高技术通讯 2012 年 第22 卷 第5 期:460 ~ 467

doi:10.3772/j.issn.1002-0470.2012.05.003

AACF:基于逻辑的非单调授权与访问控制框架①

包义保②*** 殷丽华* 方滨兴* 郭 莉*

(*中国科学院计算技术研究所信息智能与信息安全研究中心 北京 100190)

(**解放军信息工程大学电子技术学院信息安全技术系 郑州 450004)

摘 要 针对不完全语义条件下的非单调授权和权限判决问题进行形式化研究,提出了一种新的基于逻辑的非单调授权与访问控制框架——AACF。该框架通过扩展型分层逻辑程序表达授权与访问控制策略,支持不完全语义条件下的非单调授权、权限传播及冲突检测与消解等高级特性,此外,AACF 的语义查询/权限判决算法的计算复杂度证明是多项式级的。因而,AACF 比现有的访问控制框架具有更好的表达能力和计算特性,且具有更好的实用性。

关键词 授权与访问控制,框架,安全策略,语义,逻辑程序

0 引 言

访问控制是实现信息安全的关键技术之一。访问控制技术是指通过访问控制机制介入资源访问过程,根据访问控制策略对每个访问请求进行权限判决并裁定是否允许访问请求者访问[1]。自访问控制概念出现以来[2],研究人员提出了许多访问控制模型[2-8],并对访问控制模型的表达能力、权限判决及安全属性验证的可判定性和计算复杂度等问题进行了探讨。传统的访问控制机制[2-5]一般依赖于特定的策略表达方式和实现机制,灵活性和适应性受到严重制约。近年来,人们尝试设计一种至少具备以下功能的通用授权与访问控制框架[6-8]:(1)用统一的规范语言表达不同的安全策略,并进行统一管理;(2)能够根据系统运行环境的变化快速调整访问控制策略,同时保持访问控制机制不变。以上两点不仅要求访问控制框架具有灵活的表达能力,而且要求其访问控制机制算法统一、简单高效,降低出现漏洞的概率,从而使其成为可信计算基(trusted computing base, TCB)的一部分。这一思想在实际的系统中也得到了应用[9],文献[7, 8]在这方面进行了良好的尝试。

本质上说,访问控制策略表达是一种知识表达,使用逻辑来表达知识是一种极为重要的方法[10]。由于逻辑具有一系列良好的特性和能力,如描述问题的精确性及非单调推理能力等,使得它适合于表达授权与访问控制策略中的规则及权限传播等特性。使用逻辑作为工具研究访问控制问题起源于上世纪70-80 年代[11]。研究人员认为,一阶逻辑基本上能够满足形式化描述科学问题的需要。但是,直接使用一阶逻辑表达的访问控制策略,其权限判决的计算复杂度可能极高或不可判定[12]。为此,需要从语法上对一阶逻辑进行限制,选取一阶逻辑的一个片段,强制其具备可计算特性。目前,实现该思想的方法主要分为两类,一类直接对一阶逻辑语法进行限制[12],另一类利用现有的逻辑系统,如逻辑程序系统[13]。逻辑程序语言是非过程语言,具备表述性语义,逻辑程序子句是自描述的,子句之间没有次序限制,这符合策略对规则的语法要求;逻辑程序具备统一的程序性语义和语义查询算法,这符合权限判决的统一性要求。由此可见,用逻辑程序表达授权与访问控制框架具有较大的优势。通过逻辑程序表达授权与访问控制策略起源于文献[6],它通过逻辑程序统一编码各种类型的访问控制策略,通过逻辑程序语义查询算法实现统一的权限判决。虽然文献[6]提出的框架支持不完全语义条件下非单调授权与访问控制逻辑的表达,但其权限判决算法却是不可判定的。在文献[6]的启发下,文献[7, 8, 14]

① 863 计划(2009AA01Z438,2009AA01Z431),973 计划(2007cb311100)和国家自然科学基金(60703021)资助项目。

② 男,1976 年生,博士生,讲师;研究方向:网络安全,安全性分析,安全策略;联系人,E-mail: baoyibao99@yahoo.com.cn

(收稿日期:2010-11-29)

分别提出了利用不同类型的逻辑程序构建授权与访问控制统一框架的方法。虽然这些方法所构建的框架都具有确定性的权限判决算法,但并不具备表达不完全语义条件下非单调授权与访问控制逻辑的能力。本文立足于扩展型分层逻辑程序理论,提出了一种基于逻辑的非单调授权与访问控制逻辑框架(a logic-based framework of nonmonotonic authorization and access control,缩写为 AACF)。AACF 与上述框架相比有下列优势:(1)具备强大的策略表达能力,能够简捷灵活地表达出各种具有复杂控制逻辑的访问控制策略,所表达策略的语义丰富、精练;(2)不仅能够表达不完全语义条件下的非单调授权与权限传播,同时还保证其权限判决的计算复杂度是多项式级的;(3)支持灵活的授权异常处理,从而可以细粒度地表达出复杂的授权空间。

1 访问控制系统的建模与表示

本节首先形式化地给出访问控制的一些基本概念,以便于建立访问控制系统的抽象模型,然后讨论通过逻辑程序表达这些概念的方法,建立访问控制系统的逻辑程序表示,进而形成 AACF 安全策略语言。该语言不仅可以表示通常意义上的访问控制规则,还可以表示闭世界假设规则、允许异常规则及约束规则。最后通过一个基于角色的访问控制(RBAC)[5]策略实例说明利用 AACF 安全策略语言编写访问控制策略的基本方法。

