规则复合成多个复杂访问控制策略。

基于属性的原子规则可以表示成<effect, <att, value>, op>四元组的形式。假设基于属性的原子规则包含 m 个属性对，即<att_1, $value_1$>, <att_2, $value_2$>, …, <att_m,$value_m$>。$value_i$（$0<i\leqslant m$）包括单值属性值、多值属性值和区间属性值。原子规则满足如下条件：1) 操作 op 是单操作，假设操作集合 OP 包含 k 个操作，OP={op_1,op_2,…,op_k}，op 是操作集合 OP 中的一个操作；2) 访问控制效果 effect 只能是{“permit”,“deny”}中的一个，即 effect=“permit”或者 effect=“deny”；3) 属性对<att,value>是最小的集合，即减少其中任意一个属性对<att_i,$value_i$>，原子规则将不能得到有效的访问控制评估结果。比如规则 $rule_1$=<permit, department ={“A”, “B”}∧location = “D://”, read >就是一个原子规则，访问控制效果 effect =“permit”, op=read，属性对 department={“A”, “B”}和 location = “D://”去掉其中任意一个都不能得到有效的访问控制评估结果。规则 $rule_2$ = <permit, (department ={“A”, “B”}∨role = administrator)∧location = “D://”, read >就不是一个原子规则，属性对<department, {“A”, “B”}>和<role, administrator>满足其中任意一个条件都可以得到有效的访问控制评估效果。$rule_2$ 可以分解为 $rule_1$ 和 $rule_3$ 这 2 个原子规则。$rule_3$ = <permit, (role = administrator)∧location = “D://”, read >，$rule_2$= $rule_1$∨$rule_3$。基于属性的原子规则的 BNF 描述表示如下。

atomicRuleexp ::=<effect, allofattexp, op>
effect::= “permit”|“deny”
allofattexp::=allofattexp∧atomicattexp
atomicattexp::=<att,value>

3.3　基于属性原子访问控制规则的冗余与冲突检测

利用上述原子规则的 3 个条件可以判断一个访问控制规则是否为原子规则，但原子规则之间还存在冗余和冲突。比如 $rule_4$ = <permit, (department ={“B”, “C”})∧location = “D://”, read >。$rule_1$ 和 $rule_4$ 均为原子规则，但它们之间存在冗余。$rule_5$ = <deny, (department = “C”) ∧location = “D://”, read>。$rule_4$ 和 $rule_5$ 同样都为原子规则，但它们之间存在冲突。下面讨论原子规则中同一类型操作的不同规则之间的冗余和冲突检测。

1) 原子规则的冗余检测

对于任意 2 个规则 $rule_i$ 和 $rule_j$，分别包含 n_i 和 n_j 个属性对，$rule_i$=<$effect_i$, <att_1,$value_1$>, <att_2,$value_2$>, … , <att_{n_i},$value_{n_i}$>, op_i>, $rule_j$ = <$effect_j$, <att_1,$value_1$>, <att_2,$value_2$>, … ,<att_{n_j},$value_{n_j}$>, op_j>。如果规则 $rule_i$ 和 $rule_j$ 包含的操作和访问控制效果相同，即 op_i=op_j，$effect_i$=$effect_j$，属性对个数、属性和属性值均相同，即 n_i=n_j，$rule_i.att_k$ =$rule_j.att_k$，$rule_i.value_k$=$rule_j.value_k$ (k=(1, …, n_i))，那么 $rule_i$ 和 $rule_j$ 为 2 个相同的规则。判断属性值是否相同包括判断单值、多值和区间值是否相同。如果属性值为单值，直接判断 2 个值是否相等；如果属性值为多值或者区间，判断表示属性值的集合是否相等。如果规则 $rule_i$ 和 $rule_j$ 的操作和访问控制效果相同，属性对个数和属性也相同，但存在 2 个属性值不相同，即 $rule_i.value_k$≠$rule_j.value_k$，那么规则 $rule_i$ 和 $rule_j$ 存在冗余，$rule_i$ 和 $rule_j$ 可以合并为一个新的原子规则 rule。如果 $value_k$ 为单值，新的原子规则 rule 的属性值 $value_k$ 改为多值，$rule.value_k$ ={ $rule_i.value_k$, $rule_j.value_k$}；如果 $value_k$ 为多值或者区间，那么 $rule.value_k$=$rule_i.value_k$∪$rule_j.value_k$。对于原子规则的冗余检测可以采用算法 1 进行。

算法 1　RedundancyDetect($rule_1$, $rule_2$)

输入 原子规则 $rule_1$ 和 $rule_2$

判断规则 $rule_1$ 和 $rule_2$ 是否存在冗余，如果不存在冗余，返回 false；如果存在冗余，返回 true，同时将 $rule_1$ 和 $rule_2$ 合并为另一个新的原子规则 rule。

//得到规则 $rule_1$ 和 $rule_2$ 的操作 op_1 和 op_2

op_1=GetRuleOperator($rule_1$)；

op_2=GetRuleOperator($rule_2$)；

if($op_1 \neq op_2$)

return false;

