ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.4, April 2009, pp.1048−1057
doi: 10.3724/SP.J.1001.2009.03261

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于角色的管理模型隐式授权分析*

刘 伟[1,2+], 蔡嘉勇[1,2], 贺也平[1]

[1](中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

[2](中国科学院 研究生院,北京 100049)

Implicit Authorization Analysis of Role-Based Administrative Model

LIU Wei[1,2+], CAI Jia-Yong[1,2], HE Ye-Ping[1]

[1](National Engineering Research Center for Fundamental Software, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: liuwei04@ios.cn

Liu W, Cai JY, He YP. Implicit authorization analysis of role-based administrative model. *Journal of Software*, 2009,20(4):1048−1057. http://www.jos.org.cn/1000-9825/3261.htm

Abstract: Role-Based administrative models have been discussed for decentralized management in large RBAC (role-based access control) systems. The latest UARBAC model has significant advantages over other models. Due to hierarchy relationships, administrative operations of UARBAC implicate permissions. By analyzing implicit authorization, two flaws in definition and an implemental flaw in UARBAC are found, including being unable to create object, dangling reference and not supporting the least authorization. The paper corrects definitions of administrative operations for the former two. The least authorization in UARBAC is defined as the minimal role match problem. The paper proves the problem is NP-hard and gives a feasible algorithm based on greedy. The method will help the administrator use appropriate operations to achieve the least role assignment.

Key words: implicit authorization; role-based administrative model; least authorization; role-based access control

摘 要: 基于角色的管理模型被用于管理大型 RBAC(role-based access control)系统的授权关系,UARBAC 具有可扩展、细粒度等优点.UARBAC 的管理操作包含隐式授权.隐式授权分析说明 UARBAC 管理操作的两类缺陷,包括两个定义缺陷,即无法创建客体和虚悬引用,以及一个实施缺陷,即不支持最小授权.通过修改管理操作更正定义缺陷,提出实施缺陷的改进方案.定义实施最小授权的最小角色匹配问题,证明该问题是 NP 难,并给出基于贪心算法的可行方案,帮助管理员选择合适的管理操作将最小角色集合授予用户.

关键词: 隐式授权;基于角色的管理模型;最小授权;基于角色的访问控制

中图法分类号: TP309 **文献标识码**: A

在商业领域,基于角色的访问控制(role-based access control,简称 RBAC)[1]受到人们广泛的关注,成为授权

* Supported by the Science-Technology Project of the National "Tenth Five-Year-Plan" of China under Grant No.2005BA113A02 (国家"十五"攻关计划); the Graduate Innovation Grant of the Chinese Academy of Sciences (中国科学院研究生创新资金)

Received 2007-09-18; Accepted 2008-02-01

管理的重要方式之一.在实际应用中,大型组织结构复杂,角色和用户数量巨大,管理工作无法由管理员单独承担[2],需要实现分散授权.研究人员提出基于角色的管理模型[4−6],定义修改 RBAC 元素的管理操作,设置管理操作执行必须满足的条件,允许将管理权限委托给(部分)可信用户,即管理用户,减轻管理负担.

2007 年,Li 等人提出 UARBAC 模型,将用户和角色看作客体,使用统一的方式定义管理权限和访问权限,采用“RBAC 管理 RBAC”的方式,具有扩展性好、细粒度、操作明确等优点,代表基于角色的管理模型研究最新成果.

隐式授权[3]是指由显式授权所隐含的其他授权,能够减少授权规则的数量,但可能带来无法预料的授权风险.RBAC 使用角色作为用户和权限关联的中介,角色层次关系导致隐含的授权.例如,授予低层次角色的权限隐含授予高层次角色.除基本的角色层次外,UARBAC 还引入客体类别和访问模式等层次关系,造成复杂的隐含关系.

由于隐式授权的存在,管理员将管理权限授予管理用户,但不明确是否隐含授予其他权限,导致该用户能够执行的管理操作超出预期;同时,管理用户获得管理权限后执行操作授予用户访问权限,但无法了解是否能够执行隐含操作完成工作.如果把管理员看成特殊的管理用户,不区分管理和访问权限,则上述两方面统一为:当存在隐式授权时 UARBAC 如何实施最小授权.除此之外,UARBAC 中管理操作定义是否合理、操作之间是否隐含一定的关系同样不得而知.

针对上述问题,本文首先分析 UARBAC 中隐式授权的来源和形式,发现隐式授权导致模型管理操作存在两类缺陷,包括两个定义缺陷,即无法创建客体和虚悬引用,以及一个实施缺陷,即不支持最小授权.我们采用不同的方法解决上述缺陷:通过修改管理操作定义更正定义缺陷;由于隐式授权无法消除,则给出实施最小授权的可行方案.管理用户期望授予用户最少的角色满足请求,称为最小角色匹配.我们证明该问题是 NP-hard,并给出贪心算法的可行解.在获得满足权限请求的角色集合后,管理用户还需明确其是否满足执行授权操作的许可.我们给出由显式管理操作推导隐含操作的算法.

本文第 1 节简介 UARBAC.第 2 节说明 UARBAC 中隐式授权的来源和形式.第 3 节描述隐式授权导致 UARBAC 管理操作存在两类缺陷,并针对缺陷分别修改定义和给出可行方案,重点是实施最小授权.第 4 节介绍相关工作.第 5 节总结全文.

1 UARBAC

基于角色的管理模型研究 RBAC 系统中谁能执行哪些管理操作,其中的重要内容是限制管理操作的有效范围,称为管理域.ARBAC[4]是第一个被提出来的基于角色的管理模型.ARBAC(administrative RBAC)在 RBAC96 模型基础上定义独立的管理角色集合实施管理操作,包含 3 个子模型:URA97 管理用户与角色的关联;PRA97 管理角色与权限的关联; RRA97 管理角色层次关系.ARBAC 利用角色范围(role range)控制管理操作,角色层次导致用户执行管理操作对超出范围的其他部分产生影响,模型认为是非法的副作用.ARBAC 实施严格限制,禁止普通用户执行此类操作,合理的管理操作需要安全管理员的参与,这种限制损害模型的可用性,破坏分散授权的效果.

