安全操作系统授权撤销机制的研究

吴新勇　熊光泽
（电子科技大学计算机科学与工程学院　成都610054）

Research of Revoking Mechanism of Authorization in Secure Operating System

WU Xin-Yong　XIONG Guang-Ze
(Computer Science and Engineering College, UEST of China, Chengdu 610054 China)

Abstract　Revoking operation is a very important component of access control. The lack of effective revoking operation impinges on supporting dynamic security policies in secure operation system. Analyzing authorization system, this paper presents a revoke policy which supports cascade and noncascade revocation. The policy adopts Hash authorization list and critical-based callback function to implement revocation of point to point and point to plane. Our experiments in security kernel show the mechanism is feasible, which provides the basis of further researching dynamic security policies in secure operation system.

Keywords　Secure operating system, Access control, Authorization system, Revocation of permissions

1　引言

为保障系统的安全性，现代信息系统都采用了基于访问控制的安全机制。访问控制包含身份认证、授权和审计等功能，其基本思想一般是采用引用监视器来拦截用户进程的系统调用操作，通过安全策略匹配检查，授予或拒绝用户访问敏感信息的权限。系统安全功能的发展要求能够支持动态的安全策略，比如策略的动态加载，这就对权限的撤销提出了要求，一个良好的权限撤销机制能够保证系统的连续性和一致性。

权限撤销操作的实现依赖于一种授权选项方式，选择了相应授权方式的用户才可将权限传递给其他用户（任务或进程），授权选项的基本原则是有授予权力的用户必须能够撤销其发布出去的权限，并且这种撤销方式必须具有递归的能力，比如能依次撤销第2、第3、…、第 n 级获得者的权限。权限撤销操作在数据库系统中研究较多，也提出了不少撤销模型[1~3]，但在安全操作系统领域，合理的、实用性强的撤销模型太少。多数安全系统都采用了粗暴终止所有关联进程的方式来解决权限的撤销，往往给做重要操作的用户带来不可预料的后果（比如关键数据的丢失等）。以往的安全体系结构，如DTOS[4]、FLASK[5]等，也只给出了简单的撤销机制，无法解决迁移权限及递归权限的撤销等关键性的问题，FLASK 的成功实现者 Selinux[6]中也没有处理权限撤销的机制，而没有这种机制限制了这些系统支持动态策略的能力。权限撤销的操作不影响用户任务的最好方法是尽量使任务的关键操作在安全策略修改前完成，或者保存关键的数据，从而不会给用户带来损失。基于我们在安全核[7]方面的研究成果，本文提出了一种合理实用的权限撤销技术。

2　系统模型定义

在计算机系统里，所有的安全策略都可以看作一个访问控制矩阵或其中一部分，矩阵的每一行是一系列的〈客体，权限集〉二元组，称为执行环境；每一列是一系列的〈主体，权限集〉二元组，称为 ACL（Access Control List）。一般的访问控制授权表示为〈主体，权限集，客体〉，这种表达属性太少，不能表达级联授权，也不能满足较复杂的权限撤销操作的需要。在数据库访问控制中，引入了正负权限和时间信息的技术，正负权限用于约束主体行使和发布该权限，时间信息用于限制权限的使用期限。数据库系统相对操作系统而言比较小且管理集中，而操作系统对象多、层次复杂，引入正负权限使得授权管理复杂，系统开销超出所能接受的指标，所以一般情况下，安全操作系统没有引入正负权限的机制。我们用相对简单而实用的权限递归选项来替代正负权限的功能，通过递归次数来约束主体发布权限，时间特性也作为授权选项出现在授权机制中。在我们的安全核系统中，授权操作定义如下：

定义1　授权系统具有五大属性集——S 表示主体集；P 表示权限集；O 表示客体集；$GTIME$ 表示时间特性选项，表征权限被授予的时间；$GNUM$ 为自然数集，表示授权递归选项。授权操作 g（grant）用一个6元组〈s_{to}，p，o，s_{from}，gt，gn〉表示，其中：

s_{to}：权限被授予的主体，$s_{to}\in S$；p：被授予的权限，$p\in P$；o：与权限相关的客体，$o\in O$；s_{from}：授权者，$s_{from}\in S$；gt：用日历时间表示的授权时间，$gt\in GTIME$，为简化，我们用时间基数的倍数表示，倍数越大，授权时间越晚；gn：递归授权次数，$gn\in GNUM$，只有当 gn＞0时，权限被授予的主体才可向下发布权限，每次发布后 gn 减1，直至为0。