1.1 访问控制基本概念及建模

授权与访问控制涉及的因素繁多,主要由主体、客体和访问行为三个基本要素组成[2]。在本文中,所有主体构成的集合记为 S,所有客体构成的集合记为 O,且 $S \subseteq O$。所有访问行为构成的集合记为 A,且 $A = \cup_{o \in O} act(o)$,这里 $act(o)$ 表示针对客体 o 的所有访问行为的集合。客体被分为若干种类型,每种类型客体对应一个访问行为集,不同类型客体的访问行为集的交集为空集。访问控制系统中所有实体构成的集合记为 E,且 $E = O \cup A$,所有实体之间的关系通过谓词表示。

信息系统为实现访问控制所采用的模型、策略和机制的总和构成一个访问控制系统。信息系统运行过程中主体对客体的一次访问行为称为访问请求,但该请求是否被允许则取决于访问控制系统的判决结果。每个访问请求可表示为一个三元谓词 $req(s, o, a) \in S \times O \times A$。每个 2 元组 $\langle o, a \rangle \in O \times A$ 称为一个访问权限。任一个 4 元组 $\langle s, \lambda, o, a \rangle \in S \times \{+, \neg, \perp\} \times O \times A$ 称为允许(+)/禁止(¬)/未指派(⊥)主体 s 拥有访问权限 $\langle o, a \rangle$,用带有下标的 3 元谓词 $permit_\lambda(s, o, a)$ 表示,其中 $permit_+(s, o, a)$ 称为正向权限,$permit_\neg(s, o, a)$ 称为反向权限。所谓授权,是指在主体、客体和访问行为之间建立一种关系,而这种关系代表主体对客体的一种访问能力。通过授权在主体和客体之间建立的访问能力,称为主体和客体之间的授权关系。在本文中,每个授权关系可形式化地表示为带有下标的 5 元谓词 $prvl_\lambda(s, o, a, \sigma, g)$,其中:$\lambda \in \{+, \neg\}$ 表示允许(+)或禁止(¬)主体 s 拥有访问权限 $\langle o, a \rangle$,s 表示类型为主体的变量,o 表示类型为客体的变量,a 表示类型为访问行为的变量,σ 表示 s 能够拥有该权限的前提条件,$g \in G$ 表示授权者,即某个权限管理员,G 表示所有权限管理员的集合。所有授权关系构成的集合称为授权关系集。显然,每个授权关系在 s、o、a 上的投影 $\pi_{s,o,a}(prvl_\lambda(s,o,a,\sigma,g))$ 表示主体 s 拥有的访问权限 $permit_\lambda(s,o,a)$。

访问控制系统(ACS)可形式化地表示成 $ACS = \langle S, O, A, G, M, P, P' \rangle$,其中 S、O、A、G 分别表示访问控制系统中的主体、客体、访问行为和权限管理员的集合;M 表示访问控制策略模型,如 BLP(Bell-LaPadula)模型、RBAC 模型等;P 表示授权关系集;P' 表示为维护系统安全运行所需策略的集合,如权限传播策略、冲突消解策略、完整性约束策略、默认策略等[7]。

从授权与访问控制的角度来看,ACS 是所有主体拥有的权限的集合,该集合的内涵 ACS_I 由 ACS 的内部结构决定,而外延 ACS_E 为由 ACS_I 推导出的所有形如 $permit_+(s,o,a)$ 和 $permit_\neg(s,o,a)$ 的常量谓词构成的集合。如果 ACS_E 同时包含 $permit_+(s,o,a)$ 和 $permit_\neg(s,o,a)$,则 ACS 是一个有权限冲突的访问控制系统。权限冲突有时是因系统错误或权限管理员的疏忽造成的,有时也是迫不得已而引入的。因此,访问控制系统应有明确定义的权限冲突检测算法与消解策略。

对于访问请求 $req(s,o,a)$,若 $permit_+(s,o,a) \in ACS_E$ 且 $permit_\neg(s,o,a) \in ACS_E$,则权限判决应返回“冲突错误”;若仅有 $permit_+(s,o,a) \in ACS_E$,则应返回“允许”;若仅有 $permit_\neg(s,o,a) \in ACS_E$,则应返回“禁止”;如果 $permit_+(s,o,a) \notin ACS_E$ 且 $permit_\neg(s,o,a) \notin ACS_E$,则返回“未指派”。

访问控制系统中的权限判决过程可以看成查询 *ACS* 外延的过程。由于 *ACS* 的外延可能非常巨大，因此，仅通过管理员手工制定的授权关系的投影获得 *ACS* 的外延效率低下。为此，*ACS* 应明确地定义权限传播策略。所谓权限传播，是指访问控制系统通过实体之间的关系推导出的没有显式授权关系与之对应的权限。权限传播一般通过具有推理能力的逻辑演算系统实现，它是建立精简的授权关系集的主要方法。

1.2 用逻辑程序表达访问控制

本节讨论如何通过逻辑程序表达上述抽象的访问控制系统。逻辑程序系统是一个具有推理能力的逻辑演算系统，它是一阶逻辑的片段，但又有所扩展。虽然逻辑程序的语法受到一定程度的约束，但表达能力却非常强大[15]。本文所使用的术语，如常量、变量、函数、谓词、逻辑运算符、原子(atom)、文字(literal)、逻辑表达式等，都和一阶逻辑中的相关概念相对应。鉴于 AACF 对语义表达能力的要求，同时考虑到实际应用中需要“*not*”和“¬”两种不同类型的“非”逻辑运算符[16]，AACF 将 Gelfond 等提出的扩展型逻辑程序及 Gelder 等提出的三值逻辑[17]结合起来作为描述访问控制策略的工具，从而支持不完全语义条件下的非单调推理。这里“¬”为经典逻辑中的“非”，它显示地表达否定知识；“*not*”指失败“非”(negation as failure, NAF)，通常也称之为默认“非”，它间接地表示否定知识。有关这两者之间的区别，请参考文献[18]和[16]。