为了改善上述问题,SARBAC(scoped administrative RBAC)[5]提出更加灵活的管理范围(administrative scope)描述管理角色的控制范围.随着角色层次关系的改变,管理范围自动调整,影响用户能够执行的管理操作.SARBAC 完善了 ARBAC 的操作关系,简化管理操作内容,但使用角色层次定义管理域仅适用于一类特殊的组织结构,可扩展性较差.此外,二者没有完善描述对“管理角色”的管理,完备性有待进一步加强;两个模型定义管理操作的粒度较粗,例如增加角色的操作必须指定该角色的父、子角色,而实际应用中通常由创建角色和加入角色层次两个步骤完成.

针对上述缺点,Li 等人提出 UARBAC[6],将角色和用户看作客体,不再区分管理权限和访问权限,管理域由用户拥有的权限确定,采用“RBAC 管理 RBAC”的方式,具有可扩展、简化、细粒度等优点.UARBAC 的定义分为 RBAC 模式和相应的 RBAC 状态两部分,前者描述系统管理的客体类型以及实施的访问模式;后者描述对应

的系统状态.

定义 1. RBAC 模式由三元组⟨*C*,*OBJS*,*AM*⟩表示,其中:

- *C* 是系统支持的客体类别,包含两个预定义元素:*user* 和 *role*;
- *OBJS* 将每类客体映射到客体名称集合,即 *OBJS*(*c*)指定 *c* 的合法名称;
- *AM* 将 *C* 的每类客体映射到该类的访问模式.

对每类客体都有预定义的访问模式 *admin*,表示允许删除客体和将客体其他权限授予他人.预定义的访问模式包括 *AM*(*user*)={*empower*,*admin*}和 *AM*(*role*)={*grant*,*empower*,*admin*},其中,*empower* 和 *grant* 分别控制接受和授予权限.授权操作同时需要拥有对操作对象的 *empower* 权限和对操作目标的 *grant* 权限.

模型包含两类权限:客体权限和类别权限,前者表示为[*c*,*o*,*a*],其中,$c\in C,o\in OBJS(c),a\in AM(c)$,允许以 *a* 方式访问类 *c* 的客体 *o*;后者表示为[*c*,*a*],其中,$c\in C,a\in AM(c)\cup\{create\}$,允许以 *a* 方式访问类 *c* 的所有客体.UARBAC 定义安全管理员 SSO(system security officer)拥有所有类别权限.给定 RBAC 模式,模型给出 RBAC 状态的定义.

定义 2. RBAC 状态由四元组⟨*OB*,*UA*,*PA*,*RH*⟩表示,其中:

- *OB* 是系统中已有客体名称的有限集合,$\forall c\in C,OB(c)\subseteq OBJS(c)$;
- $U=OB(user),R=OB(role)$;
- $P=\{[c,o,a]|c\in C\wedge o\in OB(c)\wedge a\in AM(c)\}\cup\{[c,a]|c\in C\wedge a\in AM(c)\cup\{create\}\}$;
- $UA\subseteq U\times R$ 表示用户角色的关联;
- $PA\subseteq P\times R$ 表示角色权限的关联;
- $RH\subseteq R\times R$ 表示角色层次,符号≥表示角色间的支配关系.

下面举例说明.

给定 *C*={*file*,*user*,*role*},*OBJS*(*file*)包括所有合法的文件名,*OB*(*file*)是系统的文件名集合,对文件的访问模式 *AM*(*file*)={*read*,*write*,*append*,*execute*,*admin*},客体权限[*file*,"/*etc*/*syslog.conf*",*write*]表示允许写 syslog 的配置文件,类别权限[*file*,*read*]表示允许读所有文件.

UARBAC 将用户和角色看作客体,使用统一的方式描述访问权限和管理权限,明确操作的管理域,有利于通过委托实现分散授权.UARBAC 定义如表 1 所示条件限制管理操作的执行,称为权限要求.在一次会话中,用户激活若干角色,从而拥有一定的权限.只有满足权限要求的管理用户才能执行相应的操作.例如,拥有权限[*file*,"/*foo.txt*",*admin*]的用户才能执行 *deleteObject*(*file*,"/*foo.txt*")删除文件.