//得到 $rule_1$ 和 $rule_2$ 的访问控制效果 $effect_1$ 和 $effect_2$

$effect_1$=GetRuleEffect($rule_1$)；

$effect_2$=GetRuleEffect($rule_2$)；

if($effect_1 \neq effect_2$)

return false;

//得到 $rule_1$ 和 $rule_2$ 的属性对个数

n_1=GetRuleAttNumber($rule_1$);

n_2=GetRuleAttNumber($rule_2$);

if($n_1 \neq n_2$)

return false;

//得到 $rule_1$ 和 $rule_2$ 的属性对

$attSet_1$= GetRuleAttSet ($rule_1$);

$attSet_2$= GetRuleAttSet ($rule_2$);

for(i=1;$i \leqslant n_1$;i++)

{

if

{

if($attSet_1.att_i \neq attSet_2.att_i$)

return false;

}

}

rule=new Rule( );

$rule.attSet=rule_1.attSet_1$;

for(i=1;$i \leqslant n_1$;i++)

{

if

{

//如果属性值不相同，将 2 个属性值合并，

if($attSet_1.value_i \neq attSet_2.value_i$)

$attSet.value_i = attSet_1.value_i \cup attSet_2.value_i$;

}

}

return true；

函数 GetRuleOperator(rule)为得到规则 rule 对应的操作 op，GetRuleEffect(rule)为得到规则 rule 的访问控制效果 effect，GetRuleAttNumber(rule)为得到规则 rule 属性对个数，GetRuleAttSet(rule)为得到规则 rule 的属性对<att,value>。

2) 原子规则的冲突检测

对于任意 2 个规则 $rule_i$ 和 $rule_j$，如果规则 $rule_i$ 和 $rule_j$ 包含的操作相同、属性个数和属性相同、属性值相同或者存在交集，而访问控制效果相反，那么规则 $rule_i$ 和 $rule_j$ 存在冲突。在对属性值进行判断时，对于单值属性值，直接判断 2 个属性值是否相同，即 $rule_i.value_k=rule_j.value_k$；对于多值属性值或者区间属性值，判断 2 个属性值是否存在交集，即 $rule_i.value_k \cap rule_j.value_k \neq \Phi$。函数 ConflictDetect($rule_1$, $rule_2$)用来检测规则是否存在冲突，存在冲突返回 true，不存在冲突返回 false。

3.4 基于属性原子访问控制规则的复合

下面讨论如何由原子规则复合出语义丰富的复杂的访问控制策略。本文提出了通过与（AND, ∧）、或（OR, ∨）布尔操作组成的布尔函数，对原子规则进行复合得到语义丰富的复杂访问控制策略。

1) AND（∧），与操作要求所有的访问控制规则都同意授权时，才允许主体对客体进行对应的操作。例如，$rule=rule_1 \wedge rule_2$，$dec_i$ 表示规则 $rule_i$ 的授权结果，$dec=dec_1 \wedge dec_2$。将允许授权 permit 编码为 1，当且仅当 dec_1=1 和 dec_2=1 时，dec=1。与操作表示最小的授权原则。

2) OR（∨），或操作表示当有一条访问控制规则同意授权时，就允许主体对客体进行对应的操作。例如，$rule=rule_1 \vee rule_2$，$dec=dec_1 \vee dec_2$。当 dec_1=1 或 dec_2=1 时，dec=1。或操作表示最大的授权原则。

令变量 x_i 表示规则 $rule_i$，变量 y 表示规则 rule，如果 $rule=rule_1 \wedge rule_2$，令 $y=f_1(x_1,x_2)=x_1 \wedge x_2=x_1x_2$，那么规则 rule 可以通过规则 $rule_1$、$rule_2$ 和代数表示式 $f_1(x_1,x_2)$复合而成。如果 $rule=rule_1 \vee rule_2$，令 $y=f_2(x_1,x_2)=x_1 \vee x_2= x_1+ x_2+ x_1 x_2$，那么规则 rule 可以通过规则 $rule_1$、$rule_2$ 和代数表示式 $f_2(x_1,x_2)$复合而成。如果规则 rule 是通过对 n 个规则 $rule_1,\cdots,rule_n$ 进行与、或操作复合而成，代数表达式 $y=f(x_1,\cdots, x_n)$表示规则 rule 中各个原子规则逻辑关系，规则 rule 可以通过代数表达式 $y=f(x_1,\cdots, x_n)$对规则 $rule_1,\cdots,rule_n$ 复合生成。令 dec_i 表示规则 $rule_i$ 的授权结果，其中同意授权 dec_i=1，否定授权 dec_i=0。令 $x_i=dec_i$，将

规则 $rule_i$ 的授权结果 dec_i 代入函数 $y=f(x_1,\cdots, x_n)$就可以得到规则 rule 授权结果。

从前面分析可以看出，通过原子规则和代数表达式 f 可以复合出语义丰富的复杂访问控制规则。对于复合访问控制规则的授权结果，通过将原子规则的授权结果代入代数表达式 f 中就可以得到复杂访问控制规则的授权结果。访问控制策略是由多个访问控制规则复合而成，对于访问控制策略同样可以采用原子规则和代数表达式的方式复合生成，访问控制授权结果也可以采用同样的方法得到。

