安全操作系统研究的发展(上)*)

The Development of Research on Secure Operating Systems (1)

石文昌　孙玉芳

(中国科学院软件研究所　北京 100080)

Abstract Prior to the work of this paper, our knowledge of the evolution history of secure operating systems was limited and primarily anecdotal. This paper proposes a classification method which divides the progress process of the research on secure operating systems into foundation period, cookbook period, multi-policy period and dynamic-policy period. The originating and developing procedure of the fundamental concepts, technologies and methods of secure operating systems is analyzed systematically according to the proposed classification method. As a result, a comprehensive perspective of the evolution course of the research on secure operating systems is presented. The discussion of the topic in question consists of two parts. This is the first part.

Keywords Secure operating system, Evolution history, Development period, Multi-policy, Dynamic-policy

1 引言

对安全操作系统的技术发展历史进行全面的概括总结,是一项比较艰巨的工作,在本文的工作之前,这方面的已有信息是零碎的、不全面的[1]。本文通过对安全操作系统研究与开发方面三十几年来比较有代表性的工作的深入考察,根据这些研究与开发工作及其相应的技术成果的特点,提出了四阶段划分的思想,给出了奠基时期、食谱时期、多政策时期和动态政策时期的定义,并以此为基础,以史实的形式,归纳和描绘了安全操作系统的发展演化历程,力求为确定正确的安全操作系统开发方法建立良好的基础。

2 发展阶段划分方法

本文以时间为线索,根据安全操作系统研究与开发工作的特点,进行阶段划分。考察的对象是从第一个安全操作系统起至本文写作之时止安全操作系统研究与开发中的典型工作。

首先,有必要确定安全操作系统研究的起始年代。不少学者认为,安全操作系统的研究与开发工作是从安全 Multics 系统的研究与开发开始的,作者对这一观点有不同的看法。

实际上,安全操作系统的研究与开发工作比安全 Multics 系统的研究与开发工作要早若干年。世界上的第一个安全操作系统是 Adept-50[2~5],这是一个分时系统。安全 Adept-50 的工作始于六十年代中后期[4],而安全 Multics 的工作始于七十年代初期[6]。

作者把安全操作系统研究的历程划分为奠基时期、食谱时期、多政策时期和动态政策时期等四个发展阶段。奠基时期始于 1967 年 Adept-50 项目的启动之时,在这个时期,安全操作系统经历了从无到有的探索过程,安全操作系统的基本思想、理论、技术和方法逐步建立。食谱时期始于 1983 年美国的 TCSEC[7]标准颁布之时,这个时期的特点是人们以 TCSEC 为蓝本研制安全操作系统。多政策时期始于 1993 年,这个时期的特点是人们超越 TCSEC 的范围,在安全操作系统中实现多种安全政策。动态政策时期始于 1999 年,这个时期的特点是使安全操作系统支持多种安全政策的动态变化,实现安全政策的多样性。

3 奠基时期

1967 年,计算机资源共享系统的安全控制问题引起了美国国防部的高度重视,国防科学部旗下的计算机安全特别部队的组建拉开了操作系统安全研究的序幕[8,9]。

3.1 萌芽与访问控制抽象

1969 年,C. Weissman 发表了有关 Adept-50 的安全控制的研究成果[2]。Adept-50 是历史上的第一个安全操作系统,可以实际投入使用,运行于 IBM/360 硬件平台,它以一个形式化的安全模型—高水标模型为基础,实现了美国的一个军事安全系统模型,为给定的安全问题提供了一个比较形式化的解决方案。在该系统中,可以为客体标上敏感级别属性。系统支持的基本安全条件是,对于读操作,不允许信息的敏感级别高于用户的安全级别。对于写操作,在授权情况下,允许使信息从高敏感级别移向低敏感级别。

同年,B. W. Lampson 通过形式化表示方法,运用主体、客体和访问矩阵的思想第一次对访问控制问题进行了抽象[10]。主体是访问操作中的主动实体,客体是访问操作中的被动实体,即,主体对客体进行访问。访问矩阵是以主体为行索引、以客体为列索引的矩阵,矩阵中的每一个元素表示一组访问方式,是若干访问方式的集合。矩阵中第 i 行第 j 列的元素 M_{ij} 记录着第 i 个主体 S_i 可以执行的对第 j 个客体 O_j 的访问方式,比如 M_{ij} 等于{read, write}表示 S_i 可以对 O_j 进行读和写访问。

1970 年,W. H. Ware 推出的研究报告[8]对多渠道访问的资源共享的计算机系统引起的安全问题进行了研究。报告结合实际的国防信息安全等级划分体制,分析了资源共享系统中敏感信息可能受到的安全威胁,提出了解决计算机安全问题的建议途径。