针对授权属性，定义了相应的操作。

定义2（授权属性操作）　对于给定的授权操作 g 定义了6个操作来获取授权元素值：subject(g)＝g 的被授权主体；grantor(g)＝ 授权者；perm(g)＝授予的权限；object(g)＝与权限关联的客体；gtime(g)＝授权的时间；gnum(g)＝权限的可递归次数。

安全系统由不同的授权状态 Δ 组成，授权操作引起该状态的迁移。我们以操作系统为例说明权限授予的过程。其中主体为系统内核 S_{sys}和7个用户进程 S_i（i＝1，…，7），客体为消息队列 msgq，权限集为读写，表示为 r&w。授权状态 Δ_i表示为：

$\Delta_i=\{g_1,g_2,g_3,g_4,g_5,g_6,g_7,g_8\}$

$g_i=\langle S_1,r\&w,msgq,S_{sys},t,4,\rangle$

吴新勇　博士研究生，主要研究方向：嵌入式实时操作系统，系统安全及可靠性。熊光泽　教授，博士生导师，主要研究方向：实时计算机系统及软件开发支持。

g_2= ⟨S_2,r&w,msgq, S_1,2t,3⟩
g_3= ⟨S_3,r&w,msgq, S_1,3t,3⟩
g_4= ⟨S_4,r&w,msgq, S_2,4t,2⟩
g_5= ⟨S_5,r&w,msgq, S_3,5t,2⟩
g_6= ⟨S_5,r&w,msgq, S_4,6t,1⟩
g_7= ⟨S_7,r&w,msgq, S_5,6t,1⟩
g_8= ⟨S_6,r&w,msgq, S_4,8t,1⟩

系统为用户进程S_1创建了消息队列msgq,并赋予读写的权限,权限的递归发布次数为4,对消息队列的读写权限再通过进程S_1分别依次传递给S_2、S_3,再由S_2和S_3进程向其他进程发布,递归次数随之递减。该授权过程用标记图表示为图1。

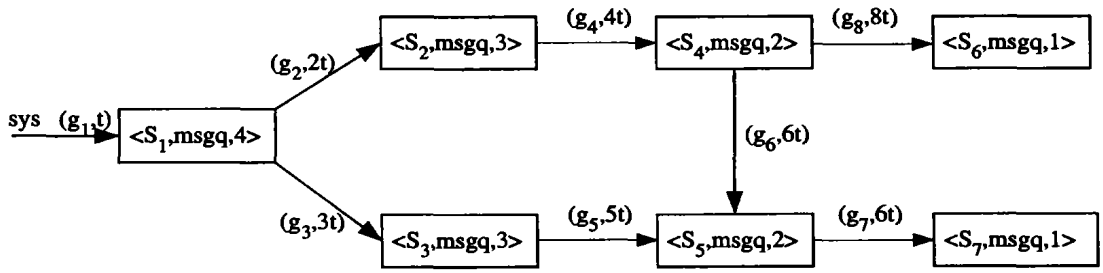

图1 样例的系统授权图

定义3(级联授权) 给定两个授权操作g_i、$g_j \in \Delta$,如果满足:

(subject(g_i)=grantor(g_j))and(perm(g_i)=perm(g_j))
and(object(g_i)=object(g_j))and
(gnum(g_i)>gnum(g_j))and(gtime(g_i)<gtime(g_j))
("and"表示逻辑"与"操作)

则我们说g_i和g_j是级联授权,表示为$g_i \rightarrow g_j$

定义4(授权链C(Δ)) 在授权状态Δ中,所有的级联授权序列组成了授权链。

$C(\Delta)=\bigcup_{1\leqslant i\leqslant n}\{\{g_1,g_2,\cdots,g_i\}\}$,其中$g_i \in \Delta$;$g_1 \rightarrow g_2 \rightarrow g_3 \rightarrow \cdots \rightarrow g_i$为级联授权序列。

根据定义4,上例授权状态Δ_1的授权链为:

$C(\Delta_1)=\{\{g_1\},\{g_1,g_2\},\{g_1,g_2,g_4\},\{g_1,g_2,g_4,g_6\},\{g_1,g_2,g_4,g_8\},\{g_1,g_3\},\{g_1,g_3,g_5\},\{g_1,g_3,g_5,g_7\}\}$

以上的四个基本定义给出了我们设计的安全核中级联授权系统的属性及状态模型,有助于我们了解和深入分析与之相反的权限撤销的过程和相应的状态转换模型,从而保持系统的连续性和一致性。下一节的权限撤销策略将基于这些基本定义来定义和描述。

