带时间特性的角色访问控制*

黄 建+, 卿斯汉, 温红子

(中国科学院 软件研究所,北京 100080)
(中国科学院 信息安全技术工程研究中心,北京 100080)

Timed Role-Based Access Control

HUANG Jian+, QING Si-Han, WEN Hong-Zi

(Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)
(Engineering and Research Center for Information Security Technology, The Chinese Academy of Sciences, Beijing 100080, China)
+ Corresponding author: E-mail: stanleyhuangjian@hotmail.com
http://www.ercist.ac.cn
Received 2002-08-16; Accepted 2003-04-16

Huang J, Qing SH, Wen HZ. Timed role-based access control. *Journal of Software*, 2003,14(11):1944~1954.
http://www.jos.org.cn/1000-9825/14/1944.htm

Abstract: The research work of RBAC is greatly emphasized in recent years. However, the main work focuses on some characters which have nothing to do with the time character. A RBAC model with time character called TRBAC is described and the former constraint is extended so that it can describe the time character. Some algorithms are developed to solve the state change problem of the time-constraint and the sessions. The consistent state of the model is analyzed and the problem for maintaining the consistent state is discussed.
Key words: information security; access control; role; constraint; time; consistency

摘 要: 基于角色的访问控制模型的研究工作近年来得到了广泛的重视,但主要工作均立足于与时间特性无关的其他方面.形式化描述了一个引入时间后的角色访问控制模型.在该模型中对原有授权约束进行了时间扩充.提出的相应算法解决了时间授权约束和会话的状态转变问题.同时分析了模型的一致性状态,并讨论了一致性状态维护问题.
关键词: 信息安全;访问控制;角色;约束;时间;一致性
中图法分类号: TP309 **文献标识码**: A

访问控制是信息安全领域中一个基础性的核心问题.本文要讨论的主题是基于一种较新的访问控制模型:基于角色的访问控制(role-based access control,简称 RBAC).在 RBAC 之中,权限被赋予角色,而不是用户.当一个角色被指定给一个用户时,此用户就拥有了该角色所包含的权限.而 RBAC 的约束(constraint),规定了权限被赋

* Supported by the National Natural Science Foundation of China under Grant No.60083007 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.GG1999035810 (国家重点基础研究发展规划(973))
第一作者简介: 黄建(1976－),男,安徽合肥人,硕士生,主要研究领域为信息安全技术.

予角色时,或角色被赋予用户时,以及当用户在某一时刻激活一个角色时所应遵循的强制性规则.在 NIST 的标准 RBAC 模型中,约束包括静态约束和动态约束两类.约束与用户-角色-权限关系一起决定了 RBAC 模型中用户的访问许可.

RBAC 访问控制模型不仅易于管理而且降低了复杂性、成本和发生错误的概率,因而近年来得到了极大的发展.文献[1]提出了 NIST 标准 RBAC 模型,统一了人们对 RBAC 的认识.在此基础上,人们提出多个扩充模型.文献[2,3]对 RBAC 模型的描述能力进行了扩充,因而扩展了该访问控制模型的适用范围.

现有的扩充 RBAC 模型都没有引入时间概念,因而首先模型中没有考虑随时间变化而引起模型的动态变化并且其授权约束与时间无关.这两点极大地削弱了系统的安全性和模型的描述能力.在本文中,我们在无时间特性的角色访问控制的形式化表达的基础上,作时间特性的扩展,以利用已有的研究成果.第 1 节介绍NIST标准RBAC 模型[1].第 2 节对授权约束及其时间特性进行分析.在第 3 节提出一个带时间特性的角色访问控制模型,它是对 RBAC 标准模型的扩展,称为 TRBAC(timed role-based access control).第 4 节讨论了 TRBAC 的安全状态及安全状态维护.第 5 节总结全文.

1 RBAC 模型

NIST 标准 RBAC 模型由 4 个部件模型组成,这 4 个部件模型分别是 Core RBAC,Hierarchal RBAC 和 Constraint RBAC 中的两个责任分离部件模型.以下是这 4 个部件模型的形式化描述.

1.1 Core RBAC

Core RBAC 定义了能构成一个 RBAC 控制系统的最小的元素集合.

定义 1(**Core RBAC**).

$Users=\{u_1,u_2,\ldots,u_m\}$所有用户(user)的集合.

$Roles=\{r_1,r_2,\ldots,r_n\}$所有角色(role)的集合.

$Ops=\{op_1,op_2,\ldots,op_k\}$所有操作(operation)的集合.

$Objects=\{ob_1,ob_2,\ldots,ob_l\}$所有访问对象的集合.

$Sessions=\{s_1,s_2,\ldots,s_p\}$所有会话(session)的集合.

$Perms=2^{(Ops\times Objects)}$所有权限(permission)的集合.

$UA\subseteq Users\times Roles$ 从用户集合到角色集合的多对多映射,表示用户被赋予的角色.

$PA\subseteq Perms\times Roles$ 从权限集合到角色集合的多对多映射,表示角色被赋予的权限.