Table 1 Administrative operations of UARBAC

表 1 UARBAC 的管理操作

Operation	Require permissions	Conditions	Effects
createObject(c,o_1,r_1)	[*c*,*create*],[*role*,r_1,*empower*]	$o_1\in OBJS(c)$ $o_1\notin OB(c)$	$OB'(c)=OB(c)\cup\{o_1\}$ $PA'=PA\cup\{[c,o_1,admin],r_1\}$
deleteObject(c,o_1)	[c,o_1,*admin*]	$o_1\in OB(c)$	$OB'(c)=OB(c)-\{o_1\}$ Relationships about o_1 are removed
grantRoleToUser(r_1,u_1)	[*role*,r_1,*grant*],[*user*,u_1,*empower*]	$r_1\in R,u_1\in U$	$UA'=UA\cup\{(u_1,r_1)\}$
revokeRoleFromUser(r_1,u_1)	[*role*,r_1,*admin*] **or** [*user*,u_1,*admin*] **or** ([*role*,r_1,*grant*],[*user*,u_1,*empower*])	$(u_1,r_1)\in UA$	$UA'=UA-\{(u_1,r_1)\}$
grantRoleToRole(r_1,r_2)	[*role*,r_1,*grant*],[*role*,r_2,*empower*]	$r_1\not\geq r_2,(r_2,r_1)\notin RH$	$RH'=RH\cup\{(r_2,r_1)\}$
revokeRoleFromRole(r_1,r_2)	[*role*,r_1,*admin*] **or** [*role*,r_2,*admin*] **or** ([*role*,r_1,*grant*],[*role*,r_2,*empower*])	$(r_2,r_1)\in RH$	$RH'=RH-\{(r_2,r_1)\}$
grantObjPermToRole([c,o_1,a_1],r_1)	[c,o_1,*admin*],[*role*,r_1,*empower*]	$([c,o_1,a_1],r_1)\notin PA$	$PA'=PA\cup\{([c,o_1,a_1],r_1)\}$
revokeObjPermFromRole([c,o_1,a_1],r_1)	[c,o_1,*admin*] **or** [*role*,r_1,*admin*]	$([c,o_1,a_1],r_1)\in PA$	$PA'=PA-\{([c,o_1,a_1],r_1)\}$
grantClassPermToRole(p_1,r_1)	Only by the SSO *role*	$(p_1,r_1)\notin PA$	$PA'=PA\cup\{(p_1,r_1)\}$
revokeClassPermFromRole(p_1,r_1)	By the SSO *role* **or** [*role*,r_1,*admin*]	$(p_1,r_1)\in PA$	$PA'=PA-\{(p_1,r_1)\}$

2 UARBAC 中的隐式授权

UARBAC 中的角色、客体权限等层次关系产生隐含的授权关系.管理操作的对象是权限,权限的隐含关系

造成管理操作之间存在隐含关系.由于 UARBAC 在设计时仅考虑显式授权,管理操作存在若干缺陷.我们首先分析 UARBAC 中隐式授权的来源及形式.

隐式授权相关研究最初应用于数据库安全领域.由于数据库元素之间的层次关系,授权操作可能产生隐含结果,例如对数据库中某个类 *class* 的读权限隐含对其相应所有实例 *instance* 的读权限.隐式授权能够减小授权规则的规模,降低管理员的配置负担,但可能带来潜在的授权风险.隐式授权分析利用推导的方式得到显式授权操作中隐含的其他授权,有利于管理员了解授权操作产生的实际影响,减少授权风险.数据库的客体之间的层次关系是产生隐式授权的主要原因.UARBAC 同样包含角色、客体等层次关系,存在隐式授权.

UARBAC 中隐式授权的来源分为 3 部分:(1) 角色层次,角色 r_1 支配 r_2,则 r_1 隐含拥有授予 r_2 的权限,由角色层次导致的隐式授权存在于所有 RBAC 系统中;(2) 权限层次,访问模式的不同含义导致权限的隐含关系.例如,访问模式 *admin* 表示能够执行所有管理操作,隐含其他的访问模式;(3) 客体层次,UARBAC 包含客体 *o* 和类别 *C*,分别对应客体权限 *op* 和类别权限 *cp*,前者描述对指定客体的访问模式,后者描述对一类客体的访问模式.客体 *o* 与类别 *C* 的所属关系推导出权限间的隐含关系.例如,对 *role* 类客体的操作权限隐含能够对所有角色 $r\in OB(role)$执行该操作.下面给出 UARBAC 权限之间的隐含规则.

定义 3. 访问模式和权限的隐含关系分别用 $\succ$ 和 $\succeq$ 表示,权限隐含的规则为:

1. $r,r'\in OB(role)\wedge r\geq r'\rightarrow[role,r,grant]\succeq[role,r',grant]$;
2. $r,r'\in OB(role)\wedge r'\geq r\rightarrow[role,r,empower]\succeq[role,r',empower]$;
3. $c\in C\wedge admin,grant\in AM(c)\rightarrow admin\succ grant$;
4. $c\in C\wedge admin,empower\in AM(c)\rightarrow admin\succ empower$;
5. $c\in C\wedge o\in OB(c)\wedge a\in AM(c)\rightarrow[c,a]\succeq[c,o,a]$;
6. $c\in C\wedge o\in OB(c)\wedge a_1,a_2\in AM(c)\wedge a_1\succ a_2\rightarrow[c,o,a_1]\succeq[c,o,a_2]$,

其中,规则 1 和规则 2 由角色层次产生,例如管理员授予用户 *r* 隐含授予该用户 *r* 支配的所有角色,授予 *r* 权限隐含授予支配 *r* 的所有角色该权限;规则 3 和规则 4 由权限层次产生,权限 *admin* 隐含 *grant* 和 *empower*;规则 5 和规则 6 由客体层次产生,类别权限隐含对该类所有客体的对应权限.

定理 1. 权限隐含是偏序关系.

证明:权限隐含的自反性显然成立.反对称性,权限 $p\in OP\cup CP$ 只有两种类型.给定 p_1,p_2,且满足 $p_1\succeq p_2$ 和 $p_2\succeq p_1$,由类别和客体的所属关系可知 $p_1,p_2\in OP$ 或 $p_1,p_2\in CP$.考虑 $p_1=[c,o_1,a_1]$和 $p_2=[c,o_1,a_2]$,如果 $c\neq role$,则 $o_1=o_2$.由 $p_1\succeq p_2$ 和 $p_2\succeq p_1$ 得到 $a_1\succ a_2$ 和 $a_2\succ a_1$.由定义 3 可知,$a_1=a_2$.如果 $c=role$,则 $a_1=a_2$.有 $o_1\geq o_2$ 且 $o_2\geq o_1$,由角色层次的偏序性可知,$p_1=p_2$.其他情况,如 $p_1,p_2\in CP$ 等同理可证.传递性,给定 p_1,p_2,p_3 且满足 $p_1\succeq p_2$ 和 $p_2\succeq p_3$,则权限的所属关系存在多种可能,我们考虑 $p_1=[c,a_1],p_2=[c,o,a_2],p_3=[c,o,a_3]$,可知 $a_1=a_2\succeq a_3$,因此, $p_1\succeq p_3$.其他情况,如 $p_1,p_2,p_3\in OP$ 等同理可证.综上,命题得证. □

角色层次同样属于偏序关系,因此,根据定理 1,UARBAC 中的权限隐含关系具有传递性.例如将[*file*,*admin*]授予 *r*,对于 $r'\geq r$,则担任 r'的用户能够执行[*file*,"*/etc/syslog.conf*",*write*].在 RBAC 中,权限与角色关联,用户通过与角色关联获得权限,用户和角色本质上都可以看成权限的集合.将用户、角色和权限统一表示为客体权限集合,管理操作对权限集合产生影响,由此获得管理操作的隐含关系.我们定义 *own* 函数表示授权实体的权限集合.

