# 安全操作系统中基于客体的保护机制

金 雷 林志强 茅 兵 谢 立
(南京大学软件新技术国家重点实验室 南京210093)

## Security Policies Based on Object in Security-Enhanced Operating System

JIN Lei LIN Zhi-Qiang MAO Bing XIE Li
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093)

**Abstract** Security of operation system is the basis of protecting computer system against attack . To resolve more and more problem in security area , we need an operation system of great security. That require we find an effective method to develop an security-enhanced operation system to meet these needs. Access control is often used in modern operation system. It is based on identity affirm and enforces control to the resources that are required by the identify. In this paper we mainly discuss security policies based on object (Mac and Dac).

**Keywords** Operating system, Subject, Object, DAC, MAC

操作系统的安全是所有计算机系统安全的基础。访问控制是计算机保护尤其是操作系统保护中极其重要的一环。在访问控制中，对其访问必须进行控制的资源称为客体，同理，必须控制它对客体的访问的活动资源称为主体。主体即访问的发起者，通常为进程程序或用户。客体包括各种资源，如文件，设备，信号量等。访问控制中第三个元素是保护规则，它定义了主体与客体可能的相互作用途径。

## 1. 访问控制的机制

这里引入保护域的概念。每一主体(进程)都在一特定的保护域下工作。保护域规定了进程可以访问的资源。每一域定义了一组客体及可以对客体采取的操作。可对客体操作的能力称为访问权(Access Right)，访问权定义为有序对的形式。一个域是访问权的集合。如域 A 有读写权，那在域 A 下运行的进程可对域中的客体 A 执行读写，但不能执行任何其他的操作。保护域并不是彼此独立的。它们可以有交叉，即它们可以共享权限，如图1所示。

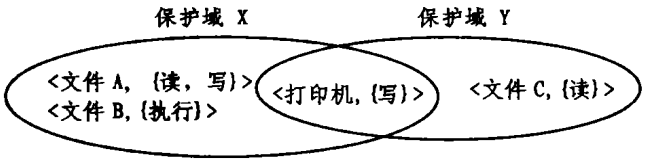


图1

主体(进程)在某一特定时刻可以访问的资源(软件，硬件)的集合称为客体。一般客体的保护机制有两种。一种是自主访问控制(discretionary access control)，一种是强制访问控制(mandatory access control)。

所谓的自主访问控制是一种最为普遍的访问控制手段，用户可以按自己的意愿对系统的参数做适当修改以决定哪些用户可以访问他们的文件。

所谓强制访问控制是指用户与文件都有一个固定的安全属性。系统用该安全属性来决定一个用户是否可以访问某个文件。安全属性是强制性的规定，它是由安全管理员，或者是操作系统根据限定的规则确定的，用户或用户的程序不能加以修改。如果系统认为具有某一个安全属性的用户不适于访问某个文件，那么任何人(包括文件的拥有者)都无法使该用户具有访问该文件的权力。

## 2. 自主访问控制

一个安全操作系统的访问控制机制要基于对主体及主体所属的主体组的识别，来限制对客体的访问，还要校验主体对客体的访问请求是否符合存取控制规定来决定对客体访问的执行与否。这里所谓的自主访问控制是指主体可以自主地将访问权，或访问权的某个子集授予其它主体。

为了实现完备的自主访问控制系统，由访问控制矩阵提供的信息必须以某种形式存放在系统中。访问矩阵中的每行表示一个主体，每一列则表示一个受保护的客体，而矩阵中的元素，则表示主体可以对客体的访问模式。目前，在系统中访问控制矩阵本身，都不是完整地存储起来，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，而且查找某个元素会耗费很多时间。实际上常常是基于矩阵的行或列来表达访问控制信息。

### 2.1 基于行的自主访问控制

所谓基于行的自主访问控制是在每个主体上都附加一个该主体可访问的客体的明细表。基于行的自主访问控制主要是依靠权限字、前缀表的方式进行实现。

2.1.1 权限字 是一个提供主体对客体具有某种特定权限的不可伪造标志。主体可以建立新的客体，并指定这些客体上允许的操作。它作为一张凭证，允许主体对某一客体完成特定类型的访问。仅在用户通过操作系统发出特定请求时才建立权限字，每个权限字标识可允许的访问，例如，用户可以创建文件、数据段、子进程等新客体，并指定它可接受的操作种类(读、写或执行)，也可以定义新的访问类型(如授权、传递