$assigned\_users(r:Roles)\rightarrow 2^{Users}$ 返回指定给角色的用户集.$assigned\_users(r)=\{u\in Users|(u,r)\in UA\}$.

$assigned\_perms(r:Roles)\rightarrow 2^{Perms}$ 返回指定给角色的权限集.$assigned\_perms(r)=\{p\in Perms|(p,r)\in PA\}$.

$assigned\_roles(u:Users)\rightarrow 2^{Roles}$ 返回指定给用户的角色集合. $assigned\_roles(u)=\{r\in Roles\mid(u,r)\in UA\}$.

$op(p:Perms)\rightarrow Ops$ 返回与指定权限相关的操作. $op(p)=\{op\in Ops\mid\exists obj\in Objects\cap(op,obj)=p\}$.

$ob(p\in Perms)\rightarrow Objects$ 返回与指定权限相关的操作对象. $ob(p)=\{ob\in Objects\mid\exists op\in Ops\cap(op,obj)=p\}$.

$user\_sessions(u\in Users)\rightarrow 2^{Sessions}$ 返回指定用户相关的会话.

$session\_roles(s:sessions)\rightarrow 2^{Roles}$ 返回与指定会话相关的角色.即

$$session\_roles(s)\subseteq\{r\in Roles\mid(session\_users(s),r)\in UA\}.$$

$session\_perms(s:sessions)\rightarrow 2^{Perms}$ 返回与指定会话相关的权限.即

$$session\_perms(s)=\bigcup_{r\in session\_users(s)}assigned\_perms(r).$$

1.2 Hierarchal RBAC

Hierarchal RBAC 引入角色间的继承关系,角色间的继承关系又可分为一般继承关系和受限继承关系.一般继承关系仅要求角色继承关系是一个绝对偏序关系,它允许角色间的多继承.而受限继承关系则进一步要求角色继承关系是一个树结构.

定义 2a(General Hierarchal RBAC,一般继承关系).

$RH \subseteq Roles \times Roles$ 是角色集合上的一个偏序关系,记作 $\geq$.即

$r_1 \geq r_2 \Rightarrow authorized\_perms(r_2) \subseteq authorized\_perms(r_1) \cap authorized\_users(r_1) \subseteq authorized\_users(r_2)$.

$authorized\_users(r \in Roles) \rightarrow 2^{Users}$ 有 $authorized\_users(r)=\{u \in Users | \exists r' \in Roles, r' \geq r \cap (u,r') \in UA\}$.

$authorized\_perms(r \in Roles) \rightarrow 2^{Perms}$ 有 $authorized\_perms(r)=\{p \in Perms | \exists r' \in Roles, r' \geq r \cap (p,r') \in PA\}$.

定义 2b(Limited Hierarchal RBAC,受限继承关系).

在定义 2a 的基础上添加以下限制:

$$\forall r, r_1, r_2 \in Roles, r \geq r_1 \cap r \geq r_2 \Rightarrow r_1 = r_2.$$

1.3 Constraint RBAC

Constraint RBAC 在 RBAC 模型中添加了责任分离关系(separation of duty relationship).责任分离包括静态责任分离(static separation of duty,也称静态授权约束)和动态责任分离(dynamic separation of duty,也称动态授权约束).

定义 3a(静态责任分离).

$SSD \subseteq 2^{Roles} \times N$.该关系满足 $\forall (rs,n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \bigcap_{r \in t} assigned\_users(r) = \varnothing$.

定义 3b(继承关系下的静态责任分离).

$SSD \subseteq 2^{Roles} \times N$ 该关系满足 $\forall (rs,n) \in SSD, \forall t \subseteq rs : |t| \geq n \Rightarrow \bigcap_{r \in t} authorized\_users(r) = \varnothing$.

定义 4(动态责任分离).

$DSD \subseteq 2^{Roles} \times N$ 该关系满足 $\forall rs \in 2^{Roles}, n \in N, (rs,n) \in DSD \Rightarrow n \geq 2 \cap |rs| \geq n$,并且

$$\forall s \in Sessions, \forall rs \in 2^{Roles}, \forall role\_set \in 2^{Roles}, \forall n \in N, (rs,n) \in DSD,$$
$$role\_set \subseteq rs, role\_set \subseteq session\_roles(s) \Rightarrow |role\_set| < n.$$

从上节的陈述可以看出,NIST 的 RBAC 模型并未涉及到时间维的特性.

2 对角色授权约束及其时间特性的分析

在对 RBAC 引入时间特征后,特别是在对原本已起着极其重要作用的约束引入时间特征后,它就具有了进一步刻画现实系统中授权的能力.本节将对角色授权约束作相应的扩展,以适应引入时间特性的需要,同时简单讨论时间约束的语义问题.

2.1 RBAC中的授权约束

在 NIST RBAC 模型中,授权约束可分为静态授权约束(SSD)和动态授权约束(DSD)两类.静态授权约束决定用户不能被指定给一个冲突角色集合中的两个或多个角色.动态授权约束则限制用户不能同时激活一个冲突角色集合中的两个或多个角色.在上节中,我们已经给出了这两种约束的形式化描述.