报告研究的主要目标是多级安全系统在计算机中的实

*)国家自然科学基金项目(60073022)、国家 863 高科技项目(863-306-ZD12-14-2)和中国科学院知识创新工程项目(KGCX1-09)资助。石文昌 博士,研究员,主要研究方向为系统软件与计算机安全。孙玉芳 研究员,博士生导师,主要研究方向为系统软件和中文信息处理。

现。报告指出，安全级别和该知权限是多级安全问题中的重要成分，基本的多级安全问题就是要确定具有特定安全级别和该知权限的个体是否能够访问给定物理环境中的某个范围的敏感信息。报告对计算机安全系统的设计提出了两个限制条件：

(1)计算机安全系统必须与现实的安全等级划分结构一致；

(2)计算机安全系统必须与现实的手工安全控制规程相符。

报告建议的计算机安全系统涉及到系统灵活性(在应用中可调整个体和信息的安全等级)、可靠性(贯彻"失败-保险"思想，当不能确定是否授权时，采取不授权的措施)、可审计性(记录安全相关行为)、可管理性(安全控制、审计控制等管理)、可依赖性(避免拒绝对用户的服务)、配置完整性(确保系统自身的完整)等特点。报告讨论了在存储资源管理方面避免遗留信息泄漏问题。报告认为，计算机系统的安全控制是一个系统设计问题，必须从硬件、软件、通信、物理、人员和行政管理规程等各个方面综合考虑。报告还给出了访问控制问题的形式化描述。

3.2 引用监控机和安全核

1972 年，作为承担美国空军的一项计算机安全规划研究任务的研究成果，J. P. Anderson 在一份研究报告[11]中提出了引用监控机、引用验证机制、安全核和安全建模等重要思想。这些思想是在研究系统资源受控共享问题的背景下产生的。

把授权机制与能够对程序的运行加以控制的系统环境结合在一起，可以对受控共享提供支持，授权机制负责确定用户(程序)对系统资源(数据、程序、设备等)的引用许可权，程序运行控制负责把用户程序对资源的引用控制在授权的范围之内。这一思想可以形象地表示为图 1 的形式。

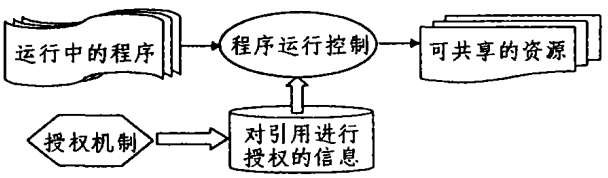

图 1　系统资源的受控共享

引用监控机思想是为了解决用户程序的运行控制问题而引入的，其目的是在用户(程序)与系统资源之间实施一种授权的访问关系。Anderson 把引用监控机的职能定义为：以主体(用户等)所获得的引用权限为基准，验证运行中的程序(对程序、数据、设备等)的所有引用。对应到图 1，引用监控机是在"程序运行控制"的位置上发挥作用的。

引用监控机是一个抽象的概念，它表现的是一种思想。Anderson 把引用监控机的具体实现称为引用验证机制，它是实现引用监控机思想的硬件和软件的组合。引用验证机制需要同时满足以下三个原则：A)必须具有自我保护能力；B)必须总是处于活跃状态；C)必须设计得足够小，以利于分析和测试，从而能够证明它的实现是正确的。第一个原则保证引用验证机制即使受到攻击也能保持自身的完整性。第二个原则保证程序对资源的所有引用都得到引用验证机制的仲裁。第三个原则保证引用验证机制的实现是正确的和符合要求的。

在受控共享和引用监控机思想的基础上，Anderson 定义了安全核的概念。安全核是系统中与安全性的实现有关的部分，包括引用验证机制、访问控制机制、授权机制和授权的管理机制等成分。

Anderson 指出，要开发安全系统，首先必须建立系统的安全模型。安全模型给出安全系统的形式化定义，正确地综合系统的各类因素。这些因素包括：系统的使用方式、使用环境类型、授权的定义、共享的客体(系统资源)、共享的类型和受控共享思想等。这些因素应构成安全系统的形式化抽象描述，使得系统可以被证明是完整的、反映真实环境的、逻辑上能够实现程序的受控执行的。完成安全系统的建模之后，再进行安全核的设计与实现。