对于复杂访问控制规则的冗余与冲突检测，可以通过对等效的原子规则和代数表达式进行冗余和冲突检测。如果 2 个访问控制规则等效的原子规则存在冗余或冲突，那么这 2 个访问控制规则一定存在冗余和冲突。对于等效的原子规则不存在冗余和冲突的访问控制规则，可以通过代数表达式进一步判断 2 个访问控制规则是否存在冗余和冲突。访问控制规则 $rule_i$ 和 $rule_j$ 对应的代数表达式为 f_i 和 f_j，如果不存在冗余和冲突，那么访问控制规则 $rule_i$ 和 $rule_j$ 不存在冗余和冲突，如果 f_i 和 f_j 存在冗余和冲突，那么访问控制规则 $rule_i$ 和 $rule_j$ 存在冗余和冲突。对于访问控制策略，可以采用类似的方法进行冗余和冲突检测。复杂访问控制策略冗余与冲突检测转化为对等效原子规则和对应代数表达式进行检测，大大降低了访问控制策略冗余和冲突检测的复杂度，提高了检测的效率。

4　基于属性访问控制策略的分解

根据第 3 节的分析可以看出，通过原子规则复合的方式提高了访问控制策略生成、冗余和冲突检测的效率。在实际信息安全系统中，系统管理员根据系统的安全需求进行访问控制策略的配置或者自动生成各种访问控制策略，系统管理员配置或者自动生成的访问控制策略一般都是复杂访问控制策略。本节重点讨论如何将复杂访问控制策略等效转化为原子规则和代数表达式的形式。

4.1　基于属性访问控制规则的分解与等效转化

基于属性访问控制策略包含一条或多条访问控制规则，下面首先讨论对访问控制策略中的一条访问控制规则进行分解的方法。基于属性的访问控制规则经需要过以下 3 个步骤等效转化为原子规则和代数表达式的形式。

Step1　算法 2 DecomposeRule( )根据访问控制规则 rule 包含的操作，把规则 rule 分解为访问控制规则集合 RuleOP 及对应的代数表达式 f，访问控制规则集合 RuleOP 中的每一个访问控制规则 Ruleop 只包含一种类型的操作。

Step2　算法 3 DecomposeRuleOp( )对访问控制规则 Ruleop 进行语义分析，将 Ruleop 分解为多个访问控制规则，生成访问控制规则集合 RuleSetOP 及对应的代数表达式 f_{op}。访问控制规则集合 RuleSetOP 中的访问控制规则 RuleSetop 只包含一种类型的操作和一种访问控制效果。

Step3　算法 4 DecomposeAtomicRule( )根据规则 RuleSetop 中主体属性、客体属性、环境属性的逻辑关系，将访问控制规则 RuleSetop 分解为原子规则集合，并对原子规则集合进行冗余和冲突检测，得到访问控制规则 RuleSetop 最终等效的原子规则集合 atomicRuleSet 以及对应代数表达式 f_{atomic}。

基于属性的访问控制规则的分解与等效转化如图 1 所示。

算法 2 DecomposeRule(rule,RuleOP,n)根据 rule 中包含的操作类型把规则 rule 分解为多个只包含一种操作类型的访问控制规则集合 RuleOP。n 为访问控制规则 rule 中操作类型的个数，也是最终生成的访问控制规则集合 RuleOP 中包含访问控制规则 Ruleop 的个数。规则 Ruleop 只包含一种操作类型，Ruleop=<effect,<att,value>,op>，op 是操作集合 OP 中的一类操作，effect 是{“permit”, “deny”}中的一个。

算法 2 首先调用函数 GetRuleOperator(rule)得到规则 rule 包含的所有操作集合 OP。然后针对操作集合中的每一个元素 op，调用函数 GetOPRule (rule, op)得到访问控制规则 rule 中操作 op 对应的访问控制规则 Ruleop。比如访问控制规则 rule 包