2.2 对授权约束的非时间扩展

虽然 NIST 标准 RBAC 模型具有相当的对授权约束的描述能力.但其仅涉及多个角色被赋予用户之间的冲突限制关系.而现实世界中,我们也可能需要多个权限赋予角色之间的限制关系或多个用户被指定给角色之间的限制关系.在此基础上,可以使用一阶谓词逻辑来扩展 RBAC 中的授权约束.

定义 5(不确定函数 *OE* 与 *AO*).

OneElement:$OE(X)=x_i, x_i \in X$.

AllOther:$AO(X)=X-\{OE(X)\}$.

再引入如下 3 个集合:

$CR=\{cr_1, cr_2, \ldots, cr_s\}, cr_i \subseteq R$ 冲突角色组的集合.

$CP=\{cp_1, cp_2, \ldots, cp_t\}, cp_i \subseteq P$ 冲突权限组的集合.

$CU=\{cu_1,cu_2,\ldots,cu_u\}$,$cu_i\subseteq U$ 冲突用户组的集合.

通过使用 RBAC 系统中已定义(或后添加)的函数、关系以及∀,∃量词和相应算符,我们可以将原有的静态约束和动态约束改造成其相应的一阶谓词逻辑形式.举例如下:

约束要求:没有用户能被赋予两个冲突的角色.即同一个冲突组中的角色不能拥有同一个用户(SSD).

一阶谓词描述:$\forall u\in Users,\forall cr\in CR:|assigned\_roles(u)\cap cr|\leq 1$.

显而易见,扩展后的约束描述能力比原有的静态约束和动态约束有了极大的提高.详情参见文献[4].

2.3 引入时间的授权约束

为了对 RBAC 系统作时间维扩展,我们就必须考虑对授权约束作相应的时间扩展.下面首先对时间约束进行简单分类.

2.3.1 约束的时间特性分类

根据约束的时间特性,时间约束可分为激活时间范围约束、激活时间长度约束和时间范围内激活时间长度约束 3 类.

2.3.1.1 激活时间范围约束

该类约束规定用户、角色或者权限只能在特定时间范围内可以激活.如一个有工作时间限制的企业中,在非工作时间的范围内,将不允许某些操作的发生,或者不允许某些用户登录进系统.

2.3.1.2 激活时间长度约束

该类约束规定用户、角色或者访问许可每次只可以激活不超过一个固定长度的时间范围.可用该类约束限制某些很重要的操作或权限以防因激活时间过长而被盗用并产生严重后果的可能.

2.3.1.3 时间范围内激活时间长度限制

该类约束规定用户、角色或者访问许可在一定的时间范围内的累计激活时间不超过一个规定的上限.其对于用户的限定特别有用,可以控制用户的平均登录时间.

2.3.2 约束影响会话数量分类

根据约束影响的会话数量,时间约束又可分为仅影响一个会话的时间约束和影响多个会话的时间约束.

2.3.2.1 单会话时间约束

单会话时间约束是仅影响一个会话的时间约束.我们可能要求在某一个特殊的会话中,用户只能在指定的时间范围内激活相应的角色.

2.3.2.2 多会话时间约束

多会话时间约束是同时影响多个会话的时间约束.如我们可能要求在一个用户所有的会话中,某个角色激活的时间之和小于一个固定值.

以上两种分类方法是从不同的角度对会话相关的时间约束的考察,一个约束可能既是激活时间范围约束又是多会话时间约束.而对于这样的约束,我们显然可以将其分解成多个仅影响一个会话的激活时间范围约束.为了易于进行时间约束的形式化定义,我们规定激活时间范围约束和激活时间长度约束属于单会话时间约束,而时间范围内激活时间长度限制属于多会话时间约束.

2.4 时间约束的语义

对同一个时间约束,有两种不同的语义.第 1 种语义是该约束在约束的时间范围内总成立.这种语义非常自然.第 2 种语义则是该约束在约束的时间范围内可能成立但也可能不成立.这种语义初看似乎很不合理,但我们可以给出一个需要这种语义的例子:我们希望用户只能在工作时间中激活自己的角色.但用户在实际工作时间中可能选择不激活自己的角色.此时对于在工作时间里激活自己的角色这个约束来说,我们需要的就是第 2 种语义.

幸运的是,在形式化建模的过程中,可以不考虑第 2 种语义,我们可以通过逻辑运算,将所有第 2 种语义的时间约束改造成第 1 种语义的时间约束.限于篇幅,其具体过程从略.

3 对 RBAC 的时间扩展(TRBAC)

本节对 RBAC 做时间维上的扩展.首先我们通过定义一个离散时间点序列来模拟现实世界中的连续时间序列.在此基础上对上节的多种时间约束提出了形式化的定义.针对这些时间约束,我们对会话和全局系统状态空间也做了扩展,以实现文中提出的解决计算时间约束变化的 5 个算法.

3.1 时间系统定义

定义 6. 时间点序列.

