An operating system oriented RBAC model and its implementation

Article · February 2004

CITATIONS
19

READS
90

2 authors, including:

Zhiyong Shan
Renmin University of China
20 PUBLICATIONS **217** CITATIONS

SEE PROFILE

一个应用于操作系统的 RBAC 模型及其实施*

单智勇　孙玉芳

（中国科学院软件研究所开放系统与中文信息处理中心　北京　100080）
E-mail: szy@sonata.iscas.ac.cn

摘要　本文主要研究了经典角色访问控制模型面向操作系统的扩展及其在内核中的实施方法。首先在 RBAC96 模型的基础上，引入可执行文件实体、细分操作系统权限，形成适用于操作系统的角色访问控制模型 OSR（Operating System oriented RBAC model）并给出简单的形式化描述。然后，采用 GFAC 与 Capability 相结合的方法在安全 Linux 内核中实现 OSR 模型。

关键词　操作系统，RBAC，访问控制

中图法分类号　TP309

基于角色的访问控制（Role-Based Access Control 简称 RBAC）是九十年代兴起的一种有效的访问控制方法,是由美国国家标准化和技术委员会（NIST）的 Ferraiolo 等人在 1992 年提出的[1]。乔治麻省大学（George Mason University）的 Sandhu 教授于 1996 年又提出 RBAC96 模型[2]，成为经典的角色访问控制模型。

操作系统安全是计算机安全系统的基础，“ 忽略操作系统安全的安全系统是建立在沙滩上的城堡 ”[4]。探讨如何在操作系统中实施 RBAC 是一项很有意义的工作，可以利用 RBAC 的以下优点[2] [5] [9]加强操作系统的安全性：

- 政策中立。在操作系统中支持多安全政策是操作系统安全的发展趋势[6] [7]。以 RBAC 作为基础的访问控制机制，可以使操作系统支持多安全政策，而且，有利于不同操作系统之间的互操作[5]。
- 良好地支持最小特权安全原则(Principle of Least Privilege)[8]。使用户和应用程序只限于拥有完成其任务所必需的权限，防止滥用权限危害系统安全。
- 良好地支持职责分离（Separation of Duties）的安全原则。使相互冲突的两个职位之间不能相互兼任。如：限制系统管理员和安全管理员之间的兼任。
- 支持数据抽象。将对客体的操作许可抽象为更有意义的表达，而不仅仅是操作系统通常提供的读写执行等操作。
- 便于安全管理。使得安全管理员能够借助角色访问控制机制直观方便地描述和实施组织安全政策，同时也方便安全管理员进行权限管理。

但是，目前对 RBAC 的实现研究主要集中在应用层[2][3][23][24][25][26][27]，如：Web 服务器和数据库。对于在操作系统中的实现研究比较少而且不完善[2] [3]。

本文在实施 RBAC 之前，针对操作系统的特点对 RBAC96 模型进行了细化和扩展，并给出简单的形式化描述。然后，采用 GFAC 和 Capability 机制相结合的办法在 Linux 内核中实现 OSR 模型。

*本文研究得到国家自然科学基金（60073022）、国家 863 高科技项目基金（863-306-ZD12-14-2）、中国科学院知识创新工程基金（KGCX 1-09）、北京市重点技术创新项目：基于红旗 Linux 的安全操作系统、中科院软件所培育基金（CXKE5143）资助。作者**单智勇**，1974 年生，博士生，主要研究领域为系统软件安全性。**孙玉芳**，1947 年生，研究员，博士生导师，主要研究领域为系统软件和中文信息处理。

1 扩展 RBAC96 模型

RBAC96 是一个模型族[2]，包括 RBAC0、RBAC1、RBAC2 和 RBAC3 等模型，其中 RBAC3 是前三者的综合（见图 1）。RBAC3 包括以下内容：

1） 用户（User）、角色(Role)、权限(Permission)、会话(Session)等四个实体。

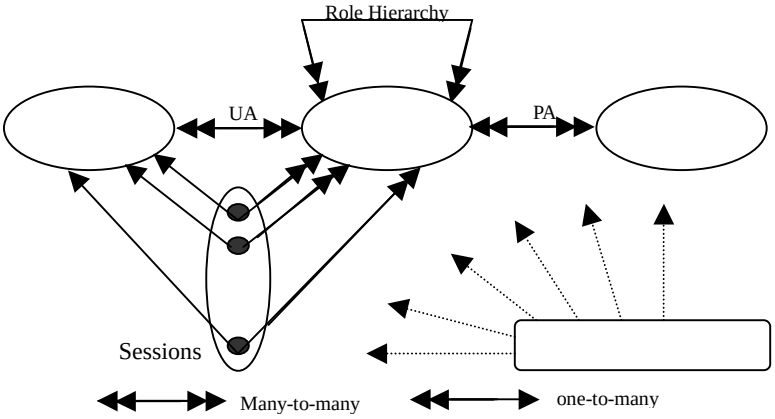

图 1 RBAC3 模型

2）权限分配（Permission Assignment）、用户分配（User Assignment）、角色继承(Role Hierarchy)和限制（Constraints）等四种关系。

3）用户→会话和会话→角色两种映射。

RBAC96 是一个高度抽象的通用模型，当应用到操作系统中时存在两点不足。

其一，可执行文件决定进程（进程对应于 RBAC96 的会话实体）的行为流程，应该参与权限的分配与传递，但是 RBAC96 模型并没有反映。依据 RBAC3，进程在运行中需要的权限由进程的角色决定，而进程角色由进程用户传递来，但是，一方面，有些程序运行后需要用户之外的特殊访问权限，比如 UNIX 中修改口令的命令 passwd，需要系统管理员的权限去修改 passwd 文件或 shadow 文件，当普通用户运行 passwd 命令时，RBAC3 无法反映这种权限传递。借助 UNIX 中的 setuid 机制和 Linux 中的 Capability 机制可以使进程获得特殊访问权限，但是 setuid 机制破坏最小特权原则，会带来安全漏洞。Capability 机制只用于超级用户特权，而不是操作系统中的所有权限，有其局限性。另一方面，有些程序运行后需要限制其权限范围，比如限定程序只能访问指定的文件，这对于防止程序破坏信息的保密性、完整性和隐私都是有益的，尤其是可以作为防止恶意代码攻击的手段。所以，如果将角色授予可执行文件，进程运行后继承文件的角色，可以使进程获得完成工作需要的特定权限，同时也可以借助可执行文件的角色限制进程的权限范围。