定义 1(扩展型逻辑程序中的规则)[18]：具有

$$l_0 \leftarrow l_1, \cdots, l_m, not\ l_{m+1}, \cdots, not\ l_n \quad (1)$$

所示语法形式的逻辑表达式称为规则。这里 $n \geqslant m(0, l_i$ 为正文字 A_i 或负文字 $\neg A_i, A_i$ 为原子，l_0 称为规则的头部，$\{l_1, \cdots, l_m, not\ l_{m+1}, \cdots, not\ l_n\}$ 称为规则的体部。如果一个原子不含变量，则称为常量原子，相应的文字称为常量文字。规则若其中的文字都是常量文字，则称为常量规则。设 $Pos(r)$ 表示 $\{l_1, \cdots, l_m\}$，$Neg(r)$ 表示 $\{l_{m+1}, \cdots, l_n\}$，则(1)式也可写成 $l_0 \leftarrow Pos(r) \cup not\ Neg(r)$ 的形式。如果规则的体部为空集，则可简写成 l_0。

定义 2(扩展型逻辑程序)[18]：由上述形式的规则构成的集合称为扩展型逻辑程序。

扩展型逻辑程序 Π 中所有头部谓词符号为 p 的规则构成的集合，称为 p 的定义，记为 $Def_{\Pi}(p) = \{r \mid r \in \Pi, Head(r) = p\}$。

根据 *ACS* 的定义，可以将 *ACS* 的符号表示转化成扩展型逻辑程序中的规则，如表 1 所示。显然，这种转换是一一映射的。根据表 1，AACF 可将每个访问控制策略编码成一个扩展型逻辑程序，该逻辑程序中的每个规则对应着 *ACS* 中的一个特征不变式或访问控制规则，从而将 *ACS* 转换成一个基于扩展型逻辑程序的逻辑演算系统。逻辑程序中的规则构成逻辑系统中的公理，其逻辑演算算子取决于逻辑程序语义查询算法，如 SLDNF[13]、XSB[19] 及 Smodel[20] 算法等。

表 1 *ACS* 中的符号在 AACF 中的表示方法

在 *ACS* 中的符号	在 AACF 中的规则
$req(s, o, a)$	$\leftarrow permit(s,o,a)$，查询结果为‘+’,‘¬’,‘⊥’之一
$prvl_{+}(s,o,a,\sigma,g)$	$prvl(s,o,a,g) \leftarrow \sigma$
$prvl_{\neg}(s,o,a,\sigma,g)$	$\neg\ prvl(s,o,a,g) \leftarrow \sigma$
$permit_{+}(s,o,a)$	$permit(s,o,a)$
$permit_{\neg}(s,o,a)$	$\neg\ permit(s,o,a)$
$permit_{\perp,+}(s,o,a)$	$not\ permit(s,o,a)$
$permit_{\perp,\neg}(s,o,a)$	$not \neg\ permit(s,o,a)$
$\pi_{s,o,a}(prvl_{+}(s,o,a,\sigma,g))$	$permit(s,o,a) \leftarrow prvl(s,o,a,\sigma,g)$
$\pi_{s,o,a}(prvl_{\neg}(s,o,a,\sigma,g))$	$\neg\ permit(s,o,a) \leftarrow \neg\ prvl(s,o,a,\sigma,g)$

利用扩展型逻辑程序的非单调特性和不完全语义描述特性，可以表达一些具有特殊意义的规则。授权与访问控制逻辑中通常隐含着闭世界假设[18]，这是一种对未来或未知知识进行逻辑推理的常用假设方式。闭世界假设通常用于表达默认策略，是一种特殊形式的规则，可以通过扩展型逻辑程序的规则表示出来，如定义 3 所示。

定义 3(闭世界假设规则)：形如 $l \leftarrow not\ \bar{l}$ 的规则称为闭世界假设规则(close world assumption, CWA)。这里文字 $\bar{l}$ 表示文字 l 的补。

访问控制规则概括地表达了多个场景下系统访问行为的控制方式，但这些场景中的部分场景可能并不适用于该规则，称为该规则的异常场景。在编制授权与访问控制策略过程中，需要有一种手段将异常场景排除在规则适应的范围之外。AACF 通过允许异常规则达到上述目的，如定义 4 所示。允许异常规则是一种特殊形式的规则，在表达默认规则、改写规则的部分行为以及消解策略中的冲突等异常情况时非常有用。

定义 4(允许异常规则)：形如 $id: l_0 \leftarrow l_1, \cdots, l_m, not\ l_{m+1}, \cdots, not\ l_n, not\ ab(id, \cdots)$ 的规则称为允许异常规则，这里 id 为规则的唯一标识。

在 *ACS* 中，约束一般是为了验证与监控 *ACS* 的

安全属性而设置的,它并不影响访问控制策略的语义。约束也可通过规则表达出来,如定义5所示。例如,通过约束规则 $\square \leftarrow permit(X_s,X_o,X_a)$, $\neg\ permit(X_s,X_o,X_a)$ 可实现 ACS 中的冲突检测。

定义5(约束规则):形如 $\square \leftarrow Pos(r)$, $not\ Neg(r)$ 的规则为约束规则。对于由任意文字构成的集合 C,若 $C \supseteq Pos(r)$, $C \cap Neg(r) = \varnothing$,则 C 违反了约束规则;否则 C 满足约束规则。。