定义 4. $p=OP\cup CP$,$S=U\cup R\cup P$.函数 $own:S\rightarrow 2^{OP}$ 满足:

1. $u\in U,own(u)=\{op\mid op\in OP,(u,r_1)\in UA\wedge(r_1,r_2)\in RH\wedge(r_2,op)\in PA\}$;
2. $r\in R,own(r)=\{op\mid op\in OP,(r_1,r_2)\in RH\wedge(r_2,op)\in PA\}$;
3. $p\in P,own(p)=\{op\mid op\in OP,p\succeq op\}$.

例如,操作 *grantRoleToUser*(r_1,*u*)增加用户 *u* 的权限集合 $P_u=own(u)\cup own(r_1)$,操作 *grantRoleToUser*(r_2,*u*)增加用户 *u* 的权限集合为 $P'_u=own(u)\cup own(r_2)$.当 r_1 支配 r_2 时,由 $own(r_1)\supseteq own(r_2)$可知, $P_u\supseteq P'_u$.因此, *grantRoleToUser*(r_1,*u*)隐含授权 *grantRoleToUser*(r_2,*u*).根据表 1 中管理操作对用户、角色集合的影响,我们获得管理操作的隐式授权推导规则[7].由于使用符号 $\succeq$ 表示权限的隐含关系,后文不再区分客体权限和类别权限,统一使用 *grantPermToRole*,*revokePermToRole* 表示对 *PA* 关系的修改.这里我们主要讨论授权操作的隐含关系,撤

销操作同样存在隐含关系减小权限集合,其安全问题可以由授权操作表示.我们根据角色层次和权限隐含获得管理操作的隐含规则.

定义 5. 管理操作的隐式授权满足以下规则:

1. $(r_1 \geq r_2) \wedge grantRoleToUser(r_1,u) \rightarrow grantRoleToUser(r_2,u)$;
2. $(r_2 \geq r_3) \wedge (u,r_1) \in UA \wedge grantRoleToRole(r_2,r_1) \rightarrow grantRoleToUser(r_3,u)$;
3. $(r_1 \geq r_2) \wedge (r_3 \geq r_4) \wedge grantRoleToRole(r_3,r_2) \rightarrow grantRoleToRole(r_4,r_1)$;
4. $(r_1 \geq r_2) \wedge (r_3 \geq r_4) \wedge (r_4,p_1) \in PA \wedge (p_1 \succeq p_2) \wedge grantRoleToRole(r_3,r_2) \rightarrow grantPermToRole(p_2,r_1)$;
5. $(p_1 \succeq p_2) \wedge grantPermToRole(p_1,r) \rightarrow grantPermToRole(p_2,r)$.

规则 1 和规则 2 描述,当存在角色层次关系时,增加 *UA* 和 *RH* 关系都会修改用户的权限集合;规则 3 和规则 4 描述,修改角色层次关系对与其相关的 *RH* 和 *PA* 关系的影响;规则 5 描述权限隐含同样导致管理操作的隐含关系.需要注意的是,上述规则仅列出单步隐含关系,多个规则间可能存在隐含推导序列.详细讨论见第 3.2 节.

3 隐式授权造成的缺陷及改进

通过对 UARBAC 中管理操作的隐式授权规则分析,我们可以判断管理操作之间的关系,发现 UARBAC 中管理操作存在若干缺陷,包括两个定义缺陷,即拥有创建客体权限普通用户也无法创建客体,删除客体操作可能造成虚悬引用,以及一个实施缺陷,即权限要求不支持最小授权.对于不同缺陷,我们采取相应的办法解决,修改管理操作更正定义缺陷.由于角色和客体类别等层次关系无法消除,隐式授权的存在不可避免.我们提出针对最小授权实施缺陷的可行方案.

3.1 管理操作的缺陷

UARBAC 中存在的隐式授权导致模型存在两个方面的缺陷:定义缺陷和实施缺陷,前者是由于管理操作定义没有考虑管理操作之间的隐含关系导致,后者是由于管理操作仅涉及显式的权限要求造成无法执行隐含操作.由于隐式授权不涉及 *OB* 集合,我们省略 *OB* 集合的相关内容.

3.1.1 定义缺陷 1:无法创建客体

根据 UARBAC 的定义,拥有类别权限[*c*,*create*]的用户能够执行操作 *createObject*(c,o_1,r_1)创建类别为 *c* 的客体 o_1,并将对 o_1 的管理权限授予 r_1.由于该操作包含将权限授予其他角色的管理操作,而二者的权限要求定义存在矛盾,造成除 SSO 外的用户无法创建客体.

定理 2. UARBAC 中创建客体操作只能由 SSO 执行.