其二，操作系统中的权限比较复杂，RBAC96 模型缺乏细致的描述，不便于将模型实施。权限可以理解为主体所持有的，在执行一项任务或进行一次访问前由访问控制机制检查的，一种标志。从这个理解出发，操作系统中所有的权限应该分为普通权限、应用层权限和特殊权限，见图 2。

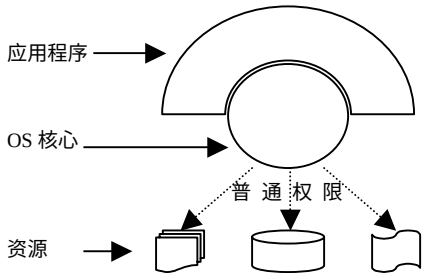

图 2 操作系统访问控制权限分析

普通权限是主体访问客体前，在内核中实施检查的标志。通常的读、写文件等操作都是普通权限。

应用层权限是主体执行一项任务或操作前，在应用程序中实施检查的标志，这种任务或操作的粒度由应用程序的内在逻辑决定，可以是控制整个系统的操作，也可以是对某个文件一个字节的访问。操作系统的整体包括内核和必要的应用程序（如各种 shell 命令），它们之间有

着不可分割的联系。因此有必要在应用程序和内核之间实施统一的访问控制，这样不仅可以实现粒度灵活的访问控制而且能借此加强操作系统整体安全。因为，操作系统中的客体往往是文件、目录、设备等等，依据访问控制矩阵[21]，若只在内核中进行访问控制，最细只能控制到对单个客体的访问，而在应用程序中加入访问控制语句后可以根据需要决定控制的粒度。比如：“ 修改其他用户的口令 ” 是一项重要的系统管理权限，应该限制只有系统管理员角色才能拥有，但是在安全操作系统中同时又要求系统管理员不能修改其它特权用户的口令，如安全管理员和审计管理员的口令。若只在内核中进行访问控制，以 passwd 可执行文件作为客体，则只能控制到 “ 是否可以运行 passwd 命令修改其他用户的权限 ”，而很难区分当前要修改的是什么用户。因此有必要在 passwd 应用程序中加入相应的语句：不允许系统管理员修改其他特权用户的口令。这种细化的控制对操作系统安全有很大影响，因为如果系统管理员可以修改其他特权用户的口令，实际上就拥有了他们的权限，这破坏了最小特权原则和职责分离原则。因此，借助应用层权限的概念安全模型可以将内核和应用层的访问控制统一起来，细化控制粒度并借此为操作系统的整体安全提供保证。

特殊权限是主体在执行一项维持安全操作系统正常而安全地运转的任务前，在内核中检查的标志，比如 “ 改变(任何)文件的属主 ”、“ 网络管理 ”、“ 系统重起 ” 等等权限都属于这种情况。可以分为系统管理特权、安全管理特权和审计特权。系统管理特权是维护系统正常运行的特权；安全管理特权是维护系统安全的特权；审计特权是进行系统安全审计的特权。这三种特权相互牵制相互监督。

Sandhu 教授在他的论文[22]中已经指出：RBAC 模型是一个开放式的概念模型，可以依据不同的应用进行扩展。所以，我们针对上述不足扩展 RBAC3，得到 OSR（Operating System oriented RBAC model）模型，见图 3。OSR 引入可执行文件作为实体，并且将操作系统所有权限分为普通权限、特殊权限和应用层权限。这样，角色可以包含三种权限，可以被分配给用户和可执行文件，进程同时从用户和可执行文件继承角色。后面一节给出模型的简单形式化描述。

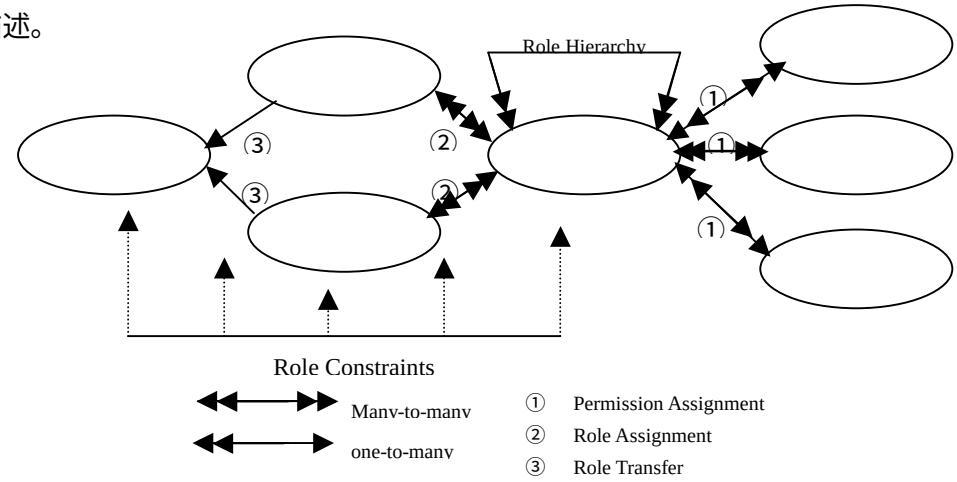

图 3　OSR 模型

2　OSR 模型的简单形式化描述

OSR 模型实施时，将普通权限表达为一个客体和操作的二元组，需要占用巨大的空间和降低系统效率，应该将客体分类，针对客体类型及其上的操作确定权限。所以，引入客体、客体种类、客体类型和操作等概念到 OSR 模型中。

2.1 有关角色、用户、进程和可执行文件的定义

定义 1 用户是一个可以独立访问计算机系统的主体，记为：u，系统所有用户构成集合 U（User）。

定义 2 可执行文件是决定进程行为的特殊客体，记为：f，系统所有可执行文件的构成集合 F (executable File)

定义 3 进程是代表用户访问系统资源的主体，记为：p，系统当前所有进程构成集合 P (Process)。

定义 4 角色是一个用户或进程为完成一项工作所需要的权限的集合，记为：r，系统所有角色构成集合 R (Role)。

从应用的角度说，角色是一项工作的描述、是一个职位的描述。从实现的角度来说，角色是一个权限的集合，也是联系主体和权限的桥梁。

2.2 有关客体的定义和规则

在 OSR 模型中，将除用户外的所有客体划分为五个种类：进程、文件目录、进程间通信、设备和系统控制数据等五个种类。“ 系统控制数据 ” 包括系统时钟、主机名和域名等在整个系统范围内起作用的数据。“ 文件目录 ” 表示系统中文件和目录的集合，将文件和目录划分为同一种类的原因在于它们有很多共同的操作方式。每个种类的客体又划分为若干个类型。一个类型代表一个部门的资源集合、或是一个用户的资源集合、或是一个应用服务要访问的资源集合、也可以是同一安全等级的客体集合。一个客体可以同时属于多个类型。因为部门间、用户间存在共享资源。