3.3 隐通道与 BLP 模型

1973 年，B. W. Lampson 通过对程序的禁闭问题的研究提出了隐通道的概念[12]。研究的背景是程序的调用与信息的传送，设程序 B 是由程序 A 调用运行的，所谓对程序 B 的禁闭就是指，制止程序 B 在运行期间向其它程序(程序 A 除外)传送信息。程序 B 向它的拥有者传送有关程序 A 的信息属于信息泄漏。Lampson 在归纳这种情形的信息泄漏的各种渠道时，把隐通道定义为，按常规不会用于传送信息但却被利用于泄漏信息的信息传送渠道。比如，一个程序对系统负载的影响，似乎与信息传送渠道无关，但该程序可以通过改变其对系统负载的影响，利用对系统负载的影响的变化情况来向另一个程序暗示某种信息，这种信息泄漏的渠道就属于隐通道。

同年，D. E. Bell 和 L. J. LaPadula 提出了第一个可证明的安全系统的数学模型[13,14]，这就是 Bell&LaPadula 模型，简称 BLP 模型。在随后的几年，该模型得到了进一步的充实和完善[15~17]。Bell 和 LaPadula 在 1976 年完成的研究报告[17]给出了 BLP 模型的最完整表述，其中包含模型的形式化描述和非形式化说明，以及模型在 Multics 系统中实现的解释。

BLP 模型是根据军方的安全政策设计的，它要解决的本质问题是对具有密级划分的信息的访问进行控制。

BLP 模型是一个状态机模型，它定义的系统包含一个初始状态 z_0 和由一些三元组(请求，判定，状态)组成的序列，三元组序列中相邻状态之间满足某种关系 W。如果一个系统的初始状态是安全的，并且三元组序列中的所有状态都是安全的，那么这样的系统就是一个安全系统。

BLP 模型定义的状态是一个四元组(b, M, f, H)，其中，b 是当前访问的集合，当前访问由三元组(主体，客体，访问属性)表示，是当前状态下允许的访问；M 是访问控制矩阵；f 是安全级别函数，用于确定任意主体和客体的安全级别；H 是客体间的层次关系。抽象出的访问属性有四种，分别是只可读$\underline{r}$、只可写$\underline{a}$、可读写$\underline{w}$和不可读写(可执行)$\underline{e}$。主体的安全级别包括最大安全级别和当前安全级别，最大安全级别通常简称为安全级别。以下特性和定理构成了 BLP 模型的核心内容。

简单安全特性(ss-特性)：如果(主体，客体，可读)是当前访问，那么一定有：level(主体)≥ level(客体)。其中，level 表示安全级别。

星号安全特性(*-特性)：在任意状态，如果(主体，客体，属性)是当前访问，那么一定有：

(1)若属性是$\underline{a}$，则：*level*(客体)≥*current-level*(主体)；

(2)若属性是$\underline{w}$，则：*level*(客体)=*current-level*(主体)；

(3)若属性是$\underline{r}$，则：*current-level*(主体)≥*level*(客体)；

其中，*current-level* 表示当前安全级别。

自主安全特性(ds-特性)：如果(主体-i，客体-j，属性-$\underline{x}$)是当前访问，那么，属性-$\underline{x}$一定在访问控制矩阵 M 的元素 M_{ij}

中。

与 ds-特性处理自主访问控制相对应，ss-特性和 *-特性处理的是强制访问控制。自主访问控制的权限由客体的属主自主确定，强制访问控制的权限由特定的安全管理员确定，由系统强制实施。

基本安全定理：如果系统状态的每一次变化都能满足 ss-特性、*-特性和 ds-特性的要求，那么，在系统的整个状态变化过程中，系统的安全性是不会被破坏的。

BLP 模型支持的是信息的保密性。继 BLP 模型之后，K. J. Biba 提出了与 BLP 模型同曲异工的 Biba 模型[18~21]，Biba 模型支持的是信息的完整性。

3.4 保护机制结构与设计原则

1975 年，J. H. Saltzer 和 M. D. Schroeder 以保护机制的体系结构为中心，探讨了计算机系统的信息保护问题[3]，重点考察了权能实现结构和访问控制表实现结构，给出了信息保护机制的八条设计原则。

为讨论信息保护问题，从概念上，可以为每一个需保护的客体建立一个不可攻破的保护墙，保护墙上留有一个门，门前有一个卫兵，所有对客体的访问都首先在门前接受卫兵的检查。在整个系统中，有很多客体，因而有很多保护墙和卫兵。对客体的访问控制机制的实现结构可分为两种类型：面向门票的实现和面向名单的实现。在面向门票的实现中，卫兵手中持有一份对一个客体的描述，在访问活动中，主体携带一张门票，门票上有一个客体的标识和可访问的方式，卫兵把主体所持门票中的客体标识与自己手中的客体标识进行对比，以确定是否允许访问；在整个系统中，一个主体可能持有多张门票。在面向名单的实现中，卫兵手中持有一份所有授权主体的名单及相应的访问方式，在访问活动中，主体出示自己的身份标识，卫兵从名单中进行查找，检查主体是否记录在名单上，以确定是否允许访问。