由上述规则编写的逻辑程序称为AACF安全策略,其相应的语言称为AACF安全策略语言。

1.3 AACF安全策略语言使用示例

本节通过表达标准RBAC策略的过程说明AACF安全策略语言的使用方法。基本RBAC的最小功能主要包括[5]将权限指派给角色和将角色指派给用户。前者称为角色授权,后者称为角色指派。除基本RBAC外,RBAC模型分层、角色分层和职责分离约束等高级特性在RBAC中也非常重要。文献[5]描述了一种标准的RBAC模型分层方法,其关键特性如表2所示。

表2 RBAC模型分层的关键特性[5]

层	RBAC分层	功能描述
1	$RBAC_F$	用户-角色之间的多对多授权关系,角色-权限之间的多对多授权关系。
2	$RBAC_{H2A}$	$RBAC_F$ +无限制的角色分层
	$RBAC_{H2b}$	$RBAC_F$ +受限制的角色分层
3	$RBAC_{C3A}$	$RBAC_{H2A}$ +职责分离(Separation of Duties, SoD)
	$RBAC_{C3B}$	$RBAC_{H2B}$ +职责分离(Separation of Duties, SoD)

AACF安全策略语言表达 $RBAC_F$ 如表3所示,表达结果为扩展型逻辑程序,记为 Π_{RBAC_F},这里 $\{assigned, activate\} \subseteq act(r)$ 表示针对角色的两个访问行为,即"角色指派"和"角色激活"。

表3 $RBAC_F$ 到AACF的映射

$RBAC_F$ 中的元素[5]	AACF中的规则
X_g 将 X_r 指派给 X_u	$prvl(X_u, X_r, assigned, X_g)$
X_g 将 $\langle X_o, X_a\rangle$ 授予 X_r	$prvl(X_r, X_o, X_a, X_g)$
权限传播: 这里 $dermit(X_s,X_o,X_a)$ 表示 X_s 通过权限传播 获权限 (X_o, X_a)	$dermit(X_s,X_o,X_a) \leftarrow prvl(X_s,X_o,X_a,X_g)$ $permit(X_u,X_o,X_a) \leftarrow$ $dermit(X_u,X_r,assigned)$, $dermit(X_r,X_o,X_a)$

向 Π_{RBAC_F} 中加入规则

$$dermit(X_u, X_o, X_a) \leftarrow prvl(X_u, X_{r_1}, assigned, X_{g_1}), prvl(X_{r_2}, X_o, X_a, X_{g_2}), X_{r_1} \prec_H X_{r_2} \quad (2)$$

即可表达 $RBAC_{H2A}$,表达结果记为 $\Pi_{RBAC_{H2A}}$,这里 $\prec_H$ 表示角色分层关系。该规则表示高层角色可以继承低层角色的正向权限,即低层角色的正向权限自动传播到高层角色。若高层角色从两个不同低层角色分别继承了相互矛盾的正向权限和反向权限,则高层角色事实上仅继承了正向权限,即采用"正向权限优先"策略冲突消解方法。若要采用"反向权限优先"策略冲突消解方法,则只需要将上述规则替换为规则

$$dermit(X_u, X_o, X_a) \leftarrow prvl(X_u, X_{r_1}, assigned, X_{g_1}), X_{r_1} \prec_H X_{r_2}, prvl(X_{r_2}, X_o, X_a, X_{g_2}), X_{r_1} \prec_H X_{r_3}, not\ \neg\ prvl(X_{r_3}, X_o, X_a, X_{g_3}) \quad (3)$$

即可。

由此可见,AACF利用扩展型逻辑程序系统的推理能力,可以实现RBAC角色分层和权限继承等特性。众所周知,RBAC仅允许通过角色之间的层次关系单调地传播正向权限,从而达到简化授权、防止策略冲突的目的。而上述示例说明,AACF不仅可以模拟RBAC权限传播的行为,还可对其进行有意义的扩展。

在保证可计算性的条件下,向 $\Pi_{RBAC_{H2A}}$ 中加入更多的规则,可以实现更符合实际要求的RBAC策略。例如,向 $\Pi_{RBAC_{H2A}}$ 中加入

$$l_1:\ prvl(X_u, r, assigned, X_g) \leftarrow X_u \in Group(GN), not\ ab(l_1, X_u) \quad (4)$$

$$l_2: ab(l_1, X_u) \leftarrow X_u \in Group(GL) \quad (5)$$

这两个规则。可实现组内用户角色指派的差异化。其中,第一个规则是允许异常规则,表示"通常情况下,如果用户属于组 GN,则该用户被自动指派角色 r,但既属于组 GN 又属于组 GL 的用户除外"。例如,为了保证软件代码的健壮性,金融软件开发小组成员(即组 GN 中的成员)可以修改代码,但该小组中新来成员(即组 GL 中的成员)除外。这一特性在现有的安全策略语言中无法轻易地表达出来。

向 $\Pi_{RBAC_{H2A}}$ 加入相应的SoD约束,则可形成表示 $RBAC_{C3A}$ 模型的逻辑程序 $\Pi_{RBAC_{C3A}}$。SoD分为静态SoD和动态SoD,静态SoD要求不能将相互排斥的两个角色同时指派给同一用户,动态SoD要求同一用户不能同时激活相互排斥的两个角色。用 $SSD \subseteq R \times R$ 表示静态角色互斥关系,用 $DSD \subseteq R \times R$ 表示动态角色互斥关系。设它们对应的谓词分别为 $ssd(X_{r_1}, X_{r_2})$ 和 $dsd(X_{r_1}, X_{r_2})$,则在不允许继承情况下静态SoD约束规则可表示为

$$□ \leftarrow prvl(X_u, X_{r_1}, assigned, X_{g_1}), prvl(X_u, X_{r_2}, assigned, X_{g_2}), ssd(X_{r_1}, X_{r_2}) \quad (6)$$