定义 5 客体记为：ob，系统中所有的客体（除用户外）构成集合 Ob；所有相同种类客体的集合记为：C；每个种类又划分为若干类型，记为：T；

所有的客体分为五个种类：

$$\bigcup C_i = Ob$$

C_1 代表进程客体的集合，C_2 代表文件目录客体的集合，C_3 代表设备客体的集合，C_4 代表进程间通信客体的集合，C_5 代表系统控制数据客体的集合。

以系统中所有种类的下标为元素，构成集合 Classes：

$$Classes := \{ 1,2,\ldots 5 \}$$

以系统中所有客体种类划分的类型数目为元素，构成集合 TypeCounts：

$$TypeCounts := \{ N_i \mid i \in Classes \}$$

每个种类的客体分为若干类型：

$$C_i := \{ T_{i1}, T_{i2}, \ldots T_{in} \} \quad i \in Classes \quad n \in TypeCounts$$

以系统中所有类型的下标为元素，构成集合：Types

$$Types := \{(i,j) \mid i \in Classes, j \in \{1,2,\ldots n\}, n \in TypeCounts \}$$

规则 1 同一个客体不能同时属于多个种类：

$$C_i \cap C_j = \varnothing \qquad i \neq j \quad i,j \in Classes$$

规则 2 同一客体可以同时属于同一种类的多个类型：

$$\exists\ T_{ij}, T_{ik} \subseteq C_i \qquad \text{有：}\ T_{ij} \cap T_{ik} \neq \varnothing \quad \text{成立}$$

其中：$(i,j),(i,k) \in Types \quad i \in Classes$

2.3 有关操作的定义和规则

定义 6 操作是对客体的一次访问，记为：op，系统所有操作构成集合 Op (Operation)，针对不同的客体种类有不同的操作子集合，所以有：

$Op := \{ Op_i \}$　$i \in Classes$　Op_i 代表对 C_i 类客体的操作子集合

规则 3　所有操作子集合的并集为操作集合：$\bigcup Op_i = Op$　$i \in Classes$

操作子集合之间允许有相同的操作：$\exists\ Op_i, Op_j$　$Op_i \cap Op_j \neq \varnothing$　$i, j \in Classes$

2.4 有关权限的定义和规则

定义 7　权限是一次操作许可的抽象，记为：a，系统所有权限构成集合 A (Authorization)。系统所有权限又分为三个集合：

$A := \{ A_g, A_s, A_a \}$

A_g 代表普通权限集合，A_s 代表特权集合，A_a 代表应用层权限集合

普通权限是对指定类型客体的操作许可，记为：

$a(op_i, T_{ij})$　$i \in Classes$　$(i,j) \in Types$

op_i 是对 i 种类客体的操作，T_{ij} 是 i 种类客体的第 j 个类型

特殊权限是执行特殊任务的许可，记为：a_s 。特殊权限集合包括三个子集合：

$A_s := \{ A_{sys}, A_{sec}, A_{aud} \}$

A_{sys} 代表系统管理特权集合，A_{sec} 代表安全管理特权集合，A_{aud} 代表审计特权集合

应用层权限是在应用层进行检查的操作许可，记为：a_a

2.5　模型中的关系

定义 8　OSR 模型包含七个分配关系：

- 客体与类型的分配关系 ObT，$ObT \subseteq Ob \times Types$，描述如下：
  $ObT = \{(ob,(i,j)) \mid ob \in Ob, (i,j) \in Types\}$
- 权限与角色的分配关系 AR，$AR \subseteq A \times R$，描述如下：
  $AR = \{(a(op_i, T_{ij}),r) \mid a(op_i, T_{ij}) \in A_g, r \in R\} \cup \{(a_s,r) \mid a_s \in A_s, r \in R\} \cup \{(a_a,r) \mid a_a \in A_a, r \in R\}$
  $= \{(a(op_i, T_{ij}),r) \mid a(op_i, T_{ij}) \in A_g, r \in R\} \cup \{(a_s,r) \mid a_s \in A_{sys}, r \in R\} \cup \{(a_s,r) \mid a_s \in A_{sec}, r \in R\} \cup \{(a_s,r) \mid a_s \in A_{aud}, r \in R\} \cup \{(a_a,r) \mid a_a \in A_a, r \in R\}$
  其中：$i \in Classes$　$(i,j) \in Types$
- 角色与用户的分配关系 RU，$RU \subseteq R \times U$，描述如下：
  $RU = \{(r,u) \mid r \in R, u \in U\}$
- 角色与可执行文件的分配关系 RF，$RF \subseteq R \times F$，描述如下：
  $RF = \{(r,f) \mid r \in R, u \in F\}$
- 角色间的继承关系 RH，$RH \subseteq R \times R$。
  若 $(r_1, r_2) \in RH$，可以记为：$r_1 \geq r_2$，读作 “r_1 继承 r_2”。描述如下：
  $RH = \{(r_1,r_2) \mid r_1 \in R, r_2 \in R\}$
- 角色间的静态冲突关系 RSC，$RSC \subseteq R \times R$，描述如下：
  $RSC = \{(r_1,r_2) \mid r_1 \in R, r_2 \in R\}$
  有静态冲突关系的两个角色不能赋予同一用户或可执行文件，也不能存在于同一进程中。
  静态冲突关系可以通过继承传递：
  $\forall\ r_1, r_2, r_3$　$r_1 \geq r_2$，$(r_2, r_3) \in RSC \Rightarrow (r_1, r_3) \in RSC$
- 角色间的动态冲突关系 RDC，$RDC \subseteq R \times R$，描述如下：
  $RDC = \{(r_1,r_2) \mid r_1 \in R, r_2 \in R\}$
  有动态限制关系的两个角色可以赋予同一用户或文件，但是无法在进程中同

时将他们激活。

动态冲突关系可以通过继承传递：

$\forall r_1, r_2, r_3 \quad r_1 \geq r_2$ ，$(r_2, r_3) \in RDC \Rightarrow (r_1, r_3) \in RDC$