权能结构属于面向门票的结构，一张门票也称作一个权能。访问控制表(ACL)结构属于面向名单的结构。在访问控制矩阵的概念模式下，权能结构对应访问控制矩阵的行结构，行中的每个矩阵元素对应一个权能；ACL 结构对应访问控制矩阵中的列结构，每一列对应一个 ACL。

Saltzer 和 Schroeder 给出的信息保护机制的设计原则是：

1)机制经济性原则：保护机制应设计得尽可能的简单和短小。有些设计和实现错误可能产生意想不到的访问途径，而这些错误在常规使用中是察觉不出的，难免需要进行诸如软件逐行排查工作，简单而短小的设计是这类工作成功的关键。

2)失败-保险默认原则：访问判定应建立在显式授权而不是隐式授权的基础上，显式授权指定的是主体该有的权限，隐式授权指定的是主体不该有的权限。在默认情况下，没有明确授权的访问方式，应该视作不允许的访问方式，如果主体欲以该方式进行访问，结果将是失败，这对于系统来说是保险的。

3)完全仲裁原则：对每一个客体的每一次访问都必须经过检查，以确认是否已经得到授权。

4)开放式设计原则：不应该把保护机制的抗攻击能力建立在设计的保密性的基础之上。应该在设计公开的环境中设法增强保护机制的防御能力。

5)特权分离原则：为一项特权划分出多个决定因素，仅当所有决定因素均具备时，才能行使该项特权。正如一个保险箱设有两把钥匙，由两个人掌管，仅当两个人都提供钥匙时，保险箱才能打开。

6)最小特权原则：分配给系统中的每一个程序和每一个用户的特权应该是它们完成工作所必须享有的特权的最小集合。

7)最少公共机制原则：把由两个以上用户共用和被所有用户依赖的机制的数量减少到最小。每一个共享机制都是一条潜在的用户间的信息通路，要谨慎设计，避免无意中破坏安全性。应证明为所有用户服务的机制能满足每一个用户的要求。

8)心理可接受性原则：为使用户习以为常地、自动地正确运用保护机制，把用户界面设计得易于使用是根本。

Saltzer 和 Schroeder 也指出，如何证明硬件和软件保护机制的设计与实现的正确性，是一项已吸引很多注意力的研究课题。

3.5 操作系统保护理论

1976 年，M. A. Harrison，W. L. Ruzzo 和 J. D. Ullman 提出了操作系统保护的第一个基本理论[22]，本文称其为 HRU 理论。HRU 理论形式化地给出保护系统模型的定义，并通过三个定理给出有关保护系统的一些结果。Harrison 等还用该模型对 Unix 系统[23]的保护系统进行了刻画。

定义 HRU1 一个保护系统由以下两个部分构成：

(1)一个以普通权限为元素的有限集合 R；

(2)一个以命令为元素的有限集合 C，命令的格式为：

```
command α(X1,X2,…,Xk)
    if r1 in (Xs1,Xo1) and
      r2 in (Xs2, Xo2) and
        …
      rm in (Xsm, Xom)
    then
    op1, op2,… opn
end
```

或者，如果 m 为0，命令格式为：

```
command α(X1, X2,… Xk)
      op1, op2,…,opn
end
```

其中，α 是命令名，$X_1,\cdots,X_k$ 是形式参数，每个 op_i 是以下原语操作之一：

```
enter r into (Xs, Xo)
delete r from (Xs, Xo)
create subject Xs
create object Xo
destory subject Xs
destory object Xo
```

$r, r_1,\cdots r_m$ 是普通权限，$s, s_1,\cdots,s_m$ 和 $o, o_1,\cdots, o_m$ 是1至 k 间的整数。

定义 HRU2 一个保护系统的一个配置是一个三元组 (S, O, P)，其中，S 是当前主体的集合，O 是当前客体的集合，$S\subseteq O$，P 是访问矩阵，$P[s, o]$是 R 的子集，是主体 s 对客体 o 的权限的集合。

定义 HRU3 设 (S, O, P)和(S', O', P')是一个保护系统的两个配置，op 是原语操作，以下式子表示(S, O, P)在操作 op 下转换成(S', O', P')：$(S, O, P)\Rightarrow_{op}(S', O', P')$。