$TE=\{t_i|i\in N\}$,TE 是虚拟时间世界所有时间点的集合.$t_i\in TE$ 代表虚拟时间世界中的一个时间点,它并不一定要与现实时间保持一致.

为讨论方便,定义一个函数映射 t_i 到真实的时间:$real\_time(t_i)$.

在定义了虚拟时间世界与现实时间的函数之后,我们可以定义虚拟时间上的<,+,−关系:

$\forall i,j\in N,\forall t_i,t_j\in TE,t_i<t_j\Leftrightarrow real\_time(t_i)<real\_time(t_j)$,

$\forall i,j,k\in N,\forall t_i,t_j,t_k\in TE,t_k=t_j+t_j\Leftrightarrow real\_time(t_k)=real\_time(t_i)+real\_time(t_j)$,

$\forall i,j,k\in N,\forall t_i,t_j,t_k\in TE,t_k=t_j-t_j\Leftrightarrow real\_time(t_k)=real\_time(t_i)-real\_time(t_j)$.

$T=seq(TE)$ 表示由时间点构成的虚拟时间世界中的时间序列.我们要求序列 T 中的元素是严格递增的.即 $\forall i,j\in N,\forall t_i,t_j\in T,i<j\Leftrightarrow t_i<t_j$.

不妨假定 $\Delta t=real\_time(t_i)-real\_time(t_{i+1})$ 是一个常数,这样,当时间序列 T,$real\_time$,Δt 确定后,虚拟时间序列就可确定地映射到现实世界时间.

定义 7. 时间区间定义.

$TR=\{(t_i,t_j)|t_i,t_j\in T,i<j\}$,时间区间是由两个时间点所构成的区间.

$TRS=2^{TR}$ 表示由时间区间构成的集合.

3.2 TRBAC的模型

TRBAC 继承了 RBAC 的所有元素,并做了相应的扩充.

3.2.1 TRBAC 约束的构成

在第 2 节,我们对 TRBAC 中的授权约束已做讨论,本节将对时间约束将给出一个形式化的描述.我们用 C 表示所有在第 2.2 节中用经过一阶谓词逻辑扩展后所描述的约束集合.T_C 表示所有 TRBAC 中的时间约束.S_S_C 表示 T_C 中所有作用于单会话上的时间约束集合.M_S_C 表示 T_C 中所有作用于多会话上的时间约束集合.

定义 8. 时间约束谓词定义.

$In\_Range\subseteq C\times TRS$ 表示约束 C 在指定的时间区间集合 TRS 内必须成立.

$Last\_Length\subseteq C\times N$ $Last_Length(c,i)$表示约束 c 从激活的时间 t 开始,最多只能激活到时间 t_0,$real\_time(t_0)-real\_time(t)=i$.

$TR\_Last\_Length\subseteq C\times TRS\times 2^N$ $TR_Last_Length(c,\{tr_1,tr_2\},\{i_1,i_2\})$表示约束 c 在指定的时间范围 tr_1,tr_2 内分别只能激活指定的时间长度 i_1,i_2.激活时间长度的定义同 $Last\_Length$.

定义 9. 时间约束(T_C)定义.

$T\_C=S\_S\_C|M\_S\_C$,

$S\_S\_C=In\_Range(C,TRS)|Last\_Length(C,i)$,

$M\_S\_C=TR\_Last\_Length(C,TRS,2^N)$.

3.2.2 TRBAC 的形式化

对于引入时间后的 RBAC 系统,会话在对系统的时间特性的研究中占有重要的地位,整个模型的时间特性集中地体现在会话的时间特性上,所以我们必须也对会话作出相应的扩展.

定义 10. session 的扩展.

session 由以下 7 元组来描述,⟨*users*,*roles*,*ua*,*pa*,*s_s_c*,*ssc_starttime*,*state_change_time*,*task*⟩.其中 *users*,*roles*,*ua*,*pa* 分别是全局系统中相应集合的子集,表示该会话所允许的用户、角色、用户角色的映射关系和角色权限映射关系.*s_s_c* 表示该会话在运行时必须满足的时间约束,此类时间约束仅作用于该会话本身.*ssc_starttime*⊆*S_S_C*×*TE* 用来记录每一个 *S_S_C* 时间约束开始激活的时间.当计算时间约束的激活长度时,我们将从 *ssc_starttime* 中查找相应信息.*state_change_time* 记录该会话的状态变化时间,将在下节作详细的描述.*task* 表示会话需完成的一系列操作.

定义 11. 函数的扩展.

$currenttime:\varnothing\to TE$:是一个全局原子函数,返回当前时间.

$start:TR\to TE:start((t_i,t_j))=t_i$,

$end:TR\to TE:end((t_i,t_j))=t_j$,

$in\_range:TE\times TR\to\{true,false\}:in\_range(t,(t_i,t_j))=(t\ge t_i)\wedge(t\le t_j)$,

$not\_in\_range:TE\times TR\to\{true,false\}:not\_in\_range(t,(t_i,t_j))=(t<t_i)\vee(t>t_j)$,

$in\_range:TE\times TRS\to\{true,false\}:in\_range(t,tr_i)=\bigcap_{tr\in tr_i}in\_range(t,tr)$.