考虑到继承情况,上述规则应更改为

$$□ \leftarrow permit(X_u, X_{r_1}, assigned), permit(X_u, X_{r_2}, assigned), ssd(X_{r_1}, X_{r_2}) \quad (7)$$

这是因为 *permit* 已经考虑了权限传播问题(参见表 3)。若把"判断用户能否激活某个角色"看成是一种动态权限,则动态 SoD 可由下列两个规则表示:

$$permit(X_u, X_{r_2}, activate) \leftarrow dsd(X_{r_1}, X_{r_2}), \neg active(X_u, X_{r_1}) \quad (8)$$

$$permit(X_u, X_{r_1}, activate) \leftarrow dsd(X_{r_1}, X_{r_2}), \neg active(X_u, X_{r_2}) \quad (9)$$

至此,我们通过 AACF 安全策略语言表示出了 $RBAC_F$, $RBAC_{H2A}$, $RBAC_{C3A}$。类似地,可以得到 $RBAC_{H2b}$, $RBAC_{C3B}$ 在 AACF 中的表示。同上所述,AACF 可容易地表达出诸如 HUR[2]、BLP[3]、Biba[4]、FAF[7]等具有各种特性的各种类型的安全策略。限于篇幅,这里不再赘述。

在传统的访问控制中,信息系统一般和特定类型的安全策略紧耦合,其访问控制机制也硬编码到信息系统中,难以更换和升级。然而,在 AACF 中,各种类型的安全策略都可通过扩展型逻辑程序表达出来,因而具有共同的访问控制机制实现方法,这给信息系统中安全策略的管理带来了极大的方便。例如,某信息系统初始时决定使用 HRU 类型的访问控制策略,运行一段时间后决定更换到 RBAC 类型的访问控制策略,再后来又要求更换到 BLP 和 Biba 类型的访问控制策略。如果使用传统的访问控制实现方法,则需要修改访问控制机制实现方法,代价巨大。显然,若使用 AACF 作为其访问控制的实现框架,则信息系统在随后的运行过程中,可容易地更换其安全策略类型,而不需要修改其访问控制机制。

2 AACF 语法规范和语义

直观地讲,逻辑程序的语义是指在给定的解释域上逻辑程序能够推导出的结果集。由于非单调逻辑运算符"*not*"的引入,使得扩展型逻辑程序可能存在多种类型的语义模型[17],到底哪一种语义模型是其正则语义,目前未有定论。AACF 采用扩展型逻辑程序作为其理论工具,为了保证 AACF 描述的每个策略都具有唯一的语义模型,需要给出相应的语法规范。对于一些特殊结构的逻辑程序,如分层逻辑程序(stratified logic program, SLP)和局部分层逻辑程序(locally stratified logic program, LSLP)[21],其各种不同类型的语义模型本质上是等价的。因此,从安全的角度来看,SLP 和 LSLP 比较符合 AACF 的意图。

2.1 AACF 语法及其约束

通过前面的讨论可知,AACF 将访问控制策略编码成扩展型逻辑程序,将权限判决编码成针对扩展型逻辑程序的查询。在本节中,AACF 通过引入恰当的语法约束保证其编写的每个访问控制策略都是 SLP,从而保证它们对所有的查询都能够在多项式时间内处理完毕。

AACF 中所有的谓词符号分为两类:一类是预置的、具有固定涵义的谓词符号,它们构成 AACF 安全策略语言的核心,如表 4 所示,这里 $Lit(B)$ 表示集合 B 中的谓词符号能够构建的所有文字的集合;另一类是由访问控制策略管理员根据实际需要定义的谓词符号,其涵义由管理员指定。

设 r 为逻辑程序 Π 中的规则,若 $l_1 \in Head(r)$, $l_2 \in Body(r)$,则称 l_1 在 Π 中依赖 l_2,记为 $l_1 \leq_\Pi l_2$。显然,关系 $\leq_\Pi$ 具有传递性,即若 $l_1 \leq_\Pi l_2$ 且 $l_2 \leq_\Pi l_3$,则 $l_1 \leq_\Pi l_3$。$\leq_\Pi(l) = \{L \in Lit(\Pi) \mid l \leq_\Pi L\}$ 表示文字 l 依赖的所有文字的集合。AACF 的语法约束有以下三种:

(1) $Def_\Pi(et \in Lit(EtRel))$ 中规则的语法约束。定义 $Lit(EtRel)$ 中的文字只能使用 $Lit(EtRel)$ 中的元素,即 $Body(Def_\Pi(et \in Lit(EtRel))) \subseteq Lit(EtRel)$,且 $\leq_\Pi$ 是 $\Pi_0 = \cup_{et \in Lit(EtRel)} Def_\Pi(et)$ 上关于 $Neg(\Pi_0)$ 的部分分层关系(参见定义 6)。

(2) $Def_\Pi(et \in Lit(AoRel))$ 中规则的语法约束。定义 $Lit(AoRel)$ 中的文字只能使用 $Lit(EtRel) \cup Lit(AoRel)$ 中的元素,即 $Body(Def_\Pi(et \in Lit(AoRel))) \subseteq Lit(EtRel) \cup Lit(AoRel)$, $\leq_\Pi$ 是 $\Pi_1 = \cup_{et \in Lit(AoRel)} Def_\Pi(et)$ 上关于 $Neg(\Pi_1)$ 的部分分层关系。