定义 9 OSR 模型包含七个映射关系：

用户具有两个角色集合：最大角色集合和活动角色集合。最大角色集合由安全管理员分配。活动角色集合是用户在开始会话时要求的，是最大角色集合的子集，用户也可以使用命令或系统调用动态改变活动角色集合。活动角色集合的意义在于贯彻最小特权原则，用户可以依据工作的需要，选择最小的活动角色集合。比如用户在执行某些不可信程序前，可以去活一些涉及用户私人秘密的角色，防止程序的恶意访问。

- User_max_roles: $U \rightarrow 2^R$ 用户到最大角色集合的映射关系，表达如下：

  $proc\_max\_roles(u_i) = \{ r \mid (\exists r` \geq r)[(r`,u_i) \in RU]\}$
- User_active_roles: $U \rightarrow 2^R$ 用户到活动角色集合的映射关系，表达如下：

  $proc\_active\_roles(u_i) = proc\_max\_roles(u_i) - R`$

  其中 $R` \subseteq user\_max\_roles(u_i)$ 是用户去活的角色集合（在登录时或者在系统中工作时）。— 代表集合的相对补运算（以下同）。
- File_roles: $F \rightarrow 2^R$ 可执行文件到角色集合的映射关系，表达如下：

  $file\_roles(f_i) = \{ r \mid (\exists r` \geq r)[(r`,f_i) \in RF]\}$
- proc_user : $P \rightarrow U$ 进程到用户的映射关系
- proc_file : $P \rightarrow F$ 进程到可执行文件的映射关系
- proc_max_roles : $P \rightarrow 2^R$ 进程到最大角色集合的映射关系，表达如下：

  令：$S_1 = \{ r \mid user\_active\_roles(proc\_user(p_i)) \cup \{ r \mid file\_roles(proc\_file(p_i))\}$

  则：$proc\_max\_roles(p_i) = S_1 - \{ r \mid \exists r` \in S_1, r \in S_1, (r,r`) \in RSC \}$
- proc_active_roles: $P \rightarrow 2^R$ 进程到活动角色集合的映射关系，表达如下：

  令：$S_2 = proc\_max\_roles(p_i)$

  则：$proc\_active\_roles(p_i) = S_2 - \{ r \mid \exists r` \in S_2, r \in S_2, (r,r`) \in RDC \} - R``$

  其中 $R`` \subseteq proc\_max\_roles(p_i)$ 是进程去活的角色集合

定理 1 对于进程 p_i，它的权限可依据下式计算：

$proc\_authorities(p_i) = \{a \mid (a,r) \in AR, r \in proc\_active\_roles(p_i)\}$

角色是权限的集合，所以进程当前的活动角色集合，决定了进程当前所拥有的权限集合。

2.6 进程角色集合变化规则

规则 4 进程拥有两个角色集合：最大角色集合和活动角色集合。它们的变化规则如下：

进程产生后：$proc\_max\_roles(p`) = proc\_max\_roles(p)$

$proc\_active\_roles(p`) = proc\_active\_roles(p)$

执行新文件后：$proc\_max\_roles`(p) = proc\_active\_roles`(p) = file\_roles(f) \cup user\_active\_roles(u)$

进程运行中：$proc\_active\_roles`(p) = proc\_active\_roles(p) \cup N$

或者：$proc\_active\_roles`(p) = proc\_active\_roles(p) - N$

p 表示进程、f 表示可执行文件、u 表示用户、N 表示用户通过程序或命令要求增加或减少的角色集合、` 标志该集合为执行一个操作之后的角色集合或新的主客体。

2.7 访问决策的规则与定理

规则 5 OSR 模型的访问决策规则：若进程请求的权限被包含在进程权限集合中，则许可，

否则拒绝。

定理 2 如果进程 $p \in P$，请求对客体 object $\in$ Ob 的操作 operation $\in$ Op，而且下式成立，则判定许可。

$$operation \in \{op \mid a(op, T_{ij}) \in proc\_authorities(p) \cap A_g, (object,(i,j)) \in ObT\}$$

进程的普通权限集合决定了进程可以对特定类型客体的许可操作集合，所以，如果当前操作被包含在这个许可操作集合中，则允许访问。

定理 3 如果进程 $p \in P$，请求特殊权限或应用层权限 authority $\in A_s \cup A_a$，而且下式成立，则判定许可。

$$authority \in proc\_authorities(p) \cap (A_s \cup A_a)$$

显然，如果请求的特殊权限或应用层权限被包含在进程权限集合中，则允许访问。

3 OSR 模型实现

我们采用 GFAC 与 Capability 机制相结合的方法在 Linux 内核中实施 OSR 模型。通用访问控制框架 GFAC 是一种支持多安全政策模块的访问控制实施方法[10]，利用它可以方便地向可信计算机系统加入新的安全政策模块，而且也很容易验证新安全政策模块的正确性[11]。Capability 机制是 Linux 内核中的一种访问控制机制，将超级用户的特殊权限分割成一组特权，每个特权对应一个 Capability，进程依据其拥有的 Capability 决定它可以完成哪些特权操作。利用 Capability 机制可以实现对超级用户特权的控制和管理

总体的实施框架见图 4。主要分为 Capability 机制和 GFAC 实施框架两部分，Capability 机制部分完成对系统管理特权（即原来超级用户的特权）的访问控制，GFAC 部分完成对普通权限、应用层权限、安全管理特权和审计特权的访问控制。另外有安全管理部分，由 GUI 管理工具、管理命令和安全管理系统调用组成，主要用于管理访问控制信息。下面依据图 4 介绍 OSR 模型的实施。

3.1 GFAC 实施部分

依据GFAC方法，可信计算基（Trusted Computing Base TCB）由访问控制实施部分（Access Enforcement Facility AEF）和访问控制决策部分（Access Decision Facility ADF）组成。ADF 实施了若干系统安全政策和一个元政策，决定进程的访问请求是否被许可。AEF 依据 ADF 的决策结果来控制安全相关系统调用的执行。

在 OSR 模型的实现中，内核的访问控制系统分为三个部分：AEF 部分、ADF 部分和 ACI（Access Control Information）部分。ACI 部分管理访问控制信息（ACI，即：主客体的安全属性）。图 5 表达了三个部分之间的关系：