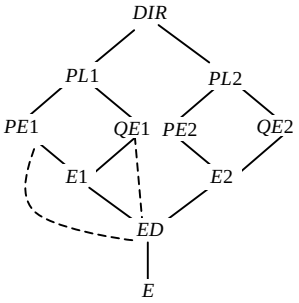
证明:*createObject*(c,o_1,r_1)执行修改 r_1 的权限集合为 $own(r_1) \cup \{[c,o_1,admin],r_1\}$,其权限要求为[*c*,*create*]和[*role*,r_1,*empower*].由于不考虑前者修改 *OB* 集合,并且在创建客体时用户无法拥有权限[c,o_1,*admin*],因此权限要求集合为 $req_{create}=\{\neg[c,o_1,admin],[role,r_1,empower]\}$.由隐式授权推导规则 5,*grantObjPermToRole*($[c,o_1,a],r_1$)修改 r_1 的权限集合 $own(r_1) \cup \{[c,o_1,a],r_1\}$隐含上述授权,其权限要求集合 $req_{addPA}=\{[c,o_1,admin],[role,r_1,empower]\}$,两个权限要求集合产生矛盾.因此,普通用户无法执行创建客体操作,而只有 SSO 拥有所有权限,能够创建客体. □

定理 2 显然与分散授权的目标不一致,产生此缺陷的原因是 UARBAC 的创建客体操作定义的粒度过粗,包含创建客体和将对客体的管理权限授予其他角色两个功能.由于 RBAC 中用户不能直接与权限关联,无法获得对创建客体的管理权限,而将对该客体的管理权限授予其他角色的操作需要该权限,从而导致矛盾的产生.

3.1.2 定义缺陷 2:虚悬引用

隐式授权导致 UARBAC 中删除角色操作可能造成虚悬引用.考虑角色客体的情况,即 *deleteObject*(*role*,*r*).表 1 中 UARBAC 要求删除客体操作后必须删除与其相关的内容.例如,删除某个角色后仍需删除该角色与用户、权限的关联关系.对于文件或用户来说,删除客体后续操作能够保持系统的一致性,但这种要求对于角色客体并不充分,因为删除角色可能产生角色层次的破坏,ARBAC 称为虚悬引用(dangling reference).在如图 1 所示的角色层次中,删除角色 *E*1 导致 *PE*1 和 *QE*1 无法继承 *ED* 和 *E* 的权限.由于角色层次的隐含关系,*PE*1 和 *QE*1

支配 *E*1,而 *E*1 支配 *ED*,隐含 *PE*1 和 *QE*1 支配 *ED*.为了维护角色层次的正常结构,需要增加图中两条虚线所示的角色层次关系.在这种情况下,操作 *deleteObject*(*role*,*E*1)直接影响 *PE*1 和 *QE*1 的权限集合,需要执行隐含操作 *grantRoleToRole*(*ED*,*PE*1)和 *grantRoleToRole*(*ED*,*QE*1).删除角色操作要求管理用户拥有权限[*role*,*r*,*admin*],这种显式的权限要求无法满足维护角色层次的需求,造成虚悬引用.产生此缺陷的原因是 UARBAC 将用户和角色看作客体,但删除客体操作将角色与其他客体采用同样处理方式,没有考虑角色层次关系的特殊性.



Role	Explicit assignment	Role	Explicit assignment
ED	p_1:[*file*, *company_doc*,*read*]	*E*2	p_4:[*file*,p_1_*design*,*write*] p_6:[*file*,p_1_*test*,*write*] p_9:[*file*,*company_dev*,*write*] p_{10}:[*file*,p_2_*design*,*read*] p_{11}:[*file*,p_2_*test*,*read*]
*E*1	p_2:[*file*,p_1_*design*,*read*] p_3:[*file*,p_1_*test*,*read*]		
*PE*1	p_4:[*file*,p_1_*design*,*write*] p_5:[*file*,*company_dev*,*read*]		
*QE*1	p_5:[*file*,*company_dev*,*read*] p_6:[*file*,p_1_*test*,*write*]	*PE*2	p_{12}:[*file*,p_2_*design*,*write*]
*PL*1	p_7:[*file*,p_1_*design*,*admin*] p_8:[*file*,p_1_*test*,*admin*]	*QE*2	p_{13}:[*file*,p_2_*test*,*write*]
		DIR	p_{14}:[*file*,*comany_dev*,*admin*]

Fig.1 Example of role hierarchy

图 1 角色层次关系示例

3.1.3 实施缺陷:不支持最小授权

最小授权是指系统应该授予用户完成工作必须的最小权限集合,也称为最小特权原则,是授权管理必须遵守的重要原则之一.在 UARBAC 中,隐式授权导致管理操作之间存在隐含关系,授予他人的管理权限可能隐含其他的权限.如果隐含权限能够满足用户请求,将其授予用户能够降低权限滥用和泄漏的风险,实现最小授权.另一方面,管理用户能够执行的管理操作受到其拥有权限的限制,只有拥有权限才能执行相应的管理操作.管理用户尽管可以通过隐含方式获得权限,但其权限集合不满足权限要求导致无法实施最小授权,从而产生矛盾.

例 1:在图 1 所示的组织结构中,SSO 将角色的管理权限[*role*,{*PE*1,*QE*1},*admin*]和为实习生 *intern* 授权的权限[*user*,*intern*,*empower*]委托授予项目经理角色 *PL*1,实现对该项目的分散授权.实习生 *intern* 仅需要查看项目文档,由角色权限的关联关系可知,角色 *E*1 即可满足.根据管理操作隐含规则 1,由 *PE*1 支配 *E*1 可知,*grantRoleToUser*(*PE*1,*intern*)隐含 *grantRoleToUser*(*E*1,*intern*).而担任 *PL*1 的用户 *pl*1 没有权限[*role*,*E*1,*grant*],只能将 *PE*1 或 *QE*1 授予 *intern*,使得该用户间接获得查看文档的权限,因而违反了最小特权原则.