金 雷 硕士研究生，研究方向为网络安全及系统安全。林志强 硕士研究生，研究方向为网络安全及系统安全。茅 兵 教授，研究方向为人机交互、分布计算和并行处理系统安全。谢 立 博导，教授，研究方向为分布式计算和系统安全。

等)。

具有转移或传播权限的主体 A 可以将其权限字的副本传递给 B,B 也可将权限字传递给 C,但为了防止权限字的进一步扩散,B 在传递权限字副本给 C 时可移去其中的转移权限,于是 C 将不能继续传递权限字。权限字也是一种程序运行期间直接跟踪主体对客体的访问权限的方法。一个进程具有自己运行时的作用域,即访问的客体集,如程序、文件、数据、I/O 设备等,当运行进程调用子过程时,它可以将访问的某些客体作为参数传递给子过程,而子过程的作用域不一定与调用它的进程相同。即调用进程仅将其客体的一部分或全部访问传递给子过程,子过程也拥有自己能够访问的其他客体。由于每个权限字都标识了作用域中的单个客体,因此权限字的集合就定义了作用域。进程调用子过程并传递特定客体或权限字时,操作系统形成一个当前进程的所有权限字组成的堆栈,并为子过程建立新的权限字。权限字也可以集成在系统的一张综合表中(如存储取控制表),每次进程请求都由操作系统检查该客体是否可访问,若可访问,则为其建立权限字。

权限字必须存放在内存中不能被普通用户访问的地方,如系统保留区、专用区或者被保护区域内,在程序运行期间,只有被当前进程访问的客体的权限字能够很快得到,这种限制提高了对访问客体权限字检查的速度。由于权限字可以被收回,操作系统必须保证能够跟踪应当删除的权限字,彻底予以回收,并删除那些不再活跃的用户的权限字。

在安全操作系统中,使用的不再是一个简单的 setuid 系统,而是构造权限字模式。所用的权限字类似于 setuid,但是给予了非常仔细的细化。一个可执行文本可以被标记,从而获得一指定特权,而不是或者全是、或者全不是的 setuid 系统。

在 Linux 中,为了支持这一模式,如下代码被作了修改:

```
If (suser()) {
  /* Do some privileged operation. */
}
```

被修改成:

```
If (capable(CAP_DAC_OVERRIDE)){
  /* Do directory access override */
}
```

当采用权限字表时,只需动态地对进程的权限字做修改,不需要特权状态,减少了滥用权利的风险。必须注意的是,一个进程必须不能直接改动它的权限字表,如果可以,则它可能给自己增加没有权利访问的资源权限。

2.1.2 *前缀表(prefixes)* 包含受保护的客体名及主体对它的访问权限。当系统中有某个主体欲访问某个客体时,访问控制机制将检查主体的前缀是否具有它所请求的访问权。但是这种方式有以下三个问题:前缀大小有限制;当生成一个新客体或者改变某个客体的访问权时,如何对主体分配访问权;如何决定可访问某客体的所有主体。

由于客体名通常是杂乱无章的,很难进行分类,而且当一个主体可以访问很多客体时,它的前缀也将是非常大的,因而也很难于管理。还有受保护的客体必须具有唯一的名字,互相不能重名,故而造成客体名数目过大。另外,在一个客体生成、撤消或改变访问权时,可能会涉及许多主体前缀的更新,因此需要进行许多操作。当用户生成新客体并对自己及其它用户授予对此客体的访问权时,相应的前缀修改操作必须用安全的方式完成,不应由用户直接修改。有的系统由系统管理员来承担。还有的系统由安全管理员来控制主体前缀的更改。但是这种方法也很不便,特别是当一个频繁更迭对客体访问权的情况,更加不适用。访问权的撤消一般也很困难,除非对每种访问权系统都能自动校验主体的前缀。删除一个客体时,需要判断在哪些主体前缀中有该客体。

### 2.2 基于列的自主访问控制

所谓基于列的访问控制是指按客体附加一份可访问它的主体的明细表。基于列的访问控制可以有两种方式:

2.2.1 *保护位* 这种方式不能完备地表达访问控制矩阵。UNIX 系统采用了此方法。保护位对所有的主体、主体组(用户、用户组)以及该客体(文件)的拥有者,规定了一个访问模式的集合。用户组是具有相似特点的用户集合。生成客体的主体称为该客体的拥有者。它对客体的所有权仅能通过超级用户特权来改变。拥有者(超级用户除外)是唯一能够改变客体保护位的主体。一个用户可能不只属于一个用户组,但是在某个时刻,一个用户只能属于一个活动的用户组。用户组及拥有者名都体现在保护位中。

2.2.2 *存取控制表* 可以决定任何一个特定的主体是否可对某一个客体进行访问。它是利用在客体上附加一个主体明细表的方法来表示访问控制矩阵的。表中的每一项包括主体的身份以及对该客体的访问权。例如,对某文件的存取控制表,可以存放在该文件的文件说明中,通常包含有对此文件的用户的身份,文件主或是用户组,以及文件主或用户组成员对此文件的访问权限。如果采用用户组或通配符的概念,这一存取控制信息表不会很长。

目前,存取控制表方式是自主访问控制实现中比较好的一种方法。

### 2.3 自主访问控制的访问许可

我们把客体的控制与对客体的访问区别开来。

由于访问许可允许主体修改客体的存取控制表,因此利用它可以实现对自主访问控制机制的控制。这种控制有三种类型:

2.3.1 *等级型* 可以将对客体存取控制表的修改能力划分成等级。例如,可以将控制关系组成一个树型结构。系统管理员的等级设为等级树的根,根一级具有修改所有客体存取控制表的能力,并且具有向任意一个主体分配这种修改权的能力。系统管理员可以按部门将工作人员分成多个子集,并对部门领导授与相应存取控制表的修改权和对修改权的分配权。部门领导又可将自己部门内的人员分成若干个组,并且对组级领导授与相应的对存取控制表的修改权。在树中的最低级的主体不再具有访问许可,也就是说他们对相应的客体的存取控制表不再具有修改权。有访问许可的主体(即有能力修改客体的存取控制表),可以对自己授与任何访问模式的访问权。

这种结构的优点是,通过选择可信任的人担任各级领导,使得能够以可信方式对客体施加控制。并且这种控制和人员的组织体系相近似。缺点是,对于一个客体而言,可能会同时有多个主体有能力修改它的存取控制表。

2.3.2 *拥有型* 另一种控制方式是对每个客体设立一个拥有者(通常是该客体的生成者)。只有拥有者才是对客体有修改权的唯一主体。拥有者对其拥有的客体具有全部控制权。但是,拥有者无权将其对客体的控制权分配给其它主体。因此,客体拥有者在任何时候都可以改变其所属客体的存取控制表,并可以对其它主体授予或者撤消其对客体的任何一种访问模式。

系统管理员应能够对系统进行某种设置,使得每个主体

都有一个"主目录"(home directory)。对主目录下的子目录及文件的访问许可权应授予该主目录的主人。使他能够修改主目录下的客体的存取控制表,但不允许使拥有者具有分配这种访问许可权的权力。可以把拥有型控制看成是二级的树型控制。在 UNIX 系统中,利用超级用户来实施特权控制,就是这样的一个例子。

2.3.3 自由型 自由型方案的特点是:一个客体的生成者可以对任何一个主体分配对它拥有的客体的访问控制权,即对客体的存取控制表有修改权,并且还可使其对主体也具有分配这种权力的能力。在这种系统中,不存在"拥有者"概念。例如,一旦一个主体 A 将修改其客体存取控制表的权力与分配这种权力的能力授予了主体 B,那么主体 B 就可以将这种能力分配给其它主体,而不必征求客体生成者的同意。这样,一旦访问许可权分配出去,那么就很难控制客体了。虽然可以从客体的存取控制表中查出所有能修改者的名字,但是却没有任何主体能对该客体的安全负责。

## 3 强制访问控制

自主访问控制是保护系统资源不被非法访问的一种有效手段。但是由于它的控制是自主的,也带来了问题。于是,人们又提出了一种更强有力的访问控制手段,这就是强制访问控制。

在自主访问控制方式中,某一合法用户可以任意运行一程序来修改他拥有的文件存取控制信息,而操作系统无法区分这种修改是用户自己的操作,还是恶意攻击的特洛伊木马的非法操作。