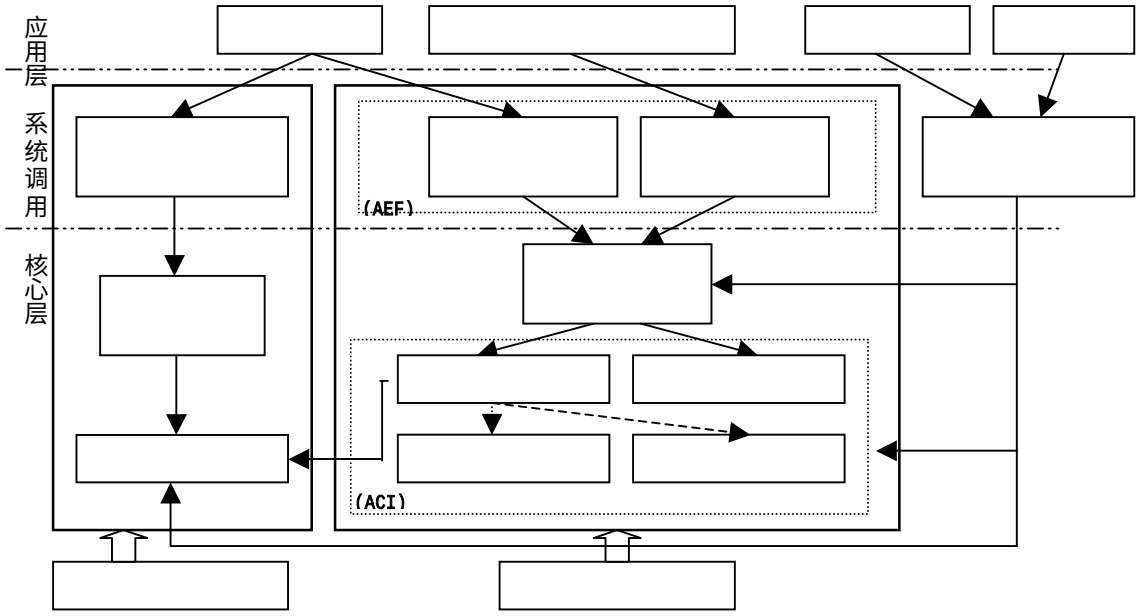

图 4　OSR 模型在 Linux 中的总体实施框架

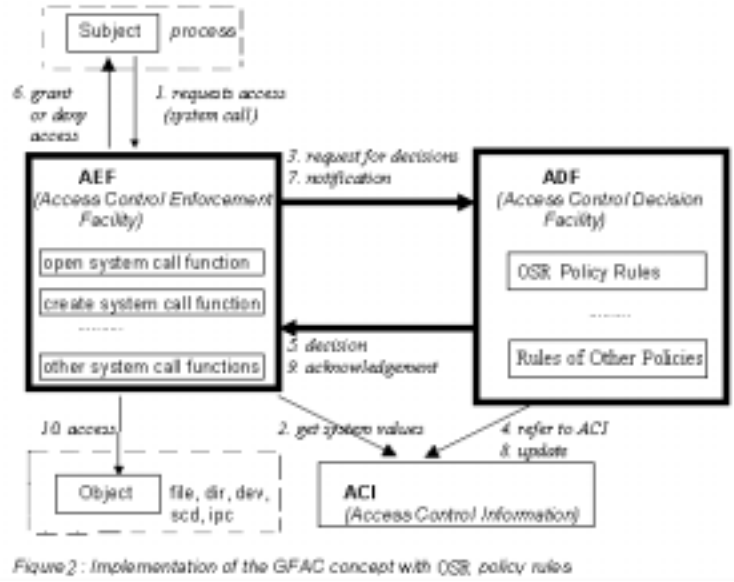

Figure 2 : Implementation of the GFAC concept with OSR policy rules

图 5　GFAC 框架的实施

发生访问时，AEF 从安全相关的系统调用中发送一个访问请求给 ADF。访问请求的参数包括：请求类型、发送请求进程的标识、要求访问客体的标识。ADF 先依据请求类型、政策规则和相关访问控制信息判断当前请求是否满足各个安全政策，然后由元政策综合各个安全政策的判断结论计算出最终的结果。若允许访问，则系统调用继续执行，并且在成功执行后由 AEF 通知 ADF 修改相应的访问控制信息。若禁止访问，则系统调用中断执行，并且返回一个错误给进程。

AEF 部分是通过在所有安全相关的系统调用中添加访问请求和修改通知来实现的。OSR 实施中，系统调用和访问请求的对应关系如表 1：

Table 1

表 1

ADF request	System calls	ADF request	System calls
ADD_TO_KERNEL	Create_module(),　init_module()	READ	Readdir(), readlink(), getdent()
ALTER	Ipc(), msgctl(), shmctl()	READ_ATTRIBUTE	Rslx_get_attr()　注：新增用于安全管理的系统调

			用
APPEND_OPEN	Open(), msgsnd()	READ_OPEN	Ipc(), open(), msgrcv(), shmat()
CHANGE_GROUP	Chgrp(), fchgrp(), setgid(), setfsgid(), setregid(), setgroups()	READ_WRITE_OPEN	Ipc(), socketcall(), open(), shmat()
CHANGE_OWNER	Chown(), fchown(), setuid(), setfsuid(), setreuid()	REMOVE_FROM_KERNEL	Delete_module()
CHDIR	Chdir(), fchdir()	RENAME	Rename()
CLONE	Fork(), clone()	SEARACH	Kernel-internel
CREATE	Create(), ipc(), socketcall(), mkdir(), mknod(), symlink(), open(), msgget(), shmget()	SEND_SIGNAL	Kill()
DELETE	Ipc(), socketcall(), rmdir(), unlink(), msgctl()	SHUTDOWN	Reboot()
EXECUTE	Exec_ve()	SWITCH_LOG	Rslx_adf_log_switch() 注：新增用于开关 ADF 日志的系统调用
GET_PERMISSIONS_DATA	Access()	SWITCH_MODULE	Rslx_switch() 注 :新增用于开关安全政策模块的系统调用
GET_STATUS_DATA	Stat(), fstat(), lstat(), new_stat(), new_fstat(), new_lstat(), statfs(), fstatfs(), msgctl(), shmctl()	TERMINATE	exit()
LINK_HARD	Link()	TRACE	Ptrace()
MODIFY_ACCESS_DATA	Utime()	TRUNCATE	Open(), truncate(), ftruncate()
MODIFY_ATTRIBUTE	Rslx_set_attr() 注：新增用于安全管理的系统调用	UMOUNT	Umount()
MODIFY_PERMISSIONS_DATA	Chmod(), fchmod(), ioperm(), iopl()	WRITE	Rename()
MODIFY_SYSTEM_DATA	Adjtimes(), stime(), settimeofday(), sethostname(), setdomainname(), setrlimit(), swapon(), swapoff(), syslog()	WRITE_OPEN	Open()
MOUNT	Mount()	CHECK_APP_RIGHT	Rslx_rac_check_app_right() 注 :新增用于检查应用层权限的系统调用