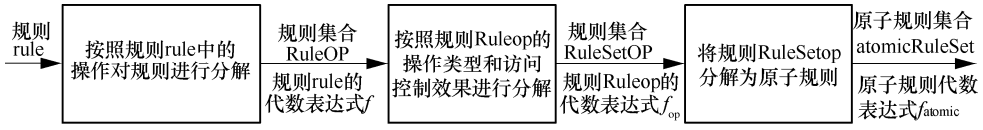

图 1　基于属性的访问控制规则的分解与等效转化

含读（read）操作和写（write）操作，rule 分解为 $\text{Rule}_{\text{read}}$ 和 $\text{Rule}_{\text{write}}$。最后函数 GetRuleFExp( )根据不同操作的访问规则在规则 rule 中的逻辑关系生成 rule 的代数表达式 f。例如，规则 rule=<permit, department={“A”, “B”}∧location=“D://”,read>∨<permit, department={“A”, “C”}∧location=“D://”, read>∨<permit,department={“A”, “C”}∧location=“D://”,write>，规则 rule 可以分解为读访问控制规则 $\text{Rule}_{\text{read}}$ 和写访问控制规则 $\text{Rule}_{\text{write}}$，$\text{Rule}_{\text{read}}$=<permit,department={“A”, “B”}∧location=“D://”,read>∨<permit,department={“A”, “C”}∧location=“D://”,read>，$\text{Rule}_{\text{write}}$=<permit, department={“A”, “C”}∧location=“D://”,write>。$f(x_1, x_2)= x_1 \vee x_2=x_1+x_2+x_1x_2$。其中 x_1 表示 $\text{Rule}_{\text{read}}$，$x_2$ 表示 $\text{Rule}_{\text{write}}$。通过访问控制规则 $\text{Rule}_{\text{read}}$、$\text{Rule}_{\text{write}}$ 和代数表达式 f 可以复合出访问控制规则 rule。

算法 2 DecomposeRule(rule,RuleOP,n)

输入 规则 rule

输出 不同操作对应的访问控制规则集合 RuleOP，RuleOP 中访问控制规则的个数 n，代数表达式 f

```
OP=Φ;
RuleOP=Φ;
//得到规则 rule 包含的操作；
OP=GetRuleOperator(rule);
n=0;
While(OP.isEmpty())
{
//获取操作集合 OP 中一个操作 op，并从集合 OP 删掉 op
op=Getop(OP);
//得到规则 rule 中操作 op 对应的访问控制规则
Ruleop=GetOPRule(rule,op);
n++;
RuleOP=AddRule(Ruleop);
}
GetRuleFExp(rule, RuleOP, f)
```

算法 3 DecomposeRuleOp(Ruleop,RuleSetOP, n_{op}, f_{op})对访问控制规则 Ruleop 的语义和逻辑关系进行分析，把规则 Ruleop 分解为 n_{op} 个等效的、只包含一种访问控制效果的、互不相交的访问控制规则集合 RuleSetOP，f_{op} 为访问控制规则 Ruleop 对应的代数表达式，通过访问控制规则集合 RuleSetOP 和代数表达式 f_{op} 可以复合出访问控制规则 Ruleop。访问控制规则集合 RuleSetOP 中的每一个规则 RuleSetOP[i]只包含一种操作类型和一种访问控制效果，RuleSetOP[i]= <effect,<att,value>, op>，op 是操作集合 OP 中的一个元素，effect 是{“permit”, “deny”}中的一个。

算法 3 首先调用函数 GetPermitRule(Ruleop)得到规则 Ruleop 中肯定授权规则集合 rulePSet，调用函数 GetRuleFExp(Ruleop, rulePSet, $f_{\text{op}P}$)得到规则 Ruleop 中肯定授权规则的代数表达式 $f_{\text{op}P}$。调用函数 GetDenyRule(Ruleop)得到规则 Ruleop 中否定授权规则集合 ruleDSet，调用函数 GetRuleFExp (Ruleop, ruleDSet, $f_{\text{op}D}$)得到规则 Ruleop 中否定授权规则的代数表达式 $f_{\text{op}D}$。算法 3 生成与规则 Ruleop 等效互不相交的访问控制规则集合 RuleSetOP，RuleSetOP 为 rulePSet 和 ruleDSet 并集，代数表达式 $f_{\text{op}}= f_{\text{op}P} \cup \sim f_{\text{op}D}$。例如，$\text{rule}_{\text{read}}$= <permit, department={“A”, “B”}∧location=“D://”, read>∨<permit,department={“A”, “C”}∧location=“D://”,read>，通过算法 3 规则 $\text{rule}_{\text{read}}$ 分解为 RuleSetread[1]= <permit,department={“A”, “B”}∧location=“D://”,read>和 RuleSetread[2]= <permit,department={“A”, “C”}∧location=“D://”, read>。$f_{\text{read}}(x_1,x_2)=x_1 \vee x_2=x_1+x_2+x_1x_2$。

算法 3 DecomposeRuleOp (Ruleop, RuleSetOP,n_{op},f_{op})

输入 规则 Ruleop

输出 与规则 Ruleop 等效的访问控制规则集合 RuleSetOP，RuleSetOP 中包含的访问控制规则数量 n_{op}，对应的代数表达式 f_{op}

```
//得到规则 Ruleop 中操作 op 对应的肯定授权规则
rulePSet=GetPermitRule(Ruleop);
n_op=0;
if(rulePSet≠Φ)
    n_op =GetRuleNumber(rulePSet);
GetRuleFExp(Ruleop, rulePSet, f_opP);
//得到规则 Ruleop 对应的否定授权规则
ruleDSet=GetDenyRule(Ruleop);
if(ruleDSet≠Φ)
    n_1= GetRuleNumber (rulePSet);
    n_op=n_op+ n_1;
GetRuleFExp(Ruleop, ruleDSet, f_opD);
```

