安全操作系统基于ACL的自主访问控制机制的设计与实现*)

孙亚楠　石文昌　梁洪亮　孙玉芳
(中国科学院软件研究所　北京100080)

摘　要　自主访问控制机制是安全操作系统必不可少的安全机制之一，而传统的文件权限保护模式不能提供更细粒度的自主访问控制，无法满足越来越高的系统安全的要求。本文探讨了基于访问控制表的自主访问控制机制的设计与实现的主要思想，并提出了利用文件系统的扩展属性机制存储访问控制表以及将自主访问控制机制实现为一个基于LSM安全框架的可加载模块的方法。

关键词　自主访问控制，访问控制表，扩展属性，Linux安全模型

Design and Implementation of ACL Based Discretionary Access Control Mechanism in Secure Operating System

SUN Ya-Nan　SHI Wen-Chang　LIANG Hong-Liang　SUN Yu-Fang
(Institute of Software, Chinese Academy of Sciences, Beijing 100080)

Abstract　As the traditional file system permission model in linux is inadequate to provide fine grained discretionary access control and can not meet the need of strong system security. This paper focuses on the Access Control List (ACL) based Discretionary Access Control (DAC) Mechanim in secure Operating System and presents a methmod to put ACL on Extended Attribute(EA) mechanism and to implement DAC mechanism as a loadable kernel module with the hooks inserted into kernel by the Linux Security Module(LSM).

Keywords　DAC(Discretionary Access Control), ACL(Access Control List), EA(Extended Attribute), LSM(Linux Security Module)

1　引言

操作系统中的所有活动都可看作是主体对客体的一系列操作，一个安全操作系统必须具备良好的访问控制机制来控制主体对客体的活动。自主访问控制(DAC)机制是安全操作系统必不可少的安全机制之一，其基本原理是在身份鉴别机制的基础上，由客体属主决定某一主体对文件、目录等客体的访问权限，权限一经确定，将作为以后判断该主体对客体是否具有以及具有什么权限的有效依据。传统的DAC机制，即"属主/同组用户/其他用户"文件权限保护模式将系统资源的访问权限划分为读、写、执行/搜索，指定给属主、同组用户和其他用户三类，没有精确到属主以外的其他任一主体，粒度较粗，不能满足实际应用的安全要求。访问控制表(ACL)是实现DAC的一个有效方法，因此国内外的安全操作系统，如Trusted Xenix、Trusted BSD、IRIX等，都采用ACL提供更细粒度和更灵活方便的自主访问控制，客体属主可以借助ACL给系统中任意主体或主体组指定自己拥有的任意权限，也可以限制系统中任意主体或主体组不能以任何方式访问客体。

本文讨论我们研制的安全操作系统SECIMOS中DAC机制的设计与实现。该机制利用内核中的扩展属性机制(EA)存储ACL，并在内核中安插钩子函数实现资源访问控制，根据用户指定方式或默认方式，阻止非授权主体访问客体，并控制访问权限扩散。访问控制的粒度是单一主体，没有访问权的主体只允许由授权主体指定对客体的访问权。我们的设计以国家标准GB17859-1999"计算机信息系统安全保护等级划分准则"第四级要求[1]为依据。

2　自主访问控制机制及其设计

2.1　访问控制矩阵模型

访问控制机制的原理可以用图1的访问控制矩阵M表示，在矩阵M中，S是主体集合，O是客体集合，R_{ij}表示主体S_i对客体O_j的访问模式[2]。

Subjects\objects		O_j	
S_i		R_{ij}	

图1　访问控制矩阵M

为了实现良好的自主访问控制机制，由访问控制矩阵M提供的信息必须以某种形式存放在系统中。目前，访问控制矩阵都不是被完整地存储，因为矩阵中的许多元素常常为空，空元素将会造成存储空间的浪费，且查找某个元素会耗费很多时间，因此在实现时通常基于矩阵的行或列来表达访问控制信息。ACL就是基于列的机制，它利用为客体提供一个主体明细表的方法来表示访问控制矩阵M。