在 TRBAC 中,函数可随用户的需要自行扩充,如定义:

$active\_length:Session\to TE$: 记录会话的激活时间.

定义 12. 扩展后的系统状态空间.

OLDS=$\{s_i|i\in N\}$表示系统中已结束会话的集合.

BLOCKS=$\{s_i|i\in N\}$表示系统中未结束但由于时间约束无法进行而阻塞的会话集合.

CURRENTS=$\{s_i|i\in N\}$表示系统中当前正在进行的会话集合.

ERRORS=$\{s_i|i\in N\}$表示系统中由于时间约束或其他原因而无法进行下去的会话集合.

这 4 个集合两两相交为空.

约定会话从激活的时候开始,一直到结束,除 *ssc_starttime* 外其所包含的属性不变.如果一个用户想改变自己某个会话属性,则可以在结束这个会话的同时,开始另外一个新的会话,新的会话继承原会话不变的属性并加入用户改变的新属性,代替原会话运行下去.原会话将作为结束的会话被放到 OLDS 里去.

我们用 *state* 表示一个系统状态,*states* 表示系统的所有状态空间:

$state=\{U,R,P,UA,PA,SA,RH,CURRENTS,BLOCKS,OLDS,ERRORS,M\_S\_C,MSC\_SESSIONS,currentime\}$,

$state=2^{state}$.

其中 *M_S_C* 表示所有作用于多会话的时间约束.*MSC_SESSIONS*⊆*M_S_C*×*SESSIONS*×2^{TE},是由 *M_S_C* 与该 *M_S_C* 相关的会话集合及会话集合中会话激活相应时间约束的起始时间构成的三元组.三元组的第 2 个元素和第 3 个元素都是集合,这两个集合的势应相等.

3.3 引入时间后的约束描述能力

显而易见的是,引入时间后,约束描述能力比以前有了显著的提高.举例描述如下:

3.3.1 激活时间约束

对于用户 u_1 的约束:

局部定义: $ALLOWTR_{u_1}\subseteq TR$.

表达式: $user\_sessions(u_1)\ne\varnothing\Rightarrow in\_range(currenttime,ALLOWTR_{u_1})$.

对于角色 r_1 的约束:

局部定义: $ALLOWTR_{r_1}\subseteq TR$.

表达式: $r_1\in session\_roles(OE(Sessions))\Rightarrow in\_range(currenttime,ALLOWTR_{r_1})$.

对于访问许可 p_1 的约束

局部定义: $ALLOWTR_{p_1}\subseteq TR$.

表达式: $p_1 \in session\_perms(OE(Sessions)) \Rightarrow in\_range(currenttime, ALLOWTR_{p_1})$.

3.3.2 激活时间长度约束

对于用户 u_1 的约束:

局部定义: $ACTIVELEN_{u_1} > 0$.

表达式: $s \in user\_sessions(u_1) \Rightarrow active\_length(s) \le ACTIVELEN_{u_1}$.

对于角色 r_1 的约束:

局部定义: $ACTIVELEN_{r_1} > 0$.

表达式: $r_1 \in session\_roles(OE(Sessions)) \Rightarrow active\_length(OE(Sesssions)) \le ACTIVELEN_{r_1})$.

对于访问许可 p_1 的约束:

局部定义: $ACTIVELEN_{p_1} > 0$.

表达式: $p_1 \in session\_perms(OE(Sessions)) \Rightarrow active\_length(OE(Sessions)) \le ACTIVELEN_{p_1}$.

因而引入时间后的系统提供了更具体、更全面的安全描述能力.

4 TRBAC 状态改变及其一致性讨论

4.1 保持时间约束的算法

TRBAC 状态改变包括两类变化,第 1 是与时间变化无关,普通 RBAC 的状态转变,该类状态转变及其一致性讨论已有很多研究.第 2 是随时间约束变化而引起的系统状态的变化.本文主要讨论第 2 类变化.

当 *currenttime* 每次变化时,系统都应该判断一下状态的合法性.典型的想法是当 *currenttime* 每次变化时判断时间约束是否成立.这种方法有明显的缺点.当时间约束规则很多时,效率难以得到保证.我们估算一下当 *currenttime* 变化时,这种方法所需作的计算量的规模.当 *currenttime* 发生变化,我们必须对所有 $CURRENTS \cup BLOCKS$ 中的会话计算约束,即使只有 *M_S_C* 约束,也必须在每次时间变化的同时进行 $|CURRENTS \cup BLOCKS| \times |M\_S\_C|$ 次时间约束的计算.假定时间变化是秒级,再考虑到进行一次时间约束的计算,包括多次对用户、会话、角色和权限的计算,随着相关集合中元素的增多,其计算量的增加是非常可观的,甚至可以达到不可接受的地步.为了解决这个问题,我们考虑对于每个时间约束,计算它的状态变化时间,即该时间约束由成立到不成立的时间及由不成立到成立的时间,然后,系统可在相应时间里,把相关会话放入相应的会话集合中作进一步处理.在本节中,共给出了 3 个算法,分别计算 3 类时间约束的状态改变时间.