RuleSetOP= rulePSet∪ruleDSet;

$f_{op}=f_{opP}\cup\sim f_{opD}$

通过算法 3 分别对访问控制集合 RuleOP 中 n 个访问控制规则 Ruleop 进行分解等效转化，分别得到各个访问控制规则 Ruleop 对应的互不相交的访问控制规则集合 RuleSetOP 及复合代数表达式 f_{op1}, $f_{op2},\cdots,f_{opn}$，访问控制规则 rule 对应的代数表达式为 $f(f_{op1},f_{op2},\cdots,f_{opn})$。

目前，访问控制规则集合 RuleSetOP[i]中各个访问控制规则 RuleSetop 并不是原子规则，各个规则之间还存在冗余和冲突，算法 4 调用函数 DecomposeAtomicRule (RuleSetop, conflictflag, atomicRuleSet, f_{atomic}, n_{atomic})，把只包含一种操作类型和一种访问控制效果的规则 ruleSetop 分解成等效的、互不相交的原子规则集合 atomicRuleSet，并检测各规则之间是否存在冗余和冲突，如果存在冲突，将冲突标记 conflictflag 置为 true；如果存在冗余，将原子规则集合 atomicRuleSet 中存在冗余的规则进行合并，得到 RuleSetop 最终原子规则集合及代数表达式 f_{atomic}。

算法 4 调用函数 GetRuleMinAttSet (RuleSetop, attSet, n_{atomic})生成规则 RuleSetop 最小属性对集合 attSet。函数 GetRuleMinAttSet( )首先对规则 ruleSetop 中属性对进行语义分析，根据各属性对之间的逻辑关系将属性对分解成 n_{atomic} 个最小属性对集合。具体分析时，先扫描规则中属性对或操作符，再根据或操作符把属性对分解成多个最小的属性对集合。例如，规则 RuleSetop 中属性对为 (<att_1,$value_1$>∧<att_2,$value_2$>)∨(<att_3,$value_3$>∧ <att_4, $value_4$>)，函数 GetRuleMinAttSet 得到属性对集合 attSet， attSet={{<att_1,$value_1$>∧ <att_2, $value_2$>}, {<att_3,$value_3$>∧<att_4,$value_4$>}}。attSet 共包含 2 个元素， attSet[1]= {<att_1,$value_1$> ∧ <att_2,$value_2$>}， attSet[2]={<att_3,$value_3$> ∧ <att_4,$value_4$>}。函数 AddRule (atomicRuleSet, <effect, attSet[i],op>)根据 <effect, attSet[i],op>生成规则并加入规则集合 atomicRuleSet 中。函数 GetRuleFExp (RuleSetop, atomicRuleSet, f_{atomic}) 分析原子规则集合 atomicRuleSet 中各个原子规则的逻辑关系得到原子规则集合 atomicRuleSet 各个原子规则的代数表达式 f_{atomic}。例如，规则 RuleSetop 分解成了 2 个原子规则，atomicRuleSet[1]= <effect, attSet[1] ,op>和 atomicRuleSet[2]= <effect, attSet[2], op>，规则 RuleSetop 对应的代数表达式 $f_{atomic}= x_1\vee x_2=x_1+x_2+x_1x_2$，$x_1$ 表示规则 atomicRuleSet[1]，x_2 表示规则 atomicRuleSet[2]。

目前生成原子规则集合 atomicRuleSet 中各原子规则之间可能还存在冗余和冲突，调用函数 ConflictDetect( )和 RedundancyDetect( )对原子规则集合 atomicRuleSet 各原子规则进行冗余和冲突检测。函数 SimplyRuleFExp(atomicRuleSet, f_{atomic}, n_{atomic})将原子规则集合中存在冗余的规则从代数表达式中删掉，并生成新的代数表达式 f_{atomic}。例如，规则 RuleSetop = (atomicRuleSet[1]∧atomicRuleSet[2])∨ atomicRuleSet[3] 对应的复合表达式为 $f_{atomic}=(x_1\wedge x_2)\vee x_3= x_1x_2+x_3+x_1x_2x_3$。如果原子规则 atomicRuleSet[1]和 atomicRuleSet[2]存在冗余，将 atomicRuleSet[1]和 atomicRuleSet[2]合并为一个新的原子规则放在 atomicRuleSet[1]，并将 atomicRuleSet[3]放在 atomicRuleSet[2]，原来的 atomicRuleSet[3]从原子规则集合删掉，新的代数表达式 $f_{atomic}=x_1\vee x_2= x_1+x_2+x_1x_2$，$n_{atomic}$ 由 3 更新为 2。例如，前文生成的读规则 RuleSetread[1] 和 RuleSetread[2] 存在冗余， RuleSetread[1] 和 RuleSetread[2] 可以合并为一个新的规则 RuleSetread[1]， RuleSetread[1]= <permit, department={“A”, “B”, “C”}∧location=“D://”, read>。

算法 4　DecomposeAtomicRule(RuleSetop, conflictflag, atomicRuleSet, f_{atomic}, n_{atomic})