3 权限撤销策略

在安全操作系统领域,权限撤销策略研究较少,主要是因为撤销的复杂性和由此带来的高昂的系统开销。在FLASK体系结构中,仅讨论了权限的全局撤销,即客体的拥有主体只能提出全局撤销请求,这时安全系统将撤销所有主体对该客体的相应访问权限,撤销策略没有选择性,更不能实现级联撤销。另外一种基于类型的撤销机制[8]提出绑定撤销属性到对象引用中,主体在访问对象时进行撤销检查,使授权系统实现运行时的动态撤销检查。该撤销模型是语言相关的,实现了点对点的权限撤销,但其撤销机制依赖于面向对象的封装特性及特殊的编译器,并且没有解决级联撤销的问题。正如前面所述,权限的递归发布不可避免,也是动态权限撤销要解决的关键问题之一。根据授权系统的特性,我们提出了支持级联和非级联的权限撤销模型。

3.1 级联权限撤销

级联撤销要求主体A撤销发布给主体B的权限时也要撤销B后续发布出去的该权限,即删除该授权链,但是又不能破坏别的主体发布给B的该权限的授权链。下面我们用形式化的方式建立级联撤销模型。

级联撤销包括了几步操作,先撤销B的该权限,再递归撤销B发布出去的该权限,最后删除该授权链。级联撤销的形式化定义如下:

定义撤销操作C_REVOKE(Δ,revoke_request),Δ表示当前的授权状态;撤销请求revoke_request用一个四元组⟨B,p,o,A⟩表示,意思为"A撤销B对客体o的权限p";Δ′表示撤销操作后的授权状态;其中主体A、B∈S,客体o∈O,权限集p∈P。则有:

$\Delta'=C\_REVOKE(\Delta,\langle B,p,o,A\rangle)=\Delta-G_{base}-G_{recursive}$,

其中

$G_{base}=\{g \mid g\in\Delta,\text{subject}(g)=B,\text{grantor}(g)=A,\text{perm}(g)=p,\text{object}(g)=o\}$

$G_{recursive}=\{g \mid g\in\Delta,\forall\{g_1,\cdots,g_i,g\}\in C(\Delta),\exists\, g_i\in G_{base},1\leqslant i\leqslant n\}$

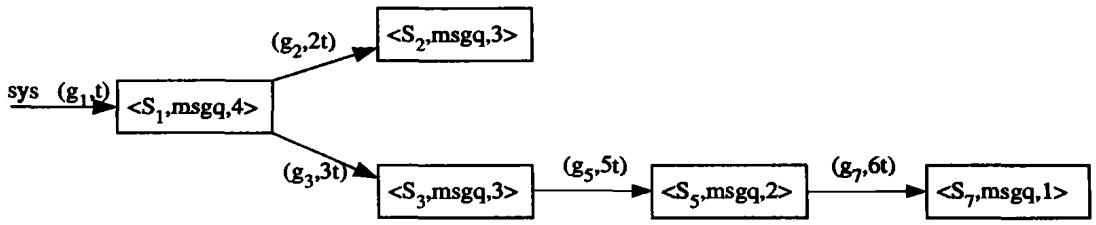

图2 S_4被级联撤销权限后的授权状态图

G_{base}表示基本的授权,即A对B的直接授权,$G_{recursive}$表示B后续发布的递归授权。

以图1为例,如果S_2请求安全核级联撤销S_4对消息队列的读写权限,则按撤销定义有:

$G_{base}=\{g_4\}$;$G_{recursive}=\{g_6,g_8\}$;$\Delta_1=\{g_1,g_2,g_3,g_4,g_5,g_6,g_7,g_8\}$

$\Delta_1'=C\_REVOKE(\Delta_1,\langle S_4,r\&w,msgq,S_2\rangle)=\Delta_1-G_{base}-$

$G_{recursive}=\{g_1,g_2,g_3,g_5,g_7\}$

$C\{\Delta_1'\}=\{\{g_1\},\{g_1,g_2\},\{g_1,g_3\},\{g_1,g_3,g_5\},\{g_1,g_3,g_5,g_7\}\}$

撤销后的授权状态标记图如图2所示。

级联撤销策略能够保证安全核实时跟踪权限的发布，通过授权的时间标记和递归次数来区分相同权限的不同来源，避免了权限的混淆和误撤销，适用于主体被删除、对象被删除和安全策略动态更新时进行点到面的权限撤销。