2.2　访问控制表的语法规则

ACL的语法规则兼容POSIX.1e+2c标准[3]，每个ACL由一组ACL规则组成，用来设置单一主体或一组主体对客体的访问权限，规则的外部形式是"[default:]规则类型标签:规则限制符:权限"，如表1所示。为了可以同时对所有主体进行设置，本文讨论的SECIMOS系统的DAC机制对ACL进行扩充，增加了ACL_ALL和ACL_NONE规则。一个有效

*)国家863高技术研究发展项目(2002AA141080)和国家自然科学基金项目(60073022)资助。**孙亚楠**　硕士生，主要研究方向为系统软件与计算机安全。**石文昌**　研究员，博士，主要研究方向为系统软件与计算机安全。**梁洪亮**　博士，主要方向为系统软件与计算机安全。**孙玉芳**　研究员，博导，主要研究方向为系统软件与中文信息。

ACL 必须包含 ACL_USER_OBJ、ACL_GROUP_OBJ 和 ACL_OTHER 三条基本规则，若包含 ACL_USER 或 ACL_GROUP 规则，则必须具有一条 ACL_MASK 规则，其余规则可选。若一个 ACL 同时包含 ACL_ACLL 和 ACL_NONE 规则，则两者的权限集不能相交。ACL 从作用上分为 Access ACL 和 Default ACL 两种类型，前者确定客体访问权限，后者只适用于目录类型的客体，用于确定在目录下创建新客体时，新客体继承父目录的权限，不直接用于访问裁决。

表1 ACL 规则的类型

类型标签	外部形式(类型标签:规则限制符:权限)	规则含义
ACL_USER_OBJ	[default:]user::permissions	客体属主对客体的权限
ACL_USER	[default:]user:uid 或 user-name:permissions	某个用户对客体的权限
ACL_GROUP_OBJ	[default:]group::permissions	属主所在组对客体的权限
ACL_GROUP	[default:]group:gid 或 user-name:permissions	某个用户组对客体的权限
ACL_MASK	[default:]mask::permissions	限制除了客体属主和 other 之外的特定用户和用户组的最大权限
ACL_OTHER	[default:]other::permissions	其他用户对客体的权限
ACL_ALL	[default:]all::permissions	所有用户对客体都具有的权限
ACL_NONE	[default:]none::permissions	所有用户对客体都不具有的权限

2.3 访问控制表的客体与权限细化及访问检查算法

ACL 的主体为进程或用户，客体细化为文件、目录、设备、符号连接、系统进程、进程间通信等类型。其中对文件、目录和设备类型，访问控制细化到具体单个客体，对符号连接、系统进程和进程间通信类型的客体则提供整体的控制。主体对客体的权限由原来的读、写、执行扩充为读、写、执行、搜索、创建、删除、硬连接、截断、追加、重命名、改变所有者或组、改变目录、读属性、修改属性、关闭、克隆、终止、发信号、获取状态数据等。

由于 DAC 是由资源的属主决定谁可以对资源进行何种访问，在实际应用中，某些关键资源的属主可能对系统和安全没有深入了解，较难准确地设置资源的安全属性。因此本文讨论的 DAC 机制必须尽可能地弥补这一点，为各类客体规定了最大权限集，防止操作人员的误操作，如表2所示。

当一个用户进程访问一个客体时，将进程的有效 UID、GID 等用户标识信息和请求访问方式(mode)与客体的 Access ACL 中的规则进行比较，决定是否赋予其相应的访问权限。仅当一条匹配的 ACL 规则的权限集包含所有请求的访问权限时，才允许进程对客体的访问[4]。当用户进程请求搜索一条路径，进程必须具有路径名中每一个目录分量的搜索权。ACL 的访问检查算法逻辑描述如图2所示。