输入　规则 RuleSetop

输出　与规则 RuleSetop 等效的、互不相交的、不存在冗余的原子规则集合 atomicRuleSet，对应的代数表达式 f_{atomic}，原子规则的个数 n_{atomic}

```
//根据 RuleSetop 中属性对的逻辑关系，将 RuleSetop 中的属性对分解成最小的属性对集合
attest = GetRuleMinAttSet (RuleSetop, attSet, n_atomic);
for(i=0;i<n_atomic;i++)
{
//把规则<effect,attSet[i],op>加入原子规则集合中
AddRule(atomicRuleSet,effect,attSet[i],op);
}
GetRuleFExp(RuleSetop, atomicRuleSet, f_atomic);
//原子规则集合冲突检测
conflictflag=ConflictDetect(atomicRuleSet);
if(conflictflag=true)
```

规则 RuleSetop 存在冲突；

//原子规则集合冗余检测 Redundancyflag= RedundancyDetect(atomicRuleSet);

SimplyRuleFExp(atomicRuleSet, f_{atomic},n_{atomic})

通过算法 4，对访问控制规则集合 RuleSetOP 中 n_{op} 个访问控制规则 RuleSetop 进行分解等效转化，分别得到各个访问控制规则 RuleSetop 对应的原子规则集合、复合代数表达式 f_{atomic1}, f_{atomic2},⋯, $f_{\text{atomic}n_{\text{op}}}$ 以及访问控制规则集合 RuleSetOP 对应的复合代数表达式 $f_{\text{op}}(f_{\text{atomic1}}, f_{\text{atomic2}},\cdots, f_{\text{atomic}n_{\text{op}}})$。

4.2 基于属性访问控制策略的分解与等效转化

基于属性访问控制策略是对多个访问控制规则复合而成。假设访问控制策略 P 是对 m 个访问控制规则通过复合函数 $f(x_1,x_2,\cdots,x_m)$复合而成，m 个访问控制规则对应的代数表达式分别为 $f_1,f_2,\cdots,f_m$。访问控制策略 P 的代数表达式 $f(x_1, x_2,\cdots, x_m)= f(f_1, f_2,\cdots,f_m)$。

5 基于属性可重构的访问控制策略的评估

本节分别从空间复杂度和时间复杂度对基于属性可重构的访问控制策略进行评估。

5.1 空间复杂度

假设系统的访问控制策略库中共有 n 个原子规则，n 元布尔函数函数 $f(x_1,x_2,\cdots,x_n)$共有 2^{2^n}，n 个原子规则可以重构出 2^{2^n} 个复杂访问控制策略。对于通过原子规则和代数表达式重构生成的访问控制策略库只需要存储 n 个原子规则和 2^{2^n} 个代数表达式。与访问控制策略相比，代数表达式占用的存储空间可以忽略不计，2^{2^n} 个复杂访问控制策略占用的空间与 n 个原子规则占用的空间是指数关系，优势显然而见。

5.2 时间复杂度

下面，从访问控制策略评估效率对本文提出的可重构的访问控制策略的时间复杂度进行定性分析。

访问控制策略的评估包括访问控制策略的匹配和访问控制请求评估两部分。对于基于属性的访问控制策略的评估需要根据访问请求中操作、主体属性、客体属性和环境属性从访问控制策略库一一匹配找到相关的访问控制策略，根据匹配到访问控制策略中的各个访问控制规则来判断是否可以对客体进行相关操作，并根据各个访问控制规则判断结果和组合算法得到访问控制请求的最终结果。从前面评估过程中看出，首先需要对访问控制策略库中的访问控制策略一一进行匹配，找到相关的访问控制策略。当策略库中访问控制策略数量比较大时，策略的匹配需要耗费大量的时间。从评估过程看，需要对匹配到访问控制策略中所有相关的访问控制规则一一进行判断。例如信息系统 S，配置有 100 个访问控制策略，100 个访问控制策略等效转化为 10 个原子规则。系统 S 的访问控制策略库由 100 个访问控制策略变成 10 个原子规则和 100 个代数表达式。当系统 S 接收到访问控制请求 R，根据访问控制请求中的操作、主体属性、客体属性和环境属性对 100 个访问控制策略一一匹配，找到最终的访问控制策略。例如访问控制请求为部门 A 的员工请求访问 D: //file1 文件。按照操作、主体属性、客体属性和环境属性的顺序进行匹配。假设首先从 100 个访问控制策略中找到与读操作有关的策略 80 个，与读操作相关的访问控制策略中与主体属性部门 A 有关策略 50 个，然后根据客体属性 D://file1 最终找到访问控制规则 rule，rule=<permit, department = {“A”, “B”}∧location= “D://”,read>∨<permit, department={“A”, “C”}∧location= “D://”,read>∨<permit, department={“A”,“C”}∧location=“D://”,write>。从访问控制策略匹配过程，假设进行一次匹配所花费的时间为 1 个单位时间，在上述信息系统 S 中最终匹配到访问控制规则 rule 所花费的时间为 100+80+50=230 个单位时间。