(3) 约束和监控规则的语法约束。由于 Π 中的每个谓词 $ab(r, \cdots)$ 的第一个参数都为常量,因此可以将之看成新的谓词 $ab_r(\cdots)$。若 $r \in \Pi_0$,则 r 必须符合(1);若 $r \in \Pi_1$,则 r 必须符合(2)。文字 □ 的定义可以使用除 $ab(\cdots)$ 和 □ 之外的所有其它文字,即 $Body(Def_\Pi(□)) \in Lit(\Pi) - \{□, ab(\cdots)\}$。

表 4　AACF 中的谓词

分类	谓词(前缀或中缀表示法)	语义描述
实体定义及实体关系谓词	$Lit(EtRel)$:$EtRel$ 表示所有实体之间关系的谓词名称的集合;每个元素 $et \in Lit(EtRel)$ 可以是 0,1,…, n 元谓词[1],例如,实体属性谓词、分层关系谓词、偏序关系谓词等,如下所示。	
	实体属性谓词 $att(e,\alpha_1,\cdots,\alpha_n)$	实体 e 具有属性 $\alpha_1,\cdots,\alpha_n(n \geqslant 0)$。
	分层关系谓词 $b \preceq_H c$	实体 b 和 c 之间具有分层关系。
	偏序关系谓词 $b \preceq_P c$	实体 b 和 c 之间具有偏序关系。
	格关系谓词 $b \prec_L c$	实体 b 和 c 之间具有格关系
	等价关系谓词 $b = c$	实体 b 和 c 完全相同。
	属于关系谓词 $b \in B(b \notin B)$	实体 b 是(或者不是)实体 B 中的一个元素,例如 $u \in X_{Group}$ 表示主体 u 是组 X_{Group} 中的一个成员。
授权相关谓词	$Lit(AoRel)$:$AoRel$ 表示所有和授权相关的谓词名称的集合;每个元素 $ao \in Lit(AoRel)$ 可以是 0,1,…, n 元谓词,例如,$prvl(X_s, X_o, X_a, X_g)$、$dermit(X_s, X_o, X_a)$、$permit(X_s, X_o, X_a)$。	
	$prvl(X_s, X_o, X_a, X_g)$	权限管理员 X_g 将访问权限 $\langle X_o, X_a\rangle$ 授予主体 X_s。
	$dermit(X_s, X_o, X_a)$	由 ΛCS_i 可推导出“允许 X_s 拥有权限 $\langle X_o, X_a\rangle$”。
	$permit(X_s, X_o, X_a)$	允许主体 X_s 拥有权限 $\langle X_o, X_a\rangle$。
约束谓词	$ab(id,\cdots)$	id 标识的规则中的异常信息,用于构建允许异常规则。
	□	所有的文字集合都包含这个 0 元谓词,用于构建约束规则。
注 1:0 元谓词也称为常量谓词。		

定义 6(分层和部分分层关系):设 R 为有限集 S 上的一个二元关系,若存在 S 的一个划分 $\{S_i \mid i = 0,1,\cdots,n\}$ 使得:对于任一 xRy,若 $x \in S_i$,则有 $y \in \cup_{j\leqslant i}S_j$,则称 R 为 S 上的分层关系,相应地,S_k 称为 S 的第 k 层。设 R 为 S 上的分层关系,B 为 S 的子集,对于任意的 $x \in S_i, 0 \leqslant i \leqslant n, y \in B$,若 xRy,有 $y \in \cup_{j<i}S_j$,则称 R 为 S 上关于 B 的部分分层关系。显然,分层关系都是部分分层关系。

定理 1:符合上述(1)、(2)、(3)语法约束且具备一致性的扩展型逻辑程序是分层逻辑程序。

证明:设 Π 为一个符合上述(1)、(2)、(3)语法约束的逻辑程序,首先利用文献[18]中的方法将 Π 转换成不含“¬”的逻辑程序 Π^+。接下来如果能够证明 Π^+ 为 SLP,则 Π 显然也是 SLP。

由于 Π^+ 中的 n 元谓词 $ab(r, X_1, \cdots, X_{n-1})$ 的第一个参数都为常数,因此可将之看成 $n-1$ 元谓词 $ab_r(X_1, \cdots, X_{n-1})$,这不会改变 Π^+ 的语义模型。由上述(1)、(2)、(3)约束可知,可设 $\Pi^+ = \Pi_0^+ \cup \Pi_1^+$。

令 $\lambda_i(l)$ 表示文字 l 在 Π_i^+ 的分层的序号。由约束(1)和(3)可知,当 $r \in \Pi_0^+$ 时,$\lambda_0(Head(r)) \geqslant \max(l \in \lambda_0(Pos(r))$;由约束(2)和(3)可知,当 $r \in \Pi_1^+$ 时,$\lambda_1(Head(r)) \geqslant \max(l \in \lambda_1(Pos(r))$。因而 Π_0^+ 和 Π_1^+ 都是分层逻辑程序。由约束(1)和(2)可知 Π_0^+ 没有使用 Π_1^+ 中的谓词,因而可构造 Π^+ 的分层映射函数

$$\lambda(l) = \begin{cases} \lambda_0(l) & l \in Lit(\Pi_0) \\ \max\limits_{L\in Pos(\Pi_0)}(\lambda_0(L)) + \lambda(l) + 1 & l \in Lit(\Pi_1) \\ \max\limits_{L\in Pos(\Pi_0)}(\lambda_0(L)) + \max\limits_{L\in Pos(\Pi_1)}\lambda(L) + 1 & l = \square \end{cases} \tag{10}$$