ACI 部分的作用是管理访问控制信息。访问控制信息主要包括主体、客体和角色的安全属性。访问控制信息的读取、添加、删除和修改都必需通过 ACI 函数来进行，任何其它方式都无法访问。与 OSR 模型实施相关的主要访问控制信息如表 2：

Table 2

表 2

	安全属性名	解释		安全属性名	解释
角色	child_roles	直接子角色列表,表达角色继承关系	进程	Rac_types	进程所属类型列表
	static_conflict__roles	静态冲突角色列表		Max_roles	进程最大角色集合
	dynamic_conflict_roles	动态冲突角色列表		Active_roles	进程活动角色集合
	Fd_right_vectors_array	对文件目录客体的操作权限列表	用户	Max_roles	用户最大角色集合
	Dev_right_vectors_array	对设备文件客体的操作权限列表		Active_roles	用户活动角色集合
	proc_right_vectors_array	对进程客体的操作权限列表	文件目录	Rac_types	文件目录所属类型列表
	Ipc_right_vectors_array	对进程间通信客体的操作权限列表		Exec_file_roles	角色列表（只对可执行文件有效）
	scd_right_vectors_array	对系统控制数据的操作权限列表	设备	Rac_types	设备所属类型列表
	secadm_privileges	安全管理特权列表			
	sysadm_priviliges	系统管理特权列表	进程间通信	Rac_types	进程间通信所属类型列表
	audadm_priviliges	审计特权列表			
	app_privileges	应用层权限列表列表			

由上表可以看出，角色的权限由九个权限列表所刻画。三个特殊权限和应用层权限列表都是权限向量，每一位代表一个权限。普通权限的五个权限列表都是权限向量组，组中的每个权限向量代表该角色对一种类型客体的权限集合，权限向量的每一位代表一个权限。

由于进程可以同时拥有很多角色，从效率的角度考虑，我们在实际实现中将进程所有激活角色的权限合并起来存放在进程访问控制信息中，以便 ADF 能够快速地判断进程是否有权限。

每种 SCD（系统控制数据）都有其固有的类型，所以不必保存 SCD 的类型信息。

OSR 模型的 ADF 实施部分可以用表 3 描述。其中请求 R_MAC_XX 和 R_IAC_XX 是从

ADF 的保密性强制访问控制和完整性访问控制政策模块发送来的，R_AUDIT_XX 是从审计系统调用以及审计守护进程发送来的，R_APPLICATION 是从应用层权限检查的系统调用发送来的。

Table 3

表 3

target / Request	T_FILE	T_DIR	T_DEV	T_PROCESS	T_IPC	T_SCD
R_ADD_TO_KERNEL	CR					
R_ALTER					CR	
R_APPEND_OPEN	CR		CR		CR	
R_READ_WRITE_OPEN						
R_CHANGE_GROUP	CR	CR			CR	
R_CHANGE_OWNER	CR	CR		①，SR，ST	CR	
R_CHDIR		CR				
R_READ						
R_SEARACH						
R_WRITE						
R_CLONE				②，SR，ST		
R_CREATE		③，ST			④，ST	
R_DELETE	CR	CR			CR	
R_EXECUTE	CR			⑤，SR，ST		
R_GET_PERMISSIONS_DATA						
R_GET_STATUS_DATA						CR
R_LINK_HARD	CR					
R_TRUNCATE						
R_MODIFY_ACCESS_DATA	CR	CR				
R_RENAME						
R_MODIFY_ATTRIBUTE	CP_sec					
R_MODIFY_PERMISSIONS_DATA	CR	CR			CR	CR
R_MODIFY_SYSTEM_DATA						CR
R_MOUNT		CR	CR			
R_READ_ATTRIBUTE	CP_sec					
R_READ_OPEN	CR	CR	CR		CR	
R_REMOVE_FROM_KERNEL						
R_SEND_SIGNAL				CR		
R_TRACE						
R_SHUTDOWN						
R_SWITCH_LOG						
R_SWITCH_MODULE						
R_TERMINATE				CR		
R_WRITE_OPEN	CR		CR		CR	
R_UMOUNT		CR	CR			
R_AUDIT_STOP	CP_aud					
R_AUDIT_SAVE_CONFIG						
R_AUDIT_RELOAD_CONFIG						
R_AUDIT_WORK						
R_AUDIT_START	CP_sys					
R_MAC_ADD_TO_KERNEL						
R_MAC_MOUNT						
R_MAC_SHUTDOWN						
R_MAC_REMOVE_FROM_KERNEL						
R_MAC_UMOUNT						
R_IAC_MODIFY_ATTRIBUTE	CP_sec					
R_IAC_READ_ATTRIBUTE						
R_MAC_GET_STATUS_DATA						
R_MAC_MODIFY_ATTRIBUTE						
R_MAC_MODIFY_PERMISSIONS_DATA						
R_MAC_READ_ATTRIBUTE						
R_MAC_SWITCH_LOG						
R_MAC_SWITCH_MODULE						
R_APPLICATION	CP_app					

注：

1. 以 CR 表示：检查进程对指定客体是否有普通访问权限；以 CP_sec 表示：检查进程是否有安全管理特权；以 CP_sys 表示：检查进程是否有系统管理特权；以 CP_aud 表示：检查进程是否有审计特权；以 CP_app 表示：检查进程是否有应用层权限。
2. 以 SR 表示：访问结束后，重新设置进程的角色集合；以 ST 表示：访问结束后，重新设置进程或客体的类型列表；
3. ①表示：判断静态角色冲突，若两个用户间或新用户和可执行文件间存在静态角色冲突则拒绝；②表示：检查进程是否对新进程所在的类型有 CREATE 权限；③表示：先检查是否有权限在该目录下创建客体，然后检查用户是否有权创建用户安全属性所指定的缺省客体类型；④表示：检查是否有权创建由用户安全属性指定的 IPC 类型；⑤表示：判断可执行文件和用户之间是否存在静态角色冲突，若是则拒绝。