定义 HRU4 设 $Q=(S, O, P)$是一个保护系统的一个配置，保护系统包含以下命令：

```
command α(X1, X2,…, Xk)
    if r1 in (Xs1, Xo1) and
        …
      rm in (Xsm, Xom)
    then op1, op2,…, opn
end
```

定义配置 Q' 如下：

(1)如果命令 α 的条件得不到满足，则定义 Q' 为 Q；

(2)如果命令 α 的条件得到满足,则设存在配置 $Q_0, Q_1, \cdots, Q_n$,使得:$Q=Q_0 \Rightarrow_{op1} Q_1 \Rightarrow_{op2} \cdots \Rightarrow_{opn} Q_n$。其中,$op_i'$ 表示用实际参数 $x_1, x_2, \cdots, x_k$ 分别代替形式参数 $X_1, X_2, \cdots, X_k$ 后得到的原语操作 op_i 的对应表示。定义 Q' 为 Q_n。

Q 与 Q' 的关系表示为:$Q \vdash \alpha(x_1, x_2, \cdots, x_k)\ Q'$

如果存在参数 $x_1, x_2, \cdots, x_k$,使得 $Q \vdash \alpha(x_1, x_2, \cdots, x_k)\ Q'$,则说 $Q \vdash \alpha Q'$;如果存在命令 α,使得 $Q \vdash \alpha Q'$,则说 $Q \vdash Q'$。用 $\vdash^*$ 表示 $\vdash$ 的零次或多次应用,则 $Q \vdash^* Q'$ 表示 Q 经过零次或多次 $\vdash$ 运算后得到 Q'。

定义 HRU5 给定一个保护系统,$\alpha(X_1, X_2, \cdots, X_k)$ 是一个命令,r 是一个普通权限,$Q=(S,O,P)$ 是一个配置,如果在 Q 上运行时,命令 α 能执行一个原语操作,把 r 放入访问矩阵中原来不包含 r 的一个元素中,则说命令 α 从配置 Q 中泄露权限 r。

定义 HRU6 给定一个保护系统和一个普通权限 r,Q_0 是初始配置,如果存在一个配置 Q 和一个命令 α,使得:(1)$Q_0 \vdash^* Q$,并且(2)α 从 Q 中泄露 r,则说 Q_0 对 r 是不安全的。如果 Q_0 对 r 不是不安全的,则说 Q_0 对 r 是安全的。

定义 HRU7 如果一个保护系统的每一个命令的解释都是一个单一的原语操作,则说这个保护系统是单一操作的。

定理 HRU1 存在一个算法,确定一个给定的单一操作的保护系统和初始配置对一个给定的普通权限 r 是否是不安全的。

定理 HRU2 一个给定的保护系统的一个给定的配置对一个给定的普通权限是否是安全的,这是不可确定的。

定理 HRU3 没有 create 命令的保护系统的安全性问题在多项式空间中是完全的。

3.6 系统设计和开发

继 Adept-50之后,特别是 Anderson 的报告之后,越来越多的安全操作系统项目相继被启动,一系列的安全操作系统被设计和开发出来[4],典型的有 Multics[6,24,17]、Mitre 安全核[25,26]、UCLA 数据安全 Unix[27,28]、KSOS[29,30]和 PSOS[31,32]等。

原始的 Multics 操作系统[33]虽然没有把安全性列入设计目标,但保护功能的设计是一个重点。最重要的创新是保护环和访问控制表。访问控制表控制用户对文件(段)的访问。系统中设若干个保护环,在同一个保护环上,特权相同,保护环由里向外,特权逐级降低。保护环结构把两状态(超级用户和普通用户)机器的思想推广到 n 状态机器,每一个状态对应一个保护环,在 Honeywell DPS 8/70M 系统上的实现支持 n=8。原始 Multics 中几乎所有的特权软件都在最里层保护环上。原始 Multics 的开发对安全模型和验证没有考虑,但微妙的保护结构和层次化的设计使得它成为一个比较好的增加安全性控制的基础。

安全增强的 Multics 把 BLP 模型应用到 Multics 系统中,成为第一个支持 BLP 模型的系统。安全增强的 Multics 系统没有把安全相关的代码分离为安全核。Honeywell 和 Mitre 对能否把 Multics 操作系统重新设计成安全核的形式进行了研究,MIT 对 Multics 超级用户的研究发现大幅度减少运行于最里层保护环的代码量是可能的,后来由于经费原因,始终没有 Multics 安全核诞生。

Mitre 安全核是基于 BLP 模型为 PDP-11机器开发的安全核原型,性能比较差。Mitre 安全核的开发提供了系统的顶层和低层描述,并进行了手工验证,验证了内核的操作和各描述层之间的一致性。为建立 Unix 仿真环境,Mitre 安全核被进行重新开发,目的是要获得更适合 Unix 要求的功能。