由文献[21]中相关定义可知,Π^+ 是分层逻辑程序。证毕。

显然,在 1.3 节讨论的访问控制策略 Π_{RBAC_P}、$\Pi_{RBAC_{H2A}}$ 和 $\Pi_{RBAC_{C3A}}$ 都是分层逻辑程序。

分层逻辑程序一定是局部分层逻辑程序,但判断一个逻辑程序是否为局部分层逻辑程序是 NP-完全问题[22]。因此,AACF 推荐使用 SLP 表达授权与访问控制策略,而不使用 LSLP,以免出现不可判定的访问控制策略。

2.2　AACF 语义模型及查询处理

默认非“*not*”的引入使得扩展型逻辑程序可能存在多种不同类型的语义模型。AACF 使用扩展型分层逻辑程序编码访问控制策略,因而 AACF 的访问控制策略中也可能存在“*not*”,这是表达不完全语义环境下非单调授权与访问控制策略所必需的。由于 AACF 具有明确的语法限制,从而使得 AACF 表达的访问控制策略不可能拥有多种不同的解释,不会造成语义上的困扰,因而用于表达访问控制策略是合适的。下面以稳定语义模型作为讨论对象。

不含“¬”的分层逻辑程序的稳定模型,最坏情

况下的语义查询算法的时间复杂度是多项式级的，使用扩展型逻辑程序的 AACF 同样具备这样的特性，定理 2 说明了这点。

定理 2：AACF 通过扩展型逻辑程序表达的访问控制策略都具有唯一的稳定语义模型，其语义查询是可判定的，且具有多项式级的时间复杂度。

证明：由定理 1 可知 AACF 中的逻辑程序 Π 都是 SLP。利用文献[18]中的方法将 Π 转换成不含“¬”的逻辑程序 Π^+，由文献[23, 24]可知，Π^+ 具有唯一的稳定模型 M_{Π^+}，由文献[20]可知，可在多项式时间内计算出 M_{Π^+}。利用前述转换方法的逆过程将 M_{Π^+} 转换成 M_Π。如果 M_Π 是一致的，则 M_Π 就是 Π 唯一的稳定模型，因而，语义查询可在多项式时间内处理完毕。如果 M_Π 不一致，AACF 可在多项式时间内完成其不一致性检测，并在冲突消解规则的作用下完成最终决策。证毕。

通常情况下，逻辑程序是在闭世界假设条件下进行语义计算的[18]。那么将闭世界假设加入 SLP 后，原 SLP 所具备的语义特性及计算特性是否会发生改变呢？我们有下述结论：

定理 3：设 Π 是 AACF 中的逻辑程序，文字 $l \notin \Pi$，则在 Π 中加入闭世界假设规则 $l \leftarrow not\ \bar{l}$ 后，其语义特性（具有唯一的稳定模型）及计算特性（查询处理可在多项时时间内处理完毕）保持不变。

证明：由定理 2 可知 Π 具有唯一的稳定模型 M_Π。由文献[18]中的命题 4 及文献[25]中的命题 2 可知，$\Pi \cup \{l \leftarrow not\ \bar{l}\}$ 具有唯一的稳定模型 $M_\Pi \cup \{l : \bar{l} \notin M_\Pi\}$。由于 Π 的查询处理是多项式级的，因而 $\Pi \cup \{l \leftarrow not\ \bar{l}\}$ 查询处理也是多项式级的。证毕。

访问控制系统应为每个形如 $\leftarrow permit(s,o,a)$ 或 $\leftarrow \neg\ permit(s,o,a)$ 的常量查询给出肯定或否定的判决结果，而 AACF 可能输出“未指派”的判决结果。为此，AACF 处理方法如下：首先，AACF 为 Π_{ACS} 中的每个授权相关的谓词符号都建立一个唯一的闭世界假设规则，所有这些规则构成闭世界假设规则库；其次，当 AACF 接收到查询请求时，它先查询 Π_{ACS}，如果返回肯定或否定的判决结果，则直接将之返回给查询者。如果返回“未指派”，则使用该查询再次查询闭世界假设规则库，从而最终得到肯定或否定的判决结果。

表 5 AACF 和现有基于逻辑程序的授权与访问控制框架的比较

特性	AACF（本文的工作）	Woo[6]	FAF[8]	DKAL[14]
理论基础	扩展型分层逻辑程序	扩展型逻辑程序	分层逻辑程序	Datalog
计算复杂度	多项式级	不可判定	多项式级	多项式
支持“¬”	支持	支持	不支持	不支持
支持“*not*”	支持	支持	部分支持	不支持
非单调授权	支持	支持	部分支持	不支持
异常处理	支持	未明确	不支持	不支持
权限传播与冲突处理	支持	未明确	部分支持	部分支持

3 结 论

本文在逻辑程序理论的基础上，提出了一种统一的授权与访问控制管理框架 AACF。由表 5 可以看出，和已有的授权与访问控制框架相比，AACF 具有明显的优势。AACF 通过扩展型分层逻辑程序表达各种具体的访问控制策略，具有良好的表达能力和语义查询效率。它通过合理的语法规范保证编写的每个访问控制策略的语义查询的计算复杂度都是多项式级的，从而对应的权限判决的计算复杂度也是多项式级的。由于 AACF 同时支持“*not*”和“¬”两种不同类型的“非”，使得 AACF 自然地具备一系列现有授权与访问控制框架不具备的高级特性：支持不完全语义条件下的授权与权限传播，支持非单调授权与访问控制逻辑的表达和语义查询，支持通过异常规则表达冲突检测与消解策略，支持通过完整性约束规则为访问控制策略提供安全监控和验证服务，等等。所有这些高级特性表明 AACF 是一个完整意义上的授权与访问控制框架，可应用于现实环境中。值得注意的是，我们从理论上保证了这些高级特性没有增加访问控制策略决策算法的计算复杂度。