产生此缺陷的原因是 UARBAC 中提出的权限要求没有考虑隐式授权,用户拥有的权限只能满足显式管理操作需要.另一方面,管理操作间的隐含关系客观存在,隐含的管理操作与显式的权限要求之间产生矛盾.

3.2 定义缺陷的修正

定义缺陷可以通过修改管理操作实现更正.对于缺陷 1,由于 RBAC 中用户无法直接与权限关联,UARBAC 创建客体后无法将客体的管理权限直接授予创建用户,只能转而授予其他角色.我们将客体的所属关系由用户改为角色,即客体的属主是角色而非用户,由属主执行相关管理操作.这种修改是合理的,与实际应用的场景相一致.例如,担任 *PE*1 的用户 *pe*1 创建的程序应该由 *PE*1 而非 *pe*1 实施管理,以便该项目中其他用户访问.在一次会话中,用户 *u* 激活角色 *r* 拥有权限[*c*,*create*],执行操作创建类别 *c* 的客体 *o*,*r* 自动获得权限[*c*,*o*,*admin*].对于 *u* 激活的多个角色 $r_1,...,r_n$ 都拥有权限[*c*,*create*]的情况,我们要求 *u* 明确指定属主角色.因此,创建客体操作修改为 *createObject*(*c*,*o*),其内容减少了授权部分,语义更加明确,拥有权限[*c*,*create*]的用户均允许执行,避免缺陷 1 的产生.需要说明的是,我们的修改并不等同于以下操作:*u* 激活 *r*,执行 *createObject*(*c*,*o*,*r*).此操作仍然要求[*role*,*r*,*empower*]∈*own*(*r*),而修改后的操作 *createObject*(*c*,*o*)是一种类似自主访问控制的方式实施.

缺陷 2 是由于删除角色需要执行后续操作导致,显式的权限要求无法满足管理操作的需求.我们修改 UARBAC 的权限要求描述,规定由隐式的管理操作不受权限要求的约束.下面分析隐含的权限要求是否能够得到满足.UARBAC 的权限定义要求执行操作 *deleteObject*(*role*,*r*)的管理用户 *u* 拥有权限[*role*,*r*,*admin*].由访问模

式的隐含关系可知,u 隐含拥有[*role*,*r*,*grant*]和[*role*,*r*,*empower*].根据角色层次关系以及定义 3 的规则 1 和规则 2,该用户同样隐含拥有权限{[*role*,*r*′,*grant*]|*r*≥*r*′}和{[*role*,*r*′,*empower*]|*r*′≥*r*}.如图 1 所示,消除删除 *E*1 产生的虚悬引用需要增加角色层次(*PE*1,*ED*)和(*QE*1,*ED*),所需的权限为{[*role*,{*PE*1,*QE*1},*empower*],[*role*,*ED*,*grant*]}.由隐含关系可知,u 隐含拥有的权限要求集合能够满足执行操作 *grantRoleToRole*(*ED*,*PE*1)和操作 *grantRoleToRole*(*ED*, *QE*1).我们定义管理用户在执行删除角色操作 *deleteObject*(*role*,*r*)后执行相关的角色层次的后续操作,避免产生虚悬引用.

3.3 实施缺陷的可行方案

对于实施缺陷,支持最小特权的授权操作要求相对复杂.通常情况的授权流程如下:用户提出权限请求,管理用户授予其恰当的角色,从而用户间接获得相应权限完成工作.由于存在隐式授权,管理用户首先判断哪些角色是“恰当”的,即能够最小满足权限请求;然后再明确自己是否拥有将这些角色授予用户的“恰当”能力,即最小管理操作集合实施授权.

3.3.1 满足访问请求的最小角色集合

授权的最终目标是用户获得对资源的访问权限,即客体权限,但实施授权的管理用户需要将“恰当”的角色授予用户.如果角色拥有的隐式授权过多,则可能造成权限的滥用和扩散;反之,权限过少则无法完成工作.例 1 中,用户 *intern* 需要权限 p_2:[*file*,p_1_*design*,*read*],多个管理操作均能实现该目标.例如,授予 *intern* 角色 *PE*1 或 *QE*1,但 *PL*1 希望将满足 p_2 的最小角色授予 *intern* 以降低授权风险.下面给出最小角色的定义.

定义 6. 给定 RBAC 状态γ,客体权限 p 的最小角色 r 是所有隐含 p 的角色中基数最小的,表示为

$$p \in OP, r' \in R, p \in own(r') \rightarrow |own(r')| \geq |own(r)|.$$

由定义可知,客体权限的最小角色可能为多个.此时,管理用户可以根据具体的安全策略选择授予用户.用户提出的权限请求可能包括多个客体权限,表示为集合 $OPS=\{op_1,...,op_n\}$.理想的情况是,管理用户找出满足权限请求且包含的客体权限数量最少的最小角色集合 $MRS=\{r_1,...,r_m\}$,将其授予用户,我们称其为最小角色匹配问题.

定理 3. 给定 RBAC 状态γ,最小角色匹配问题是 NP-hard 的.

证明:我们将上述问题归约为最小加权集合覆盖问题 MSVC(minimum sum vertex cover).根据 RBAC 状态γ计算每个角色 $r \in R$ 的客体权限集合 *own*(*r*),其权重 *w*(*r*)=|*own*(*r*)|.最小角色匹配问题等价于选择客体权限的若干子集 $MRS=\{r_1,...,r_m\}$覆盖客体权限集合 $OPS=\{op_1,...,op_n\}$,即 $\bigcup_{r \in MRS} own(r) \supseteq OPS$,并且 $\sum_{r \in MRS} w(r)$ 最小.容易证明,上述归约中给定γ和 *r* 计算 *own*(*r*)在多项式时间内完成.已知 MSVC 是 NP-hard 的,因此命题得证. □

由于 MSVC 是 NP-hard 的,研究人员利用贪心算法给出 MSVC 的可行解,具有近似比 2ln*n*+1.与 MSVC 相比,最小角色匹配问题不要求精确匹配,即 MRS 的客体权限集合包含 OPS,并非相等,因此无法直接使用贪心算法.我们利用算法 1 给出最小角色匹配问题的可行解[8],其基本思想是:首先找出与 OPS 相关的角色集合 R_Q,根据角色的权重与对客体权限的涵盖性之间的关系递归选择角色集合,直至所选角色集合的权限包含 OPS.