ADF 的决策流程如图 6：

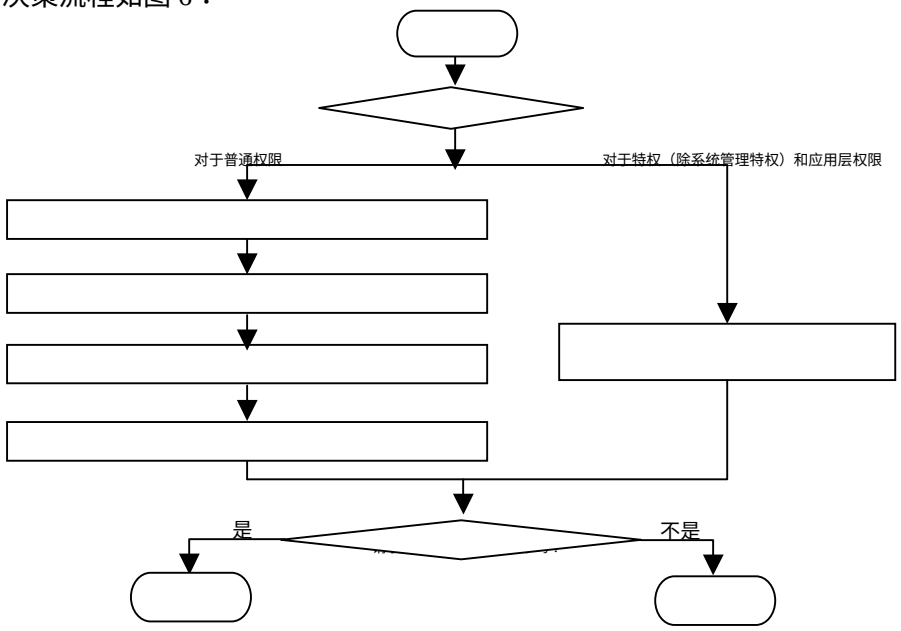

图 6　ADF 决策流程

3.2　Capability 实施部分

Linux 中的 Capability 机制是依据 POSIX 标准草案 POSIX draft 1003.1e[12]来实现的。Capabilities 是将超级用户的所有特殊权限分割后得到的一组权限。进程有三个 Capabilities 权限向量。在进程执行中，依据它的一个权限向量 effective capabilities 决定它当前拥有哪些超级用户特权。进程在产生时完全继承父进程的三个 Capabilities 向量，执行新映像文件后，按照一定的规则改变进程的三个 Capabilities 向量。

我们在 OSR 模型的实现中，只利用进程的 effective capabilities 权限向量，并且去除 Capability 机制的 Capabilities 向量变化部分。每当进程的活动角色集合发生变化，都依据活动角色集合计算出进程的系统管理特权，并立即更新 effective capabilities 权限向量。需要检查进程的超级用户特权时，内核依据 effective capabilities 权限向量进行判断。这样，进程当前活动角色集合所确定的系统管理特权，成为 Capability 机制实际判断权限的依据，就将 Capability 机制和 OSR 模型的实现统一起来了。

4 相关工作及比较

在操作系统中实施 RBAC 的研究比较有影响的系统有 Trusted Solaris 8、SE-Linux、LOCK6 和其它一些实验系统。

Trusted Solaris 8[13][14]中借助用户帐号来实现角色，是商业化比较成功的例子，实现方式简单，但是有其不可克服的缺点。首先，角色之间无法实现继承关系，而继承是 RBAC 模型的重要特点。其次，在同一时刻进程只能激活一个角色，激活的角色用户将代替进程原来的属主，所以，用户在系统中工作可能要反复地在角色和用户之间、角色和角色之间切换，而每一次切换都需要口令认证，显然给用户增加了操作负担。如果要减少切换次数，势必增加角色的权限（将多个角色的权限合并），这就违反了最小特权原则。由论文[2]也可以知道，允许进程同时激活多个角色是 RBAC 模型支持最小特权原则的关键。另外，Trusted Solaris 8 中的 RBAC 只是实现了对超级用户特权的访问控制，没有实现对操作系统所有权限的控制，所以其作用是有限的。

LOCK6[3]借助 Type Enforcement 机制实现 RBAC，但是没有真正在内核底层支持角色访问控制机制。以 Type Enforcement 访问控制机制为基础实现 RBAC，实际就在角色和权限之间增加了一层：访问控制域（domain），使系统变得更复杂了。因为，LOCK6 中，一个进程同时具有属主用户、角色和访问控制域三个可以决定其权限的属性，必须维护三者之间一致性，也就是说进程的这三个属性必须同时与系统的安全属性数据库一致：当前的角色是否在用户允许的角色集合中，当前的访问控制域又是否在角色允许的域集合中。其次，没有在内核中支持继承和职责分离等要求。再其次，在同一时刻进程只能激活一个角色，也就存在着与 Trusted Solaris 8 类似的问题。最后，LOCK6 不是一个主流的操作系统，所以限制了它的实用意义。

SE-Linux[15]是支持安全政策多样性和动态性最著名的安全操作系统，它在 FLASK 结构下实现 RBAC，但只支持“角色迁移”和“继承”，没有提供直接的机制支持角色间的限制关系，如角色间的互斥关系，所以它不便表达职责分离原则。另外，在同一时刻进程也只能激活一个角色，存在着与 Trusted Solaris 8 类似的问题。

本文的研究相比之下有几个优点：

- 直接在操作系统内核中实施 RBAC，而不是在其它机制的基础上实施（如 Trusted Solaris 8 借助用户帐号表达角色、LOCK6 借助 Type Enforcement 机制支持 RBAC），使系统的访问控制更简洁，而且便于以角色访问控制机制为基础支持多安全政策[16]。
- 对经典的 RBAC 模型做了面向操作系统的扩展。角色赋给可执行文件，可以更好地限制应用程序的权限范围，防止恶意盗用和滥用权限，有利于贯彻最小特权原则；能够控制整个操作系统的所有权限，包括普通权限、系统管理特权、安全管理特权、审计特权和应用层权限，符合访问监控机“监控所有访问”的要求[18]，也更有利于实现最小特权原则。
- 所实现的 RBAC 特性更全面，能更好的支持基于角色的访问控制安全政策。依据美国国家标准化和技术委员会（NIST）2000 年 12 月提出的 RBAC 标准草案[15]，RBAC 实施中应该支持的特性包括：基本 RBAC 要求、继承关系、静态职责分离关系和动态职责分离关系。在本文的工作中已经基本实现。
- 利用 GFAC 和 Capability 相结合的方法实现 RBAC。在对所有权限进行控制的前提下，最大限度地减少了对内核的修改，特别是对内核关键数据结构的修改。因为，ADF 和 ACI 是独立添加的部分，AEF 实际也只是在系统调用中添加了 ADF 访问请求和 ADF 属性设置通知。经过实际测试，在缺省配置下（实现了系统管理员、安全管理员、审计