对本文提出的通过原子规则重构生成的访问控制策略进行匹配时，首先根据访问请求中的操作，筛选出和访问请求中相同操作的原子规则，然后根据主体属性、客体属性和环境属性从筛选的原子规则找到最终匹配的原子规则集合。对上述信息系统 S，10 个原子规则中有 5 个原子规则为读操作相关规则，与主体属性部门 A 相关的原子规则有 3 个，而与客体属性 D://file1 相关原子规则只有 1 个，找到最终原子规则 RuleSetread[1]= <permit, department={“A”, “B”, “C”}∧location=“D://”, read>。在 10 个原子规则中匹配到最终原子规则 RuleSetread[1]所花费的时间为 10+5+3=18 个单位时间。230 个单位时间远远大于 18 个单位时间。实际上，匹配原子规则花费的时间远远小于复杂访问控制策略，也就是说匹配到最终原子规则 RuleSetread[1]所花费的时间小于 18 个单位时间。

下面分析对访问控制请求评估所花费的时间。通过访问控制规则 rule 对访问控制请求 R 进行评估时，允许读 D://文件的主体为{“A”, “B”}，部

门 A 在允许的范围内，部门 A 的员工可以读文件 D://file1。继续扫描规则 rule，规则 rule 中对读操作的权限限制为<permit,department={“A”,“B”}∧location=“D://”,read>和<permit, department={“A”,“C”}∧location= “D://”,read>，它们之间为或的关系，当判断部门 A 的员工可以读文件 D: //file1 时，后续限制条件<permit, department ={“A”,“C”}∧location=“D://”,read>不需要再进行判断。继续扫描规则 rule，下面为写操作，与读操作无关，也不用判断，得到最终的访问控制结果为部门 A 的员工可以读文件 D://file1。通过原子规则 RuleSetread[1]对访问控制请求 R 进行评估时，允许读 D://文件的主体为{“A”,“B”,“C”}，部门 A 在允许的范围内，部门 A 的员工可以读文件 D://file1。从前面的分析过程发现，通过规则 rule 对访问控制请求 *R* 进行判断所花费的时间至少是通过原子规则 RuleSetread[1]所花费时间的 1 倍多。假设访问控制请求 *R* 为部门 C 的员工请求访问 D://file1 文件，采用前面类似的方法进行分析可以看出，通过访问控制规则 rule 进行判断所花费的时间至少是通过原子规则 RuleSetread[1]的 2 倍多。

5.3　评估实验

为测试本文提出的轻量级可重构的访问控制策略的评估效率，本文采用 Python 语言开发了一个中间件（middleware）实现访问控制策略决策功能，中间件运行在 Intel1.4 GHz i5CPU 16GiBRAM 计算机。利用上述中间件对不同的访问控制请求 *R* 分别通过访问控制规则 rule 和原子规则进行了评估，为准确记录评估时间，每个访问控制请求评估 10 000 次，测试的结果如表 1 所示。从实验结果来看，通过原子规则进行评估的效率是通过规则 rule 效率的 2 倍以上。对于逻辑关系复杂的访问控制策略，采用本文提出的方法生成的访问控制策略效果会更明显，比如部门 C 员工的写访问请求，通过原子规则进行评估，效率提高了 3 倍左右。

表 1　　**访问控制规则评估效率**

访问控制请求 R	通过规则 rule 进行评估所花费时间/ms	通过原子规则进行评估所花费时间/ms
部门 A 员工读文件 D://file1	7.92	3.41
部门 C 员工读文件 D://file1	13.72	6.46
部门 C 员工写文件 D://file1	20.92	5.18

5.4　访问控制策略冗余和冲突检测效率

访问控制策略冗余与冲突检测一直是访问控制策略的研究热点。现有文献分别从有向无环图模型、概念格、策略决策树等不同的方面提出了访问控制策略冗余与冲突检测的方法，本文将复杂访问控制策略等效转化为原子规则和代数表达式，通过判断访问控制策略等效的原子规则和代数表达式是否存在冗余或冲突，来判断访问控制策略是否存在冗余和冲突。如果原子规则存在冗余和冲突，那么对应的访问控制策略肯定存在冗余和冲突；如果等效的原子规则不存在冗余和冲突，需要进一步判断代数表达式是否存在冗余和冲突。如果代数表达式不存在冗余和冲突，那么对应的访问控制策略不存在冗余和冲突；如果代数表达式存在冗余和冲突，那么对应的访问控制策略存在冗余和冲突。由于最终复合生成的访问控制策略和原子规则在数量上为指数级关系，复杂度也不是同一个数量级。复杂访问控制策略冗余与冲突检测转化为对等效原子规则和代数表达式的检测，大大降低了难度和复杂度，提高了检测的效率。