表2 客体的 ACL 最大权限集

客体	客体的最大权限集
普通文件	读、写、创建、执行、删除、硬连接、截断、追加、重命名、改变所有者或组、改变目录、读属性、修改属性、关闭
目录	搜索、创建、删除、重命名、改变所有者或组、读属性、修改属性
设备	读、写、读属性、修改属性、符号连接、改变所有者或组、关闭
系统进程	创建、克隆、发信号、终止、改变所有者或组、获取状态数据
符号连接	由符号连接目标的最大权限集所决定
进程通信	创建、删除、读属性、修改属性、读、写、关闭

```
if(客体 Access ACL 包含 ACL_ALL 规则) 且 (该规则的权限集 包含 进程请求的权限)
  then Access is granted
  else {
    if(客体 Access ACL 包含 ACL_NONE 规则) 且 (该规则的权限集 包含 进程请求的权限)
      then Access is denied
      else {
        if(进程的有效 UID 匹配 客体属主的 UID)
          then {
            if(ACL_USER_OBJ 规则的权限集 包含 进程请求的权限)
              then Access is granted
              else Access is denied
          }
          else {
            if (客体 Access ACL 包含 ACL_USER 规则)且(进程的有效 UID 匹配
                其中一条 ACL_USER 规则的限制符)
              then {
                if(该 ACL_USER 规则的权限集包含 进程请求的权限)且
                    (ACL_MASK 规则的权限集 包含 进程请求的权限)
                  then Access is granted
                  else Access is denied
              }
              else {
                if(进程的有效 GID 匹配 客体属主的 GID)
                  then {
                    if(ACL_USER_OBJ 规则的权限集 包含 进程请求的权限)
                      then Access is granted
                      else Access is denied
                  }
                  else {
                    if (客体 Access ACL 包含 ACL_GROUP 规则)且(进程
                        的有效 GID 匹配 其中一条 ACL_GROUP 规则的限制符)
                      then {
                        if(该 ACL_GROUP 规则的权限集包含 进程请求的权限)
                            且(ACL_MASK 规则的权限集 包含 进程请求的权限)
                          then Access is granted
                          else Access is denied
                      }
                      else {
                        if (ACL_OTHER 规则中的权限 包含 进程请求的权限)
                          then Access is granted
                          else Access is denied
                      }
                  }
              }
          }
      }
  }
```

图2 访问检查算法伪代码

2.4 访问控制表的初始化及继承机制

创建一个客体时，由属主、超级用户或授权用户决定是否为它初始化 Access ACL。首先确定各类客体的上级客体。文件/目录类型的客体的上级客体就是父目录，可以一直追溯到

根(/)目录,/目录的上级客体指定为该类型的 Default 客体。设备类型的客体的上级客体是该类型的 Default 客体。对于 IPC 和系统进程的客体,系统初始默认只有 Default 客体。各类客体的 Default 客体是整个此类客体的抽象,具有不会随着客体的变化而变化的 Default ACL[5]。

当用 *creat*()、*mkdir*()、*mknod*()、*mkfifo*()和 *open*()(有 *O_CREAT* 参数)等函数新建一个文件时,它的 Access ACL 按如下顺序计算:

1. 若它的上级客体具有 Default ACL,则继承这个 Default ACL 作为它的初始 Access ACL。再修改这个 Access ACL 中与文件权限位模式对应的 ACL_USER_OBJ、ACL_GROUP_OBJ 和 ACL_OTHER 规则,去除创建进程的 *mode* 参数中没有指定的权限,得到新客体的 Access ACL。

2. 如果上级客体没有 Default ACL,则由创建进程的 *mode* 参数和文件创建掩码 *umask* 一起决定新客体的 Access ACL。新客体的 ACL_USER_OBJ、ACL_GROUP_OBJ、和 ACL_OTHER 规则的权限由文件创建掩码决定,去除创建进程的 *mode* 参数中没有指定的权限,得到新客体的 Access ACL。

由于对访问权限进行了细化,为了更好地实现权限继承机制,引入权限屏蔽值 *mask* 的概念,其每一位对应一项权限,分别用0/1表示。权限屏蔽值 *mask* 一般默认为全1。如果没有具体指明主体对客体(文件/目录、符号连接、设备)的权限,则主体可以继承它对当前客体的上级客体的 Default ACL,若不想继承,就将相应的权限位设为0,反之为1。当客体被创建后,可以设置它的权限屏蔽值。将权限屏蔽值与继承上级客体的 Default ACL 的权限项相与,作为当前主体对当前客体的权限,并覆盖 Default ACL 相应的权限项。