参考文献

[1] Samarati P. Access control: policies, models, and mechanisms. *Foundations of Security Analysis and Design*, *Lecture Notes in Computer Science*, 2001, 2171:137-196

[2] Harrison M A, Ruzzo W L, Ullman J D. Protection in operating systems. *Communications of ACM*, 1976, 19(8):

461-471
[3] LaPadula L J, Bell D E. Secure computer systems: a mathematical model: [Technical Report], ESD-TR-278, Vol. 2, The Mitre Corporation, Bedford, MA, 1973
[4] Biba K J. Integrity considerations for secure computer systems: [Technical Report], TR-3153, the Mitre Corporation, Bedford, MA, April 1977
[5] Sandhu R S, Ferraiolo D, Kuhn R. The NIST model for bole-based access control: Towards a unified standard. In: Proceedings of the 4th ACM Workshop on Role-Based Access Control, Berlin, Germany, 2000. 47-61
[6] Woo T Y C, Lam S S. Authorizations in distributed systems: a new approach. *Journal of Computer Security*, 1993,2(2,3):107-136
[7] Jajodia S, Samarati P, Sapino M L, et al. Flexible support for multiple access control policies. *ACM Transaction on Database System*. 2001, 26(2): 214-260
[8] Gelfond M, Lobo J. Authorization and obligation policies in dynamic systems. In: Proceedings of the 24th International Conference on Logic Programming. Lecture Notes in Computer Science, Vol. 5366. Springer-Verlag, Berlin, Heidelberg, 2008, 22-36
[9] Loscocco P, Smalley S. Integrating flexible support for security policies into the Linux operating system. In: Proceedings of the FREENIX Track: USENIX Annual Technical Conference, Berkeley, USA, 2001, 29-42
[10] Lifschitz V, Morgenstern L, Plaisted D. Handbook of Knowledge Representation. New York: Elsevier, 2007
[11] Landwehr C E. A survey of formal models for computer security: [Technical Report], NRLReprt 8489, Naval Research Laboratory, 1981
[12] Halpern J, Weissman V. Using first-order logic to reason about policies. *ACM Transaction of Information System Security*, 2008, 11(4):1-41
[13] Floyd J W. Foundations of Logic Programming, 2nd. New York: Springer-Verlag Inc, 1993
[14] Gurevich Y, Neeman I. DKAL: Distributed knowledge authorization language. In: Proceedings of Computer Security Foundations Symposium, Pittsburgh, USA, 2008. 149-162
[15] Dantsin E, Eiter T, Gottlob G, et al. Complexity and expressive power of logic programming. *ACM Computer Survey*. 2001, 33(3):374-425
[16] Wagner G. Web rules need two kinds of negation. In: Proceedings of the International Workshop on Principles and Practice of Semantic Web Reasoning, Mumbai, India, 2003(2901). 33-50
[17] Gelder A V, Ross K A, Schlipf J S. The well-founded semantics for general logic programs. *J ACM*, 1991, 38(3):619-649
[18] Gelfond M, Lifschitz V. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 1991, 9:365-385
[19] Chen W, Warren D S. Tabled evaluation with delaying for general logic programs. *J ACM*. 1996, 43(1):20-74
[20] Simons P, Niemelá I, Soininen T. Extending and implementing the stable model semantics. *Artificial Intelligence*. 2002, 138(1-2): 181-234
[21] Apt K R, Blair H A. Towards a theory of declarative knowledge. In: Foundations of Deductive Databases and Logic Programming, San Francisco: Morgan Kaufmann Publishers, 1988. 89-148
[22] Bol R N. Loop checking and negation. *Journal of Logic Program*. 1993, 15(1-2):147-175
[23] Gelfond M, Lifschitz V. The stable model semantics for logic programming. In: Proceedings of the 5th International Conference on Logic Programming, Seattle, USA, 1988. 1070-1080
[24] Baral C R, Subrahmanian V S. Stable and extension class theory for logic programs and default logics. *Journal of Automated Reasoning*, 1992, 8(3):345-366
[25] Lifschitz V, Turner H. Splitting a logic program. In: Proceedings of the 11th International Conference on Logic Programming, Cambridge, USA, 1994. 23-37

AACF: a logic-based framework of nonmonotonic authorization and access control

Bao Yibao*,**, Yin Lihua*, Fang Binxing*, Guo Li*
(*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
(**Institute of Electronic Technology, PLA Information Engineering University, Zhengzhou 450004)

Abstract

On the basis of researching formally on the complex security policies with nonmonotonic authorization and access control logic under incomplete context knowledge, this paper proposes a unified logic-based framework for nonmonotonic authorization and access control, called AACF. The AACF declares nonmonotonic authorization and access control logic through the extended and stratified logic program. With the proposed syntax structure in this paper, as a full-fledged authorization and access control framework, the AACF naturally possesses some advantages such as nonmonotonic authorization, authorization propagation, and conflict checking and resolution. Furthermore, the computational complexity of the semantic query evaluation (i.e., access control decision) algorithm of the AACF proved to be polynomial. Hence, the AACF has the better expression ability and computing characteristic than the existing ones.

Key words: authorization and access control, framework, security policy, semantic, logic program