定义 13. 状态变化时间函数.

$state\_change\_time:((CURRENTS \cup BLOCKS) \times S\_S\_C) \rightarrow TE$.

$state\_change\_time:(states \times M\_S\_C \times \{true,false\}) \rightarrow TE$.

state_change_time 函数的第 1 种形式用来计算单会话约束的翻转时间.第 2 种形式用来计算多会话约束的翻转时间.

下面给出算法来计算 3 种时间约束的状态变化时间.本文中所有算法包含如下假定:

(1) 时间约束的定义和形式同上节定义,并且所有的时间区间集合中包含的时间区间不交.

(2) 当计算时间约束由成立到不成立的状态变化时间时,时间约束必须成立,否则时间约束不成立.

(3) 算法输入参数中参数为 true 表示计算约束成立到不成立的状态变化时间,否则反之.

算法 1. 计算激活时间区间约束的状态变化时间.

输入:会话 *s*,激活时间区间约束.

输出:状态变化时间.

计算过程:

if $\exists tr \in trs, in\_range(currenttime, tr)$

$sct = end(tr)$

else

在 *trs* 中找到一个时间区间 *tr* 满足

$currentime < start(tr) \cap \forall tr_i \in trs, (start(tr) - currentime) \le (start(tr_i) - currenttime)$

$sct=start(tr)$

sct 即为所求的状态变化时间.

算法 2. 计算激活时间长度约束的状态变化时间.

输入:会话 *s*,激活时间长度约束 *ssc*.

输出:状态变化时间.

计算过程:

if $\exists t \in TE, (ssc, t) \in s.sst\_starttime$

$sct=t+i$

else

$sct=\infty$

sct 即为所求的状态变化时间.

在描述算法 3 之前,为了描述方便,我们先引入一对虚拟辅助函数,get_index 得到一个指定的元素在元素集合中的序数,get_element 得到在一个元素集合中,指定序数的元素.我们略去这两个函数的形式化定义.

算法 3a. 计算时间范围内激活时间长度约束的状态变化时间.

输入:时间范围内激活时间长度约束 *msc*,{true,false}.

输出:状态变化时间.

计算过程:

IF $\exists sessions \in Sessions, \exists starttimes \in 2^{TE}, (msc, sessions, starttimes) \in MSC\_SESSIONS$

if 第 2 个参数为 true //计算约束成立到不成立的状态变化时间

//计算到当前时间点,所有的会话已用的时间总和

$$alltime = \sum_{s \in sessions} (currenttime - get\_element(starttimes, get\_index(sessions, s))).$$

//将所有剩余时间平均分配给所有会话.

在 *trs* 中找到相应的 *tr*,满足 $in\_range(currenttime, tr)$

$$sct = currenttime + (get\_element(\{i_1, ...\}, get\_index(tr, trs)) - alltime) / |sessions|. \quad (1)$$

else

在时间范围集合 *trs* 中找到如下一个时间区间 *tr* 满足:

$currenttime<start(tr)\cap\forall tr_i\in trs,(start(tr)-currenttime)\le(start(tr_i)-currenttime)$ $sct=start(tr)$. (2)

else

$sct=\infty$//未找到指定约束. (3)

sct 即为所求的状态变化时间.

算法 3a 将所有剩余时间平均分配给所有与该时间约束有关的会话,这一假定可能与现实中的情况不符,通过引入如下一个虚拟函数,我们可以在算法 3b 中解决这个问题.

$level\_percent: sessions \times Sessions \to N$,该虚拟函数计算一个会话在指定的会话集合中所占的加权百分比.可能的实现包括计算各会话的历史记录、优先级等等,其具体实现依赖用户.

算法 3b. 计算时间范围内激活时间长度约束的各会话的状态变化时间集合.

输入:算法假定同算法 3a.

输出:状态变化时间集合.

计算过程:

将算法 3a 中式(1)变为:

按序取出 *sessions* 中每一个会话 *s*,进行如下计算:

$$lifetime=(get\_element(\{i_1,...\},get\_index(tr,trs))-alltime)\times level\_percent(s,sessions)/100$$

$$sct=currenttime+lefetime$$

将 *sct* 放入状态变化时间集合中.

将式(2)和式(3)处的赋值改为将值多次赋入集合.

读者可能注意到,在算法 3,尤其是算法 3b 中,我们没有检查该类约束的状态变化时间是否越过了时间区间的边界,这是由于我们考虑到计算约束的状态变化时间是从属于计算会话的状态变化时间的,所以我们将这样的检查放在算法 4 中.