3 实现方法

3.1 访问控制表的存储与传递

要实现访问控制表,关键问题之一是访问信息的存放,需保证其机密性、完整性和可用性。由于访问控制信息一般都很短小,因此可以作为扩展属性存储,并通过预分配好大小的缓冲区在内核和用户空间传递。为一个客体增加一个 ACL 时,就为之分配一个单独的扩展属性块,记录着(*name*:*value*)对。在 Ext2/Ext3文件系统中每个 *i* 节点结构中都有一个预留域 *i_file_acl*,指向存储与这个 *i* 节点关联的 ACL 信息的扩展属性块。因此只需一次磁盘访问就可以检索到 *i* 节点的扩展属性。为了保证信息的完整性,一个 inode 的所有扩展属性一般存储在一个扩展属性块中[4,6]。

为了提高效率,若多个 *i* 节点的 ACL 扩展属性都相同,这些 *i* 节点就可以共享相同的扩展属性块。通过计算扩展属性块的哈希值可以检测到相同块并对该块进行缓冲。当需要产生一个新块时,会先在缓冲中寻找。如果找到同样的块,重用该块(只需要增加磁盘引用计数,最大为1024),否则分配一个新块。如果一个指向共享的 EA 块的 inode 和它的 ACL 属性名和值被修改了,且没有被引用,则采用覆盖写的机制进行更新。

当一个文件系统中的客体第一次被访问时,它的访问控制表从 *i* 节点和扩展属性块中转换成能被快速处理的内存格式。由于对 ACL 访问会很频繁,当沿着路径在嵌套很深的目录中访问 ACL 时尤其如此,因此为了加快权限检查和文件创建,我们对 ACL 采用缓存方式。

3.2 访问控制表与文件权限模式的协调

Linux 的文件权限保护模式是在每个文件上附加一段有关存取信息的二进制位,缺点是客体的属主不能精确控制单个主体对客体的访问权,优点是非常简单、高效,在某些应用上起到一定的作用。因此将权限位保护模式与 ACL 共存于系统中,文件权限位模式的 *owner*/*group*/*other* 分别映射到 ACL_USER_OBJ、ACL_GROUP_OBJ(若 ACL 有 ACL_MASK 则映射到 ACL_MASK)、ACL_OTHER,保持完全同步。ACL 的处理函数兼容文件权限模式的处理函数,如 *chmod*()、*creat*()、*stat*()等。

当要进行访问权限检查时,采用一定的算法使二者协调工作。Trusted Xenix 安全操作系统采用的算法是首先判断要访问的客体是否具有 ACL,若有,进行 ACL 权限检查,若无,则检查文件权限保护模式[7]。本文的自主访问控制机制是基于 LSM 安全模型框架上实现,我们采用与 Trusted Xenix 不同的算法。当要决定主体对客体是否拥有某种权限时,首先判断主体请求是否满足系统对客体的文件权限保护模式,满足则允许主体进行请求的操作;若不满足,则检查该客体是否具有访问控制表,若有再判断是否满足访问控制表机制,若满足,允许主体进行请求的操作,反之拒绝请求。

3.3 自主访问控制机制的基本框架

本文讨论的 SECIMOS 安全操作系统的 DAC 机制的实现目标是基于 LSM 安全模型框架[8],通过完善钩子函数,实现对资源的访问控制,减小对内核的改动,能不加修改地运行现有应用程序,并且使得系统开销应尽可能减小。DAC 机制的基本框架如图3所示。