算法 1. 最小角色匹配问题的可行解.

输入:RBAC 状态和客体权限集合 OPS;

输出:MRS 的可行解.

1. $total\_OP=R_Q=MRS=\varnothing$;$select\_OP=OPS$;
2. for all *r* in *R* do
3. if $own(r) \cap OPS \neq \varnothing$ then
4. $R_Q=R_Q \cup r$
5. while $|R_Q|>0$ and $OPS \cap total\_OP \neq OPS$ do
6. if $select\_OP \cap own(r) \neq \varnothing$ then
7. select minimum $t(r)=w(r)/|select\_OP \cap own(r)|$ of all *r* in R_Q

8. $MRS=MRS\cup r;R_Q=R_Q-\{r\};select\_OP=select\_OP-own(r)$

9. return *MRS*

上述算法循环次数等于$|R_Q|$,该集合需要计算所有角色的客体权限集合,等价于计算角色层次关系的传递闭包,由 Warshall 算法计算具有多项式时间的复杂度 $O(|R|^3)$.在如图 1 所示的组织结构中,用户 *consultant* 要求能够读写公司开发计划并修改项目 1 的测试文档,即 $OPS=\{p_5,p_6,p_9\}$.为了满足上述授权要求,最简单的方案是管理员执行 *grantRoleToUser*(*DIR*,*consultant*),将角色 *DIR* 授予 *consultant*,显然其拥有的权限过多,不符合最小特权原则.理想的方案是实现最小角色匹配,对于 NP-hard 问题没有多项式时间的最优解,只能给出可行解.根据算法 2,与 OPS 相关的角色集合 $R_Q=\{PE1,QE1,PL1,E2,PE2,QE2,PL2,DIR\}$,分别计算其中每个角色的 $t(r)$,选择最小的 $t(QE1)$,递归选择 $t(E2)$,获得 $MRS=\{QE1,E2\}$.

3.3.2 管理操作隐式授权查询

由于隐式授权的存在,管理用户可能拥有某个管理权限(隐含其他权限),但隐含权限的要求该用户无法满足,导致该用户无法执行满足最小授权的管理操作.另一方面,在授权时管理用户并不了解执行该操作是否隐含能够执行其他管理操作,无法明确授权操作的风险.我们修改模型的权限要求,描述为:由隐式授权的管理操作不受权限要求的约束.根据定义 5 的隐含规则,如果 RBAC 状态γ满足 $cond_1$ 和 $cond_2$,由于 $rule_1:cond_1\wedge op_1\rightarrow op_2$ 后置谓词满足 $rule_2:cond_2\wedge op_2\rightarrow op_3$,形成隐含规则的传递序列,即操作 op_1 隐含 op_2 和 op_3.由此类序列,管理员由算法 2 获得管理操作的所有隐含操作.根据最小角色匹配问题的可行解,选择最小管理操作集合实施授权.

算法 2. 推导管理操作 $admin(arg_1,arg_2)$的隐含操作集合.

输入:RBAC 状态$\gamma(UA,PA,RR)$和管理操作 $admin(arg_1,arg_2)$;

输出:隐含的管理操作集合 IAS.

1. $farg(admin)=arg_1;sarg(admin)=arg_2;$

2. $junior(r)=\{r'\in R\mid r\geq r'\}$; $senior(r)=\{r'\in R\mid r'\geq r\}$; $objperm(p)=\{op\in OP\mid p\succeq op\}$

3. $IAS=\{admin(arg_1,arg_2)\}$;

4. if ($admin==grantRoleToRole$) then

5. $IAS=IAS\cup\{grantRoleToRole(r_2,r_1)\mid r_1\in senoir(sarg(admin)),r_2\in junoir(farg(admin))\}$

6. if $(u,senior(sarg(admin)))\in UA$ then

7. $IAS=IAS\cup\{grantRoleToUser(junior(farg(admin)),u)\}$

8. if $(p,junior(farg(admin)))\in PA$ then

9. $IAS=IAS\cup\{grantPermToRole(objperm(p),senior(sarg(admin)))\}$

10. else if ($admin==grantRoleToUser$) then

11. $IAS=IAS\cup\{grantRoleToUser(junior(farg(admin)),sarg(admin))\}$

12. else if ($admin==grantPermToRole$) then

13. $IAS=IAS\cup\{grantPermToRole(objperm(farg(admin)),senior(sarg(admin)))\}$

14. return *IAS*.

例 1 中,*PL*1 拥有权限[*role*,*PE*1,*grant*],能够执行授予 *intern* 角色 *PE*1 的操作 *grantRoleToUser*(*PE*1,*intern*),使用算法 2 获得相应的 $IAS=\{grantRoleToUser(\{PE1,E1,ED,E\},intern)\}$,执行实施满足授权要求的管理操作 *grantRoleToUser*(*E*1,*intern*),满足最小特权原则.

4 相关工作

隐式授权分析方法最早应用于数据库领域.文献[3]研究授权主体、类型及客体导致的隐式授权,提出由显式授权获得隐式授权的推导规则,讨论隐式授权的主要应用方向.文献[9]列举 IRO(relational and object-oriented)数据库中隐式授权的 9 条授权规则,分别由角色、授权类型、粒度、类别等层次关系产生.上述研究虽然仅针对数据库系统,并且没有具体推导算法及应用实例,但仍然为本文提供很好的借鉴.例如,UARBAC 中隐

式授权的产生原因:角色层次、访问类型和客体类别分别对应授权主体、类型和客体.与其相比,本文应用隐式授权分析方法发现 UARBAC 中的缺陷并根据推导规则实现最小授权,具有较好的可操作性和实用性.