4.2 会话的状态变化

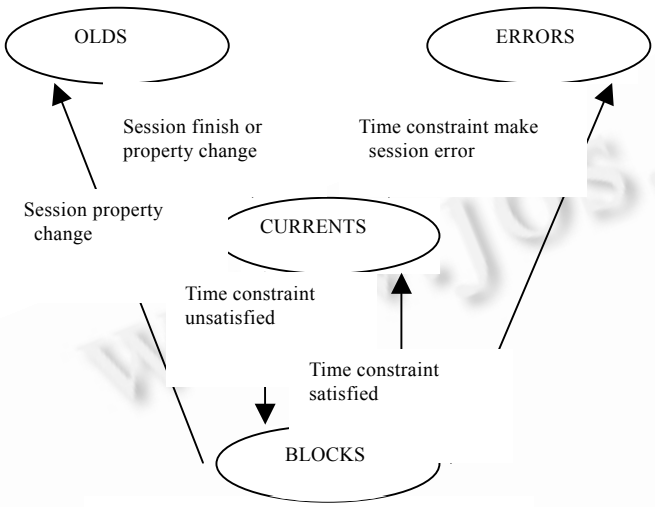

Fig.1 Session state change pattern

图 1 会话状态变化图

在系统正常状态下,会话将在 *ERRORS*, *OLDS*,*CURRENTS*,*BLOCKS* 这 4 个会话集合中变化.当系统创建一个会话时,将根据会话的时间约束是否满足而将该会话放入 *CURRENTS* 或 *BLOCKS* 中.如果会话属性不发生变化,会话可能在 *CURRENTS* 或 *BLOCKS* 两个集合中迁移零次或多次,并最终将由 *CURRENTS* 中正常结束而被移动到 *OLDS* 中,或由于无法继续运行而被移动到 *ERRORS* 中.而如果会话属性发生变化,会话也可能从 *BLOCKS* 中直接迁移到 *ERRORS* 或 *OLDS* 中.图 1 显示了会话的状态变化及引起状态变化的时间约束类型.

上面已讨论了对 3 种类型的时间约束如何计算其状态变化时间,那么如何判断何时会话的状态将发生变化呢?算法 4 解决了这个问题.

算法 4. 计算会话的状态变化时间.

输入:会话 *s*,{true,false}.

输出:会话的状态变化时间.

计算过程:

sct=∞ //无任何时间约束起作用

if 第 2 个参数为 true //计算约束成立到不成立的状态变化时间

  if $s.ssc\_starttime\neq\varnothing$ //存在已经被激活的单会话时间约束

    对 *s.ssc_starttime* 中每一个激活的时间约束,调用算法 1、算法 2 计算相应的状态变化时间

    *temp*1=上述状态变化时间的最小值

  //计算多会话约束的状态变化时间

  if $\exists msc,sessions,time,(msc,sessions,times)\in MSC\_SESSIONS\cap s\in sessions$

    对于每一个与 *s* 有关的多会话时间约束调用算法 3 计算相应的状态变化时间

    *temp*2=上述状态变化时间的最小值

    对于每一个与 *s* 有关的多会话时间约束,找出 *currenttime* 所属时间区间的结束时间

    *temp*3=上述结束时间的最小值

  sct=min(*temp*1,*temp*2,*temp*3)

else //计算约束不成立到成立的状态变化时间

  if $s.ssc\_starttime\neq\varnothing$ //存在已经被激活的单会话时间约束

对 *s.ssc_starttime* 中每一个激活的时间约束,调用算法 1、算法 2 计算相应的状态变化时间

*temp*1=上述状态变化时间的最大值

//计算多会话约束的状态变化时间

if $\exists msc, sessions, times, (msc, sessions, times) \in MSC\_SESSIONS \cap s \in sessions$

对于每一个与 *s* 有关的多会话时间约束调用算法 3 计算相应的状态变化时间

*temp*2=上述状态变化时间的最大值

sct=max(*temp*1,*temp*2)

sct 即为所求得的会话状态变化时间.在计算出会话的状态变化时间之后,可将其保存在会话中的 *state_change_time* 中.

4.3 系统一致性状态及其维护

定义 14. 系统一致性状态是指满足所有时间约束及一般约束的系统状态.

我们必须维护系统一致性状态.本节将从两方面讨论这个问题.首先,在前述 4 个算法的基础上,我们进一步分析何时计算各类约束及会话的状态转变时间,并在此基础上,提出系统会话调度算法.其次,我们讨论两种状态维护策略来弥补由于使用离散的递增时间点序列来模拟连续的时间空间而可能带来的误差.

4.3.1 状态转变时间的计算时机和系统会话调度算法

对单会话时间约束来说,由于其仅涉及一个会话,其计算时机较为简单.其第 1 次计算应在该时间约束被激活时进行,以后在会话改变状态时进行计算即可.对于多会话时间约束而言,由于其约束时间长度涉及所有与该约束相关的会话,故除了单会话时间约束的情况以外,当一个新会话激活该约束、一个与该约束相关的会话暂停或终止运行时,都必须重新计算.表 1 列出了 3 类时间约束的相应计算时机.