通过强加一些不可逾越的访问限制,系统可以防止某一些类型的特洛伊木马的攻击。在强制访问控制方式中,系统对主体和客体都分配一个特殊的安全属性,而且这一属性一般不能更改,系统通过比较主体和客体的安全属性来决定一个主体是否能够访问某个客体。用户的程序不能改变他自己及任何其它客体的安全属性。强制访问控制还可以阻止某个进程共享文件,并阻止通过一个共享文件向其它进程传递信息。

强制访问控制施加给用户自己客体的严格的限制,但也使用户受到自己的限制。但是,系统为了防范特洛伊木马,必须要这么做。即便是不再存在特洛伊木马,强制访问控制也有用,它可以防止在用户无意或不负责任的操作时,泄露机密信息。强制访问控制对专用的或简单的系统是有效的,但对通用、大型系统并不那么有效。一般强制访问控制采用以下几种方法:

① 限制访问控制。自主控制方式允许用户程序修改他拥有文件的存取控制表,这为非法者带来可乘之机。因而,系统可以不提供这一方便,在这类系统中,用户要修改存取控制表的唯一途径是请求一个特权系统调用。该调用的功能是依据用户终端输入的信息,而不是靠另一个程序提供的信息来修改存取控制信息。

② 过程控制。在通常的计算机系统中,只要系统允许用户自己编程,就没办法杜绝特洛伊木马。但可以对其过程采取某些措施,这种方法称为过程控制。例如,警告用户不要运行系统目录以外的任何程序。提醒用户注意,如果偶然调用一个其它目录的文件时,不要做任何动作,等等。需要说明的一点是,这些限制取决于用户本身执行与否。因而,自愿的限制很容易变成实际上没有限制。

③ 系统限制。显然,实施的限制最好是由系统自动完成。要对系统的功能实施一些限制。比如,限制共享文件,但共享文件是计算机系统的优点,所以是不可能加以完全限制的。再者,就是限制用户编程。事实上,有许多不需编程的系统都是这样做的。

不过这种做法只适用于某些专用系统。在大型的通用系统中,编程能力是不可能去除的。在网络中也不行,在网络中一个没有编程能力的系统可能会接收另一个具有编程能力的系统发出的程序。有编程能力的网络系统可以对进入系统的所有路径进行分析,并采取一定措施。这样就可以增加特洛伊木马攻击的难度。

采用 MAC 结构需要将安全策略贯穿于系统中所有的主体和客体,由包含各种安全相关信息的标签来决定访问控制。这能使系统有效地保护自己,并能对应用程序的安全提供强大的支持。它能建立安全的进程管道,使各种应用能安全地执行其他非可靠的应用程序。由于 MAC 限制了执行进程的特权,它使受攻击的系统服务和应用程序的潜在危险限定在一个范围内。MAC 通过对合法用户授权来保护信息,即使授权用户无意中运行了恶意程序,MAC 也可以保护数据。这些保护系统的能力对构造一个安全的系统是必需的。

当前 Linux 建立的自主访问控制(DAC)与 MAC 是两个并行的概念,DAC 并不会因为 MAC 的加入而得到加强。仅仅依靠用户标志和所有权来进行访问控制是远远不够的。我们还必须考虑与安全相关的其他因素:如用户角色,程序的信任度和功能,数据的敏感性和完整性。若用户对客体具有完全的自主性,那么对数据流的控制和在系统范围内进行安全策略的加强是不可能的。

## 参 考 文 献


1 The Limits of Formal Security Models National Computer Systems Security Award Acceptance Speech Dorothy E. Denning Oct. 1999

2 石文昌. 安全操作系统研究的发展. 计算机科学,2002,29(6~7)

3 茅兵. 基于 Linux 的安全操作系统的开发. 2000 年中国自由软件发展战略研讨会暨第一届中国自由软件应用论坛会刊,中国国家高技术智能计算机系统专家组与中国共创软件联盟主办,北京,2000

4 A Security Policy Configuration for the Security-Enhanced Linux Stephen Smalley, NAI Labs, Timothy Fraser, NAI Labs, Feb. 2001

5 Linux Intrusion Detection System. http://www.lids.org