UCLA 数据安全 Unix 是为 PDP-11机器开发的提供 Unix 用户界面的安全核原型。该项目的任务由两大部分组成:(1)开发能够降低安全判定和安全实施软件的规模和复杂性的系统体系结构;(2)通过形式化验证方法证明系统能够满足安全性要求。

系统的安全相关软件由核心、政策管理器、对话器、调度器和应用进程等若干分离的组件构成。除了核心组件外,其他组件都是外层进程。政策管理器负责安全判定,它实现对 BLP 模型的支持。系统的验证工作借助了 XIVUS 系统[34]的定理证明器。系统验证完成了从顶层描述到系统代码的机器辅助证明各个步骤的验证工作,但没有验证完核心的所有部分。

KSOS 项目的目标是为 PDP-11/70机器开发一个可投放市场的安全操作系统,系统的要求是:(1)与 Bell 实验室的 Unix 系统兼容;(2)实现多级安全性和完整;(3)正确性可以被证明。

项目起初由 Ford 航空航天与通信公司承担,开发了一个安全核原型和一个 Unix 系统仿真环境。项目的开发提供了形式化的顶层描述和验证,代码与描述间的一致性证明由手工完成。系统后来由 Logicon 公司进行改造,把核心做得更小、更快,并开发一套接口包取代原来的仿真环境。系统的改造保持形式化描述的一致性。

PSOS 是安全操作系统的一个设计项目,它提供一个层次结构化的基于权能的操作系统设计,设计中的每一个层次管理一个特定类型的对象,系统中的每一个对象通过该对象的权能表示进行访问。PSOS 基于层次式开发方法,通过形式化技术实现对安全操作系统的描述和验证。

与安全操作系统设计紧密相关,1976年,T. A. Linden 讨论了结构化设计技术对操作系统安全性的影响[35],重点论述了小保护域和扩展类型的客体这两个支持安全性的系统结构化思想。小保护域可以实现程序模块级的控制,它允许一个程序中的模块运行在受到控制的环境中,防止模块的行为对程序的其它部分造成不良的影响。扩展类型的客体提供一种数据抽象的机制,允许应用程序定义新的客体类型并为新类型创建新客体,新类型的客体意味着新的客体操作,把系统的保护特性扩展到面向应用的客体操作。扩展类型的客体既支持自主访问控制,也支持强制访问控制。小保护域和扩展类型的客体的思想通过基于权能思想的方法可以得到有效的实现。

4 食谱时期

从无到有,在探索如何研制安全计算机系统的同时,人们也在研究着如何建立评价标准去衡量计算机系统的安全性。第一个计算机安全评价标准的诞生,把安全操作系统研究带入了一个新的阶段。

4.1 第一个计算机安全评价标准

1983年,美国国防部颁布了历史上第一个计算机安全评价标准,这就是著名的可信计算机系统评价标准[2],简称 TCSEC,又称橙皮书。1985年,美国国防部对 TCSEC 进行了修订[9]。

TCSEC 标准是在基于安全核技术的安全操作系统研究的基础上制定出来的,标准中使用的可信计算基(TCB)就是安全核研究结果的表现。

1979年，G. H. Nibaldi 在描述一个基于安全核的计算机安全系统的设计方法时给出了 TCB 的定义[36]，该方法要求把计算机系统中所有与安全保护有关的功能找出来，并把它们与系统中的其它功能分离开，然后把它们独立出来，以防止遭到破坏，这样独立出来得到的结果就称为 TCB。Nibaldi 在同年提交的计算机安全评价建议标准[37]中运用了 TCB 的思想。

TCB 在 TCSEC 中的定义是：一个计算机系统中的保护机制的全体，它们共同负责实施一个安全政策，它们包括硬件、固件和软件；一个 TCB 由一个产品或系统上共同实施一个统一的安全政策的一个或多个组件构成。安全核在 TCSEC 中的定义是：一个 TCB 中实现引用监控机思想的硬件、固件和软件成分；它必须仲裁所有访问、必须保护自身免受修改、必须能被验证是正确的。

TCSEC 提供 D、C1、C2、B1、B2、B3和 A1七个等级的可信系统评价标准，每个等级对应有确定的安全特性需求和保障需求，高等级的需求建立在低等级的需求的基础之上。