3.2 非级联权限撤销

如上节所说，级联撤销仅适用于点到面的权限撤销，如果用户只想撤销个别违反安全策略的进程的访问权限，而又想保留授权链上其他可信进程的访问权限时，级联撤销就不适用了。为了满足用户灵活的权限撤销需求，我们定义了非级联权限撤销策略来弥补级联撤销的不足。非级联撤销要求主体A在撤销B的权限时，只删除对B的授权，不得取消授权链上其它的操作，所以A将接管后续授权链，操作相对更复杂。非级联撤销的形式化定义如下：

定义撤销操作 $REVOKE(\Delta,\text{revoke\_request})$，授权策略集 Ω 定义合法的授权规则，撤销请求为 $\langle B,p,o,A\rangle$，其中主体 $A,B\in S$，客体 $o\in O$，权限集 $p\in P$。给定初始授权状态 Δ，则有：

$\Delta'=REVOKE(\Delta,\langle B,p,o,A\rangle)=C\_REVOKE(\Delta\cup G_{receive},\langle B,p,o,A\rangle)$，

其中

$G_{base}=\{g\mid g\in\Delta,\text{subject}(g)=B,\text{grantor}(g)=A,\text{perm}(g)=p,\text{object}(g)=o\}$

$G_{receive}=\bigcup\{\{g\mid g=\langle\text{subject}(g_i),\text{perm}(g_i),\text{object}(g_i),A,\text{gtime}(g_i),\text{gnum}(g_i)\rangle,g\in\Omega,\exists\ g_i\in\Delta,\exists\ g_i\in G_{base},\text{subject}(g_i)\neq B,\text{subject}(g_i)\neq A,\ g_i\rightarrow g_j,1\leqslant i,j\leqslant n\}\}$

G_{base} 表示基本的授权，即A对B的直接授权；$G_{receive}$ 表示A接管B后续的递归授权。撤销操作先由撤销者接管被撤销主体后续的授权，再用级联授权撤销无用的授权链。

以图1为例，S_2 请求安全核仅撤销 S_4 对消息队列的读写权限，则按撤销定义有：

$G_{base}=\{g_4\}$；$G_{receive}=\{g_6',g_8'\}$；$g_6'=\langle S_5,r\&w,msgq,S_2,6t,1\rangle$；$g_8'=\langle S_6,r\&w,msgq,S_2,8t,1\rangle$

if $g_6'\in\Omega$ and $g_8'\in\Omega$ then

$\Delta_1'=C\_REVOKE(\Delta_1\cup G_{receive},\langle S_4,r\&w,msgq,S_2\rangle)$

$=\Delta_1\cup G_{receiv}-G_{base}-G_{recursive}$

$=\{g_1,g_2,g_3,g_5,g_6',g_7,g_8'\}$

$C(\Delta_1')=\{\{g_1\},\{g_1,g_2\},\{g_1,g_2,g_6'\},\{g_1,g_2,g_8'\},\{g_1,g_3\},\{g_1,g_3,g_5\},\{g_1,g_3,g_5,g_7\}\}$

撤销后的授权状态标记图见图3所示。

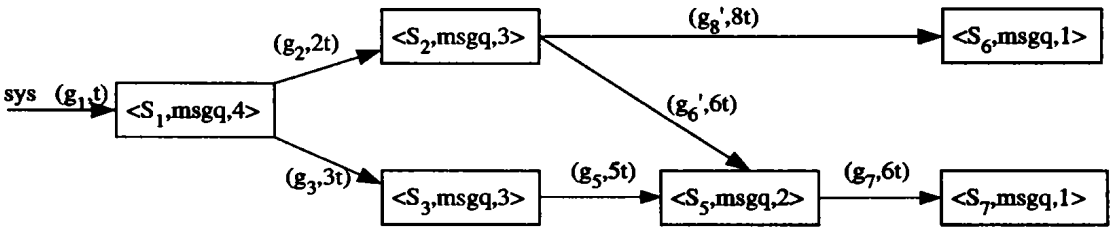

图3 S_4 被非级联撤销权限后的授权状态图

非级联撤销时，会遇到撤销者接收了本来不是它亲自给予的授权的情况，我们引入了授权策略集 Ω 来解决这一问题，如果该授权符合授权策略，则接收合法，由撤销者行使后续授权者的权力，否则抛弃后续授权。