在管理模型中,用户执行管理操作往往凭借主观经验,管理效果很难判断.Li 提出安全性分析[10]的概念,用于回答一类安全查询,例如多次管理操作后是否每个用户都拥有某个权限,或某个系统状态下用户拥有某个权限.研究人员根据 RBAC 中若干问题进行安全性分析,涉及 ARBAC 中的 URA97,PRA97[11]等模型,以及可信用户等问题.安全性分析能够为策略配置提供理论基础,关心管理操作的结果是否满足安全策略的预定要求.与其目的不同,本文研究管理操作之间的隐含关系,用于分析管理模型定义和实施的合理性.

研究人员将委托的概念引入 RBAC,提出基于角色的委托模型.RBDM(role-based delegation model)[12]研究委托实施条件、作用范围和与角色层次的相互影响,提供细粒度、基于权限的委托操作.文献[13]提出的委托模型支持传递委托,防止委托权限的恶意扩散和非法滥用.从委托的权限类型上看,我们认为模型可以分为两类:对个体权限的委托和对管理权限的委托.前者可以看作自主访问控制的扩展,委托意愿由资源属主自己控制;后者的前提是对管理模型的定义,委托管理权限中的隐式授权可能对系统安全带来威胁.本文的隐式授权分析方法同样能够应用于委托模型.

5 结 论

由于角色、权限和客体等层次关系,最新的基于角色的管理模型 UARBAC 中仍然存在复杂的隐式授权关系.本文首次针对 UARBAC 实施隐式授权分析,发现 UARBAC 管理操作存在缺陷,包括定义缺陷(即普通用户即使拥有创建客体权限也无法创建客体,删除客体可能造成虚悬引用)和实施缺陷,即权限要求定义违反最小特权原则.定义缺陷通过修改管理操作更正,实施缺陷则需要通过提供实施方案来解决.层次关系导致隐式授权的存在不可避免.对于用户提出的权限请求,我们证明满足请求的最小角色匹配问题是 NP-hard 的,并给出基于贪心算法的可行解.我们给出管理操作的查询算法,管理员根据已有的权限执行该算法获得能够执行的管理操作集合,将满足最小角色匹配问题的角色集合授予用户,实施最小授权.

References:

[1] Sandhu R, Coyne E, Feinstein H, Youman C. Role-Based access control models. IEEE Computer, 1996,29(2):38−47.

[2] Schaad A, Moffett J, Jacob J. The role-based access control system of a European bank: A case study and discussion. In: Sandhu R, Jaeger T, eds. Proc. of the 6th ACM Symp. on Access Control Models and Technologies. Chantilly: ACM Press, 2001. 3−9.

[3] Rabitti, F, Bertino E, Kim W, Woelk D. A model of authorization for next-generation database systems. ACM Trans. on Database Systems, 1991,16(1):88−131.

[4] Sandhu R, Bhamidipati V, Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans. on Information and Systems Security, 1999,2(1):105−135.

[5] Crampton J, Loizou G. Administrative scope: A foundation for role-based administrative models. ACM Trans. on Information and System Security, 2003,6(2):201−231.

[6] Li NH, Mao ZQ. Administration in role-based access control. In: Deng R, Samarati P, eds. Proc. of the 2nd ACM Symp. on Information, Computer and Communications Security. Singapore: ACM Press, 2007. 127−138.

[7] Dekker M, Cederquist JG, Crampton J, Etalle S. Extended privilege inheritance in RBAC. In: Deng R, Samarati P, eds. Proc. of the 2nd ACM Symp. on Information, Computer and Communications Security. Singapore: ACM Press, 2007. 383−385.

[8] Chen L, Carmpton J. Inter-Domain role mapping and least privilege. In: Lotz V, Thuraisingham BM, eds. Proc. of the 12th ACM Symp. on Access Control Models and Technologies. Sophia Antipolis: ACM Press, 2007. 157−162.

[9] Emayr W, Kastner F, Pernul G, Preishuber S, Tjoa A. Authorization and access control in IRO-DB. In: Su SYW, ed. Proc. of the 20th Int'l Conf. on Data Engineering. New Orleans: IEEE Computer Society, 1996. 40−47.

[10] Li NH, Tripunitara MV. Security analysis in role-based access control. ACM Trans. on Information and System Security, 2006,9(4): 391−420.

[11] Yang QW, Hong F, Yang MX, Zhu X. Security analysis on administrative model of role-based access control. Journal of Software, 2006,17(8):1804−1810 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1804.htm

[12] Zhang XW, Oh S, Sandhu R. PBDM: A flexible delegation model in RBAC. In: Ferrari E, ed. Proc. of the 8th ACM Symp. on Access Control Models and Technologies. Como: ACM Press, 2003. 149−157.

[13] Crampton J, Khambhammettu H. Delegation in role-based access control. In: Gollmann D, Sabelfeld A, eds. Proc. of the 11th European Symp. on Research in Computer Security. Hamburg: Springer-Verlag, 2006. 174−191.

附中文参考文献:

[11] 杨秋伟,洪帆,杨木祥,朱贤.基于角色访问控制管理模型的安全性分析.软件学报,2006,17(8):1804−1810. http://www.jos.org.cn/1000-9825/17/1804.htm

刘伟(1979－),男,河南信阳人,博士生,主要研究领域为系统软件,安全操作系统.

贺也平(1962－),男,博士,研究员,博士生导师,主要研究领域为网络安全,安全协议.

蔡嘉勇(1978－),男,博士生,主要研究领域为系统软件,安全操作系统.