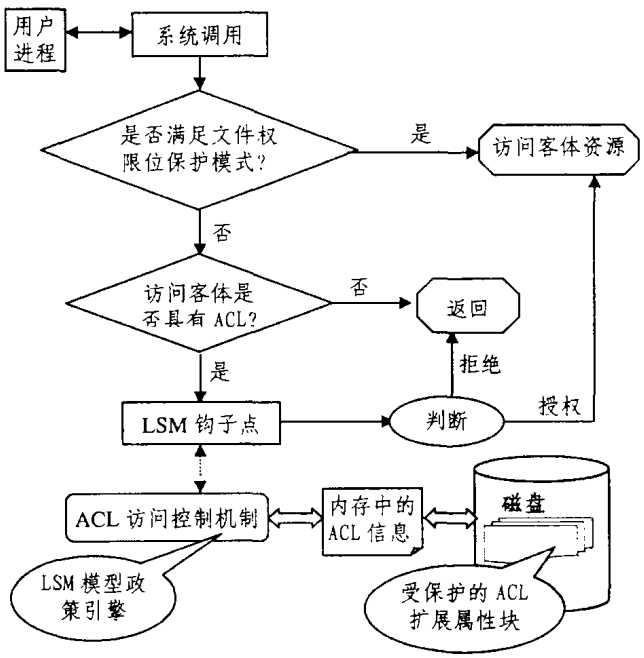

图3 DAC 机制基本框架

在 DAC 中,将 ACL 作为 EA 在用户空间和内核空间传递、在 VFS 和更底层的文件系统层之间传递,这样简化了系统接口。扩展属性是根据名字来识别的,比如"*system.posix_ACL_Access*"和"*system.posix_ACL_Default*"分别代表了客体的 ACL 和 Default ACL,ACL 属性值以一种体系结构无关的格式存储。遵循 Posix.1e 标准的应用程序通过调用 libACL 库实现兼容,该库实现了 posix.1e 标准定义的 ACL 处理函数。ACL 的权限检查通过在系统调用中插入钩子函数来实现,当关闭 ACL 机制时,钩子返回 NULL。

(下转第162页)

5 UA 设计

UA 驻留于客户端，主要为用户应用系统提供接入服务，并且作为接收方还要随时接收从 MC 发送的其它用户的交换数据。UA 的设计要考虑跨平台和可移植的要求，不能受到用户应用系统运行平台的限制。EasySwitch 采用 XML-RPC 作为接口层基本的服务调用方式，最大限度地降低了数据中间件的 API 层的访问难度，同时保证了平台独立性要求。

5.1 UA 和用户应用系统的交互模式

UA 屏蔽了 EasySwitch 内部的复杂结构和运行过程，向企业应用系统展现出一个单一的服务接口。同时，UA 还能适应用户环境的变化，比如当用户 IP 地址发生变化时，UA 能够即时发送更新信息到 MC。UA 和用户应用系统之间的交换模式如图4所示。

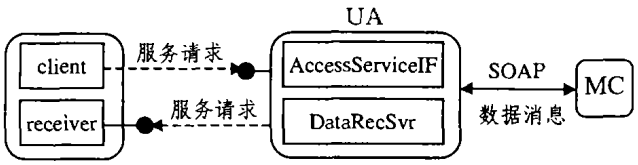

图4 UA 和用户应用系统交互模式示意图

1)AccessServiceIF 是 UA 的接入服务接口，它为用户应用系统中的 client 方提供包括数据交换在内的所有服务调用。

2)DataRecSvr 是 UA 中的数据接收服务器，它随时准备接收从 MC 发送的其它企业的交换数据消息。

3)receiver 是用户应用系统提供的数据接收服务接口，用户必须在安装 UA 时定义 receiver 的位置信息。当 DataRecSvr 收到数据消息后，从配置文件中读取 receiver 的信息，然后通过动态调用方式生成 XML-RPC 服务请求并发送到 receiver 中，完成一次数据交换过程。

5.2 基于 XML-RPC 服务接口的实现

UA 中的 AccessServiceIF 和用户应用系统提供的 receiver 都是基于 XML-RPC 方式定义的服务接口，它通过 WSDL 文档来描述其服务的名称及参数类型，调用指令和返回数据都是以标准的 SOAP 封装传送的。UA 方的 AccessServiceIF 是采用 JAX-RPC 来开发的，JAX-RPC 提供了基于 SOAP 的 RPC 调用运行环境，其核心是实现了标准 XMLSchema 数据类型和 Java 数据类型之间的映射。AccessServiceIF 的 Java 定义如下：

```
public interface AccessServiceIF extends Remote {
  public abstract Source sendData(Source sourceData) throws RemoteException;
  public abstract Source receiveData(String memberId) throws RemoteException;
  public abstract SchemaInfo [] getSchemaInfo(String memberId) throws RemoteException;
  public abstract ErrorInfo [] getErrorInfo() throws RemoteException;
}
```