级联和非级联授权是互补的策略，两者结合可以解决点到点和点到面的撤销操作，用户通过撤销选项来确定他们的撤销操作类型，该综合策略描述简洁，可实现性强，能够满足安全操作系统可能有的绝大多数权限撤销操作，下节我们将讨论权限撤销的实现机制。

4 权限撤销的实现机制

安全核作为操作系统的非功能性模块，要求尽可能少地涉及系统功能，以保持其独立性，从而便于验证和移植。权限撤销是安全核的一个重要功能，实现上力求简洁、适用，系统开销小，不应像数据库系统中实现那样复杂。基于安全系统的特点，我们提出了用哈希授权链表和基于关键度的回调函数等机制来实现安全核的权限撤销。

4.1 哈希授权链表

在以前的权限撤销研究文献中使用过CRL(证书撤销链)[9,10]、CRS(证书撤销系统)[11]和CRT(证书撤销树)[12]等实现机制，这些实现以表或树的方式处理授权和撤销操作，涉及到认证、加密和网络通信等技术，有一定的可参考性。根据安全核的特点，我们提出了基于哈希表的授权链表来辅助权限撤销操作。操作系统中主客体繁多，授权链多而交织，利用哈希表便于快速搜索链表。哈希链以客体的所有者为链头，按权限的发布形成链，如图4所示。其中每个节点被称为授权链节点，授权链节点又包含了授权属性节点，其属性包括授权者、被授权者、客体、客体的拥有者、权限、授权时间、递归次数。属性节点和链节点定义如下。

```
typedef struct Auth_Node
{
    sec_id g_subject_id;
    perm_t g_perm;
    sec_id g_object;
    sec_id grantor;
    sec_id owner;
    time_t g_time;
    int gnum;
}Auth_Node_t

    suruct Auth_List_Node
{
  Auth_Node_t * Au_node;
  struct Auth_List_Node * next;
}
```

同一主体的授权链往往容易混淆，而客体所有者、授权时间、权限这三要素是最好的区分属性，为了区别同一发布者发布的不同权限，避免重值，授权链的哈希值采用(perm，object，owner，gt)计算，以(p，o，A，gt)为例(A为客体所有者)，哈希算法为：

hash_value(⟨p，o，A，gt⟩)= (gt|(A ^ (o≪2) ^ (p≪4)))& (hash_slots_size-1)。授权链表具有遍历、定位插入、

删除等功能，以配合权限删除操作。

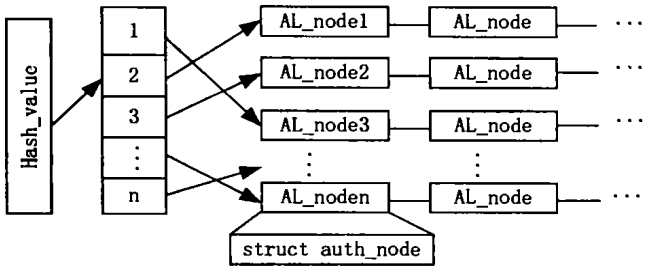

图4 Hash 授权链表

在安全操作系统中，每个任务的 TCB(任务控制块)中都有存储安全属性的结构，授权者、接受者及客体的安全结构中包含有授权节点指针，任务在作安全操作时，都进行撤销检查，确保实时地响应策略升级和用户请求。

4.2 基于关键度的回调函数

权限撤销中最复杂和最关键的步骤是如何合理地处理相关进程的操作。当系统的安全策略改变时，进程处于不同的状态，进程的关键操作有的已经完成，有的还在计算中，其数据可能处于中间处理状态，如果武断地终止进程的操作，抛弃关键数据或杀死进程，将给用户带来不可预测的后果，在高可信赖系统中这是不可接受的。然而，安全核又要及时地更新策略，保证不阻塞新的授权请求，所以不能等待进程在旧的策略下完成操作，二者是矛盾的。要求找到既能保证用户不会丢失关键数据，又能使策略更新及时完成的方法。基于关键度的回调函数就是其中的一种解决方法。

用户任务的数据及关联计算具有不同的关键度，如同任务的优先级一样，在权限撤销时，根据数据的关键度作不同的处理(或作简单计算保存，或抛弃)，将任务恢复到某一良好状态，不会破坏后续操作。虽然衡量任务和数据的关键度对系统来说是困难的问题，可是应用编程者却能够正确定位单个任务及其数据的状态，并根据关键度作相应的处理。所以安全核给编程者提供了称为回调函数的接口，在安全策略改变时，如果进程不再具有对特定客体的访问权限时，执行该函数并根据任务数据的关键度作定制处理，将进程保持在某一良好的状态，尽量保持进程的一致性和连续性。比如被阻塞在消息队列的进程将被唤醒，按预先的设定保存数据、恢复状态。回调函数在系统授权时被注册到访问缓存中，当全局策略改变或进程提出权限撤销请求时，访问缓存要对先前已授予的权限重新验证，如果验证旧的授权已无效，则策略实施者将调用访问缓存中的回调函数作权限撤销前的操作。