6　结束语

以基于属性的访问控制策略为基础，根据访问控制规则中的操作类型、主体属性、客体属性和环境属性将基于属性的访问控制策略划分多个不相交的原子规则，并通过与、或等逻辑关系构成的代数表达式，将原子规则重构出复杂访问控制策略；根据原子规则中包含的属性是否相同，属性值是否相同或者是否存在交集，访问控制效果相同还是相反，判断原子规则是否存在冗余与冲突。实际系统中的访问控制策略大多是复杂访问控制策略，甚至有些策略存在冗余和冲突。本文提出将复杂访问控制策略转化为等效的原子规则和代数表达式的方式，降低访问控制策略的数量和复杂度；通过将原子规则评估结果代入代数表达式中，得到复杂访问控制策略评估结果；通过对转化生成的原子规则和代数表达式进行冗余与冲突检测，实现对复杂访问控制策略进行冗余与冲突检测。从空间复杂度和时间复杂度 2 个不同角度对本文提出的可重构的访问控制策略进行定性评估，实验结果表明，本文提出的访问控制策略重构方法大大降低了访问控制策略的长度、数量和复杂度，提高了访问控制策略冗余与冲突检测的效率以及访问控制策略评估的效率。

下一步工作将把原子规则等效转化的方法应用到实际信息系统中分析原子规则复合方法的效率，同时研究不同操作之间原子规则的冗余与冲突检测，尤其是有相互影响关系的操作之间原子规则之间的冗余和冲突检测。

参考文献：

[1] RIBEIRO C, ZUQUETE A, FERREIRA P, et al. SPL: an access control language for security policies and complex constraints[C]//The Network and Distributed System Security Symposium(NDSS'01). 2001: 89-107.

[2] DAMIANOU N, DULAY N, LUPU E, et al. The ponder policy specification language[C]//The International Workshop on Policies for Distributed Systems and Networks. 2001: 18-38.

[3] OASIS XACML. eXtensible access control Markup language XACML version 3.0[S]. OASIS Standard, 2013.

[4] RAO P, LIN D, BERTINO E, et al. An algebra for fine-grained integration of XACML policies[C]//The 14th ACM Symposium on Access Control Models and Technologies (SACMAT'09). 2009: 63-72.

[5] SHAHZAD M. Towards composing access control policies[C]//IEEE International Conference on Communications (ICC). 2018: 1-6.

[6] XU Z, STOLLER S. Mining attribute-based access control policies[J]. IEEE Transactions on Dependable and Secure Computing, 2015, 12(5): 533-545.

[7] NGO C, DEMCHENKO Y, LAAT DE C. Decision diagrams for XACML policy evaluation and management[J]. Computers & Security, 2015, 49: 1-16.

[8] 姚键, 茅兵, 谢立. 一种基于有向图模型的安全策略冲突检测方法[J]. 计算机研究与发展, 2005, 42(7): 1108-1114.
YAO J, MAO B, XIE L. A DAG-based security policy conflicts detection method[J]. Journal of Computer Research and Development, 2005, 42(7): 1108-1114.

[9] 李瑞轩, 鲁剑锋, 李添翼, 等. 一种访问控制策略非一致性冲突消解方法[J]. 计算机学报, 2013, 36(6): 1210-1223.
LI R X, LU J F, LI T Y, et al. An approach for resolving inconsistency conflicts in access control policies[J]. Chinese Journal of Computers, 2013, 36(6): 1210-1223.

[10] BECKERLE M, MARTUCCI L A. Formal definitions for usable access control rule sets from goals to metrics[C]//The Ninth Symposium on Usable Privacy and Security (SOUPS). 2013: 1-11.

[11] IYER P, MASOUMZADEH A. Mining positive and negative attribute-based access control policy rules[C]//The 23nd ACM Symposium on Access Control Models and Technologies (SACMAT'18). 2018: 161-172.

[12] CHAKRABORTY S, SANDHU R, KRISHNAN R. On the feasibility of attribute-based access control policy mining[C]//The 20th IEEE Conference on Information Reuse and Integration (IRI). 2019: 1-8.

[13] BONATTI P, VIMERCATI S D C, SAMARATI P. An algebra for composing access control policies[J]. ACM Transactions on Information and System Security (TISSEC), 2002, 5(1): 1-35.

[14] LUPU E C, SLOMAN M. Conflicts in policy-based distributed systems management[J]. IEEE Transactions on Software Engineering, 1999, 25(6): 852-869 .

[15] ST-MARTIN M, FELTY A P. A verified algorithm for detecting conflicts in XACML access control rules [C]//The 5th ACM SIGPLAN Conference on Certified Programs and Proofs. 2016: 166-175.

[作者简介]

谢绒娜（1976− ），女，山西永济人，西安电子科技大学博士生，主要研究方向为网络与系统安全、访问控制、密码工程。

李晖（1968− ），男，河南灵宝人，博士，西安电子科技大学教授、博士生导师，主要研究方向为密码信息安全、信息论与编码理论。

史国振（1974− ），男，河南济源人，博士，北京电子科技学院副教授、硕士生导师，主要研究方向为网络与系统安全、嵌入式安全。

郭云川（1977− ），男，四川营山人，博士，中国科学院副研究员、博士生导师，主要研究方向为访问控制、形式化方法。