管理员和普通用户角色），所有应用程序都能顺利运行。同时，这种实现方法较好地结合了 GFAC 和 Capability 机制，将超级用户的特权纳入角色机制进行统一控制，从而真正达到对 Linux 操作系统所有权限的控制。

本文的研究和实施工作已经加入红旗安全操作系统*3.0 版。希望该文能够对角色访问控制模型和安全操作系统的研究有所裨益。

---

* 红旗安全操作系统是由中国科学院软件研究所和中科红旗软件公司联合开发的基于 Linux 的安全操作系统[28]，于 2001 年 6 月通过公安部组织的国家信息安全评价标准第三级认证，于 2002 年 1 月通过中国科学院组织的“基于国际/国家标准的安全操作系统”（中科院成鉴字[2002]第 003 号）鉴定会的鉴定。

参考文献

1. Ferraiolo D F, Kuhn R. Role-Based access control. In: Proceedings of the 15th National Computer Security Conference. Baltimore, MD, 1992. 554~563, http://hissa.ncsl.nist.gov/kuhn/
2. R. S. Sandhu, et al. "Role-Based Access Control Models", IEEE Computer 29(2): 38-47, IEEE Press, 1996.
3. J. Hoffman. Implementing RBAC on type enforced systems. In Proceedings, 13th Annual Computer security applications conference. IEEE Computer Society Press, 1997.
4. "The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments." P. Loscocco et al, National Security Agency. November 1997. Proceedings of the 21st National Information Systems Security Conference.
5. David F. Ferraiolo, “An Argument for the Role-Based Access Control Model”, In: Proceedings of Sixth ACM Symposium on Access Control Models and Technologies, ACM SIGSAC, P142-143, Chantilly,Virginia,USA, May 3-4, 2001
6. Feustel, Mayfield: The DGSA: Unmet Information Security Challenges for Operating System Designers, Operating Systems Review, ACM SIGOP, January, 1998, p. 3-22
7. DARPA/ISO. 1997. Research Challenges in Operating System Security. http://www.ito.arpa.mil/Proceedings/OS_Security/challenges/challenges_long.html. July 2, 1997.
8. Saltzer,J H. and Schroeder,M. “The Protection and Control of Information Sharing in Computer Systems”, Proc of IEEE, Vol 63, No 9 (September 1975).
9. R. Sandhu, and P. Samarati, "Access Control: Principles and Practice", IEEE Computer, pp 40-48, Sept 1994.
10. M. Abrams, L. LaPadula, K. Eggers, and I. Olson. A Generalized Framework for Access Control: an Informal Description. In Proceedings of the 13th National Computer Security Conference, pages 134--143, Oct. 1990.
11. Leonard LaPadula, “Rule-Set Modelling of Trusted Computer System”, Essay 9 in: M.Abrams, S.Jajodia, H.Podell, “Information Security – An integrated Collection of Essays”, IEEE Computer Society Press, 1995.
12. Standard for Information Technology Portable Operating System Interface (POSIX) Part I: System Application Protgram Interface (API), Report 1003.1e, (April 1994).
13. Glenn Faden. “RBAC in UNIX Administration”, ACM SIGSAC, 1999.
14. Sun Microsystems.Inc, “Trusted Solaris 8 Operating Environment——a Technical Overview” 2000.
15. Peter Loscocco, Stephen Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. Technical report, NSA and NAI labs, Jan 2, 2001.
16. S. Osborn, R. Sandhu and Q. Munawer. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. ACM Transactions on Information and System Security, 3(2), 2000.
17. David F.Ferraiolo, Ravi Sandhu, Serban Gavrila, “A Proposed Standard for Role-Based Access Control”, December 18, 2000. http://csrc.ncsl.nist.gov/rbac/RBAC-std-draft.doc
18. J. P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, US Air Force,

October 1972. (Two volumes) .
19．T. Jaeger, F. Giraud, N. Islam, and J. Liedtke. A role-based access control model for protection domain derivation and management. In Proceedings of the Second ACM Role-Based Access Control Workshop, November 1997.
20．Trent Jaeger, Nayeen Islam, Rangachari Anand, Atul Prakash, and Jochen Liedtke. Flexible Control of Downloaded Executable Content. http://www.ibm.com/Java/education/ excontrol, 1997.
21．B. W. Lampson. Dynamic Protection Structures. In Proceedings of the AFIPS Fall Joint Computer Conference, pages 27--38, 1969.
22．Ravi Sandhu. Future Directions in Role-Based Access Control Models . http://www.list.gmu.edu/conference_papers.htm, 2001.
23．查义国,徐小岩,张毓森. 在 Web 上实现基于角色的访问控制. 计算机研究与发展, Vol 39, No 3, pp. 257 – 263.Mar 2002.
24．钟华,冯玉琳,姜洪安. 扩充角色层次关系及其应用. 软件学报, 2000,11(6):779~784.
25．李成锴,詹永照,茅兵,谢立. 基于角色的 CSCW 系统访问控制模型. 软件学报, 2000,11(7):931~937.
26．乔颖,须德,戴国忠. 一种基于角色访问控制(RBAC)的新模型及其实现机制. 计算机研究与发展, vol.37,No1,Jan 2000.
27．S.Qing,T.Okamoto,and J.Zhou. A Role-Based Access Control Model and Implementation for Data-Centric Enterprise Applications. ICICS 2001,LNCS 2229,pp.316-327,2001.
28．石文昌,孙玉芳,等. 安全 Linux 内核安全功能的设计与实现. 计算机研究与发展, Vol.38, No.10, 2001 年 10 月,pp.1255-1261.

An Operating System oriented RBAC model and It`s Implementation

SHAN Zhi-yong　SUN Yu-fang

(Institute of Software, The Chinese Academy of Sciences, Beijing 100080)

Abstract　In this paper, the construction and implementation of an operating system oriented RBAC model are discussed. Firstly, on the basis of RBAC96 model, a new RBAC model which is named “OSR” is presented and formalized, by adding executable-file component and subdividing permission component. Secondly, the OSR model is enforced in secure Linux kernel by the way of integrating GFAC method and Capability mechanism together.

Key words　operating system，RBAC，access control