让安全核无区别地处理权限撤销时的进程操作是不现实的，也具有不可预测的破坏性。而引入基于关键度的回调函数，把这种差异性留给编程者处理是合理和灵活的，也是切实可行的方法。

系统撤销操作最后由回调函数和两种撤销函数组成，其实现方法简化如下：

```
Sys_revoke(auth_node_t*an,
(int*)callback(),revoke_opt…)
{
callback();//撤销前的工作
base_revoke();//撤销第一级授权
if(revoke_opt==RECURSIVE_OPT)//是级联撤销
{   while(an->next!=NULL){//遍历授权链
    cascade_revoke(); //撤销后续授权
    an=an->next;
    }
}
else//否则非级联撤销
{   takeover_node()//接管后续授权
    while(an->next!=NULL){//遍历授权链
      cascade_revoke(); //撤销后续授权
      an=an->next;
    }
}
```

系统撤销操作先调用访问缓存中注册的回调函数作撤销前的工作(包括关键数据的保存，任务状态的恢复等)，然后撤销后续的基本授权(即直接授权)，判断是否级联撤销，如是则遍历授权链，撤销后续的授权；如否即是非级联的撤销，先接管后续的授权，接管操作包括接管的授权的合法性判定(即是否符合授权策略)和创建新的节点，最后遍历授权链进行级联撤销，更新授权链表。

结束语 支持动态的安全策略是安全操作系统的重要特性，其实现的基础是系统具有合理和适用的权限撤销机制，而现有安全操作系统对权限撤销操作支持不足。本文根据安全核的特点，提出了一种结合级联和非级联两种互补特性的权限撤销机制，用于点到点和点到面的权限撤销。其实现机制采用了哈希授权链表来解决复杂的授权链的问题，而对于权限撤销可能对任务的关键操作带来的不可预料的破坏问题，则采用基于关键度的回调函数进行了解决。我们已在嵌入式实时操作系统 CRTOS 的安全核上实现了该撤销机制。实践表明，它能够满足安全操作系统可能有的绝大多数权限撤销操作需要，本文的后续工作将进一步研究基于组或域的权限撤销机制。

参 考 文 献

1 Majetic I, Leiss E L. Authorization and Revocation in Object-Oriented Databases. IEEE Trans. on Knowlegde and Data Engineering, 1997, 19(4)

2 Bertino E, Jajodia S, Samarati P. A Non-Timestamped Authorization Model for Relational Databases. In: Proc. of the 3rd ACM Conf. on Computer and Communications Security, New Delhi, India, 1996. 169～178

3 Hagstrom A, et al. Revocations-A Classification. In: 14th IEEE Computer Security Foundations Workshop (CSFW'01), 2001

4 Secure Computing Corporation. DTOS Lessons Learned Report. DTOS CDRL A008, June 1997

5 Spencer R, et al. The Flask Security Architecture: System Support for Diverse Security Policies. In: Proc. of the 8th USENIX Security Symposium, Aug. 1999

6 Loscocco P, Smalley S. Integrating Flexible Support for Security Policies into the Linux Operating System: [NSA Technical Report]. http: // www.nsa.gov/selinux/slinux-abs.html, October, 2000

7 吴新勇，熊光泽. 支持动态策略的安全核(Security Kernel)机制的研究. 计算机科学，2002，29(11)：154～156

8 Hawblitzel C, von Eicken T. Type System Support for Dynamic Revocation. ACM SIGPLAN Workshop on Compiler Support for System Software, May 1999

9 Housley R, et al. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459, Jan. 1999

10 Cooper D A. A more efficient use of delta-CRLs. In: Proc. of the 2000 IEEE Symposium on Security and Privacy, May 2000. 190～202

11 Micali S. Efficient certificate revocation: [Technical Memo MIT/LCS/TM-542b]. Massachusetts Institute of Technology, Laboratory for Computer Science, March 1996

12 Kocher P. A Quick Introduction to Certificate Revocation Trees (CRTs): [Technical report]. ValiCert Inc. 1999