4.2 LINUS IV 系统的开发

1984年，AXIOM 技术公司的 S. Kramer 发表了 LINUS IV 系统的设计与开发成果[38]。LINUS IV 是 Unix 类的实验型安全操作系统。传统的 Unix 系统虽然提供一定的保护机制，但安全性不是它的设计目标[39]。LINUS IV 以4.1BSD Unix 为原型，结合 TCSEC 标准的要求[7]，对安全性进行了改造和扩充。

4.2.1 *身份鉴别* 修改了口令生成方案，把加密的口令从/etc/passwd 文件中分离出来，存放在受保护的/etc/protpasswd 独立文件中。

4.2.2 *自主访问控制* 通过 ACL 结构实现自主访问控制，在 ACL 中除提供 read、write、execute 权限外，还提供 noaccess 权限。传统的 user/group/others 访问控制方式依然有效，但实际上它是通过映射成相应的 ACL 访问控制方式来实现的。

4.2.3 *强制访问控制* 实现 BLP 模型，用户、进程是主体，文件是客体（所有主存和外设都以文件形式实现）。用户在 login 时指定安全等级（不能高于用户的最高安全等级），在整个会话过程中，安全等级不能改变。只有安全管理员能够改变文件的安全等级。

通过隐藏子目录的方法实现对/tmp 等共享目录的支持。进程要访问共享目录时，系统在共享目录下为之建立对应的隐藏子目录，隐藏子目录的安全等级与进程的安全等级相等，进程实际访问到的是隐藏子目录，而不是表面上的共享目录。隐藏子目录的存在是动态的，访问前建立，访问后删除。隐藏子目录的存在对于用户来说是透明的，他们只看到传统意义上的共享目录。

4.2.4 *安全审计* 提供基于事件的严重性程度的审计支持。严重性程度用于对该监控和不该监控的系统活动进行划分。安全管理员可动态设定审计的级别，让系统检测所有文件的打开、关闭、创建、删除操作，或让系统只检测对最敏感的数据的访问和对最特权的操作的启动。

4.2.5 *超级用户特权的分隔* 把传统的超级用户划分为安全管理员、系统操作员、系统管理员三种特权用户，安全管理员行使诸如设定安全等级、管理审计信息等系统安全维护职能，系统操作员行使诸如磁盘数据备份、启动和关闭系统等系统日常维护职能，系统管理员行使诸如创建和删除用户帐户、处理记帐信息等系统管理职能。传统的超级用户不复存在。

4.3 安全 Xenix 系统的开发

1986和1987年，IBM 公司的 V. D. Gligor 等发表了安全 Xenix 系统的设计与开发成果[40~42]。安全 Xenix 是以 Xenix 为原型的实验型安全操作系统，属于 Unix 类的安全操作系统，它要实现的是 TCSEC 标准 B2-A1级的安全要求。

4.3.1 *系统开发方法和保障目标* Gligor 等把 Unix 类的安全操作系统的开发方法划分为仿真法和改造/增强法两种方式[40]。按照仿真法开发的 Unix 类安全操作系统由一个安全核、一组可信进程和一个 Unix 仿真环境组成。Unix 仿真环境由一组可信程序或进程组成。安全核实现引用监控机的功能，向上提供一个小型操作系统的接口。Unix 仿真环境和可信进程在安全核之上构成另一个软件层，它们实现 Unix 的安全政策和提供 Unix 系统接口。

改造/增强法对 Unix 系统的原有内核进行改造和扩充，使它支持新的安全政策和 Unix 的安全政策，并保持相应的系统接口。安全 Xenix 的开发采用这种方法。

在安全保障方面，安全 Xenix 项目重点考虑实现以下目标：(1)系统设计与 BLP 模型之间的一致性；(2)实现的安全功能的测试；(3)软件配置管理工具的开发。

4.3.2 *访问控制方式* 按照 BLP 模型实现访问控制，主体是进程，客体是进程、文件、特别文件（设备）、目录、管道、信号量、共享内存段和消息。访问权限有 read、write、execute 和 null。

在自主访问控制中，传统 Xenix 的 owner/group/others 访问控制机制与 ACL 访问控制机制并存，对于确定的客体，只能有一种机制起作用，由用户确定使用传统 Xenix 机制或 ACL 机制。