总结 为适应各企业应用系统间进行数据交换协作的需求，解决异构系统集成困难，本文基于行业内交流数据的相似性和相关性理论，通过行业数据标准的设立和数据模式的节点分解构造出可重构的数据交换核心，并在此基础上通过采用 SOAP，XML-RPC 等标准协议规范开发了应用集成框架，从而完整地实现了一个异构系统基于数据交换的协作平台 EasySwitch。通过对数据交换核心的重构可使 EasySwitch 适用于不同领域，比如证券系统间交易数据的交换，电子政务系统间的公文流转等。本文所实现的 EasySwitch 系统现已初步应用于重庆市摩托车电子商务平台，是一个典型的行业内企业数据交换的应用。需要指出的是，EasySwitch 不仅可用于行业内相关企业进行数据交换集成，对于企业内部各个子系统进行数据协作也能很好地支持。

根据企业实际的业务流程框架及信息技术的发展状况可以把企业集成分为数据集成和服务集成。基于数据处理的商务过程，对于自主运行的应用系统来说，在收到某一类型的业务数据后能够触发内部处理流程，EasySwitch 对于这种基于数据交换的应用协作提供了完整的解决方案。如何扩展 EasySwitch 的应用集成框架，实现对企业间基于功能服务的集成是下一步研究的内容。

参 考 文 献

1 Gabrick K A, Weiss D B. J2EE and XML Development, Manning Publication, 2002
2 SOAP Version1.2 Specification. http://www.w3.org/TR/2003/PR-soap12-part1-20030507/. http://www.w3.org/TR/2003/PR-soap12-part2-20030507/
3 JAXRPC-1.0 Specification, Sun Microsystems, 2002
4 SAAJ1.1 Specification, Sun Microsystems, 2002
5 Myerson J M. The Complete Book of Middleware. Auerbach Publications, 2002

(上接第155页)

3.4 性能影响分析

如果将 ACL 信息集中存放在文件中，要访问 ACL，在文件最近未被访问过的前提下，需要至少两次磁盘访问，而通过文件系统的扩展属性机制存放 ACL，利用文件系统 inode 的缓冲机制，可以减少访问磁盘的次数。且扩展属性块可以被多次引用，提高了效率。

结束语 本文探讨了安全操作系统 SECIMOS 中基于访问控制表的自主访问控制机制的设计与实现的主要思想，并讨论了 ACL 如何与传统 linux 的文件权限保护模式在系统中协调工作的问题。本文所提出的 DAC 机制不仅对授权主体细化到单个用户，并且对权限和系统可操作的客体类型也加以细化，针对不同类型的客体给出客体的权限范围，完善了访问权限检查算法。提出了权限屏蔽值和 Default 客体的概念，使我们的安全操作系统 SECIMOS 的 DAC 机制能够灵活地支持实际应用中的 DAC 要求。

参 考 文 献

1 中国国家标准-计算机信息系统安全保护等级划分准则 GB17859-1999
2 石文昌，孙玉芳. 安全操作系统研究的发展(上). 计算机科学，2002，(29)6
3 IEEE Draft P1003.1e/2c IEEE Standard Department, 1997
4 Grunbacher A. Posix Access Control Lists on Linux, SuSE Labs, linux AG
5 红旗安全操作系统技术白皮书. http://www.redflag-linux.com/security/document/whitepaper.pdf
6 Gildfind A. Access Control lists and Extended attributes on linux, SGI
7 Final evaluation report. Trusted Xenix v3.0, National Computer Security center
8 Morris J, et al. Linux Security Modules: General Security Support for the Linux Kernel. http://www.intercode.com.au/jamesm/lsm-usenix-html/lsm-html.html, 2002