Table 1 The state change calculating time for 3 kinds of time constraint

表 1 3 类时间约束状态改变时间的计算时机

	Satisfy to not satisfy	Not satisfy to satisfy
Time range constraint	(1) The first time a session active the constraint (2) The session is moved from *blocks* to *currents*	The session is moved from *currents* to *blocks*
Active length constraint	The first time a session active the constraint	
Active length constraint in time range	(1) The first time a session active the constraint (2) The session is moved from *blocks* to *currents* (3) Other sessions first active this constraint	(1) The session is moved from *currents* to *blocks* (2) A session concerning this constraint is stopped or blocked

由表 1 易知,会话由运行到阻塞的状态改变时间的计算时机为:

(1) 会话自身激活某时间约束时;

(2) 当会话从 *blocks* 移动到 *currents* 中时;

(3) 当有其他会话激活了与该会话相关的一个多会话约束时.

而会话由阻塞到运行的状态改变时间的计算时机为:

(1) 当会话从 *currents* 移动到 *blocks* 中时;

(2) 当有一个与该会话共同受一个多会话约束限制的其他会话中止或暂停时.

算法 5. 系统会话调度算法.

计算过程:

while(true){

if 一个会话正常结束,then 将该会话由 *currents* 中移动到 *olds* 中

if 一个会话由于 2 类约束或 1、3 类约束由于时间区间用完而无法运行下去,then

将该会话由 *currents* 移动到 *errors* 中

if 一个会话首次激活一个 *S_S_C* then

计算该会话的状态改变时间并保存在该会话的 *state_change_time* 中

if 一个会话激活了一个 *M_S_C* 或一个受某 *M_S_C* 约束的会话中止或暂停时 then

对所有受该 *M_S_C* 约束的会话,重新计算并保存这些会话的状态改变时间

if 一个 *currents* 中的会话的 *state_change_time* 到达,then

将该会话从 *currents* 移动到 *blocks* 中,并计算、保存该会话的状态改变时间

if 一个 *blocks* 中的会话的 *state_change_time* 到达 then

将该会话从 *blocks* 移动到 *currents* 中,并计算、保存该会话的状态改变时间

}

4.3.2 状态恢复策略

在引入时间特性之前,RBAC 可在进行某个操作之前进行判断,当一致性不被满足时,操作就不被允许.在TRBAC中,是 *currenttime* 的递增操作引起了约束规则的状态变化.当系统发现 *currenttime* 越过了某条时间约束的状态变化时间,即某条时间约束不满足时,系统必须进行状态恢复.

最简单的状态恢复是将引起时间约束不满足的相关会话中止或暂停,但如果恢复的粒度变细,如要求对会话在时间约束不满足的时间里的相应操作完全恢复,或考虑到未来的研究中可能必须引入会话的优先级等其他因素,进而必须考虑如何选择相关的会话中止或暂停,我们仍有许多工作要做.

4.3.3 确保安全状态策略

确保安全状态则不允许系统进入不安全状态.其基本思路为,找到状态变化时间 *sct* 之前的某个时间 *t*,满足 $t \le sct \cap (sct - t) \le \Delta t$,通过适当地选择一个很小的 Δt,我们可以确保当下次计算 *currenttime* 时,必然有 $currenttime > sct$,则在时间 *t* 我们预先将可能引起系统进入不安全状态的会话中止.相对而言确保安全状态实现简单.但无论怎样选择 Δt,都存在将原本可以完成的会话误中止或暂停的可能性.

5 结 论

TRBAC 在 RBAC 基础上作了时间特性方面的扩展.引入时间后的系统有着更全面、更具体的安全属性描述能力.TRBAC 定义了系统时钟,对约束、会话和系统状态空间本身进行了时间扩充.解决了时间授权约束和会话的状态转变问题,同时对系统一致性状态的维护进行了讨论与分析.但在一致性状态维护方面仍有很多工作需要进一步地研究.

References:

[1] Ferraiolo DF, Sandhu R, Gavrila S. Proposed NIST standard for role-based access control. ACM Transactions on Information and System Security, 2001,4(3):224~274.

[2] Osborn S, Sandhu R. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Transactions on Information and System Security, 2000,3(2):85~106.

[3] AHN G-J, Sandhu R. Role-Based authorization constraints specification. ACM Transactions on Information and System Security, 2000,3(4):207~226.

[4] Dong GY, Qing SH, Liu KL. Role-Based authorization constraint with time character. Journal of Software, 2002,13(8):1521~1527 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/1521.pdf.

[5] Sandhu R. Issues in RBAC. In: Youman C, Sandhu R, Coyne E, eds. Proceedings of the 1st ACM Workshop on Role-based access control. ACM Press, 1996. 21~24.

[6] Ferraiolo D, Kuhn R. Role based access control. In: Proceedings of the 15th National Computer Security Conference, National Institute of Standards and National Computer Security Center. 1992. 641~650.

附中文参考文献:

[4] 董光宇,卿斯汉,刘克龙.带时间特性的角色授权约束.软件学报,2002,13(8):1521~1527. http://www.jos.org.cn/1000-9825/13/1521.pdf.