由强制访问控制带来的诸如/tmp 等共享目录的访问问题，通过隐藏子目录的方法解决，隐藏子目录在用户 login 时由系统建立。

4.3.3 *安全等级的确定* 安全 Xenix 的强制访问控制的安全判断以进程的当前安全等级、客体的安全等级和访问权限为依据。进程的当前安全等级（CPL）在创建进程时确定，在进程的整个生存期内保持不变。login 时生成的会话进程的 CPL 值由用户在 login 过程中根据需要自行指定，但必须满足以下条件：

$$System\_Low \leqslant CPL \leqslant PML$$

其中，System_Low 是系统的最小安全等级，由安全管理员赋值。PML 是进程的最大安全等级，由下式确定：

$$PML = greatest\_lower\_bound(UML, GML, TML)$$

greatest_lower_bound 是最大下界函数。UML、GML 和 TML 分别是用户、用户组和终端的最大安全等级，由安全管理员赋值。

其他进程的 CPL 值等于其父进程的 CPL 值。

设备的安全等级由安全管理员指定。文件、管道、目录、信号量、消息和共享内存段等的安全等级等于创建它们的进程的 CPL 值。目录的安全等级也可以在创建时指定，但其值必须在创建它的进程的 CPL 和 PML 之间。

4.3.4 *安全注意键与可信路径* 安全 Xenix 的可信通路是以安全注意键（SAK）为基础实现的。SAK 是由终端驱动程序检测到的键的一个特殊组合。每当系统识别到用户在一个终端上键入的 SAK，便终止对应到该终端的所有用户进

程，启动可信的会话过程。

4.3.5 *特权用户* 对系统的管理功能进行分割，设立可信系统程序员、系统安全管理员、系统帐户管理员和系统安全审计员等特权用户，通过不同的操作环境和操作界面限定各特权用户的特权。

可信系统程序员具有超级用户的全部特权，但只能在系统的维护模式下工作，负责整个系统的配置和维护。系统安全管理员负责管理系统中有关的安全属性，如安全等级等。系统帐户管理员负责用户帐户的设立、撤消及相关的管理工作。系统安全审计员负责配置和维护审计机制和审计信息。

4.4 System V/MLS 系统的开发

1988年，AT&T Bell 实验室的 C. W. Flink II 和 J. D. Weiss 发表了 System V/MLS 系统的设计与开发成果[11]。System V/MLS 是以 AT&T 的 Unix System V 为原型的多级安全操作系统，以 TCSEC 标准的安全等级 B 为设计目标。

4.4.1 *BLP 模型在 Unix System V 中的解释* 系统按照 BLP 模型提供多级安全性支持，主体是进程，客体包括文件、目录、i-节点、进程间通信(IPC)结构和进程。BLP 模型的 ss-特性和 *-特定在 Unix System V 中的映射如表1所示。

表1 BLP 模型在 Unix System V 中的映射

操 作	条 件
Read/Search/Execute	label(S)≥label(O)
Write(Overwrite/Append)	*label*(S)=*label*(O)
Create/Link/Unlink	*label*(S)=*label*(Od)
Read-status	*label*(S)≥*label*(O)
Change-status	*label*(S)=*label*(O)
Read-ipc	*lable*(S)≥*label*(O)
Write-ipc	*label*(S)=*label*(O)
Send-signal	*label*(S)=*label*(O)

其中，*label* 表示安全等级标记，*S* 表示主体，*O* 表示客体，*Od* 表示目录客体，实际上指被操作的客体所在的目录。

在强制访问控制下，共享目录问题采用隐藏子目录的方法解决。

4.4.2 *安全标记机制设计* Unix System V 提供 owner/group/others 方式的保护机制，支持用户组的概念，用户组由组标识(GID)表示。在系统中，每个主体的进程表中、每个客体的 *i*-节点中、每个 IPC 的数据结构中，都有一个 GID 域。为保持系统兼容性，不修改 Unix 的底层数据结构，System V/MLS 利用系统中的 GID 域表示强制访问控制的安全等级标记。为扩大安全等级标记和用户组的取值范围，采用间接标记方式，即，在 GID 域中存放的是索引，索引指向的数据结构存放安全等级标记、自主访问控制的 GID 和其它信息。

4.4.3 *多级安全性实现方法* 通过在内核中加入多级安全性(MLS)模块实现对多级安全性的支持。MLS 模块是内核中可删除、可替换的独立模块，负责解释安全等级标记的含义和多级安全性控制规则，实现强制访问控制判定。在原系统的与访问判定有关的内核函数中插入调用 MLS 模块的命令，实现原有内核机制与 MLS 机制的连接。

4.5 安全 TUNIS 系统的开发

1989年，加拿大多伦多大学的 G. L. Grenier、R. Holt 和 M. Funkenhauser 发表了安全 TUNIS 系统的设计与开发成果[44]。TUNIS 是加拿大多伦多大学开发的一个与 Unix 兼容的操作系统，是 Unix 内核的一个新的实现，该系统用强类型的 Turing Plus 高级语言编写，具有较好的模块化结构。安全 TUNIS 系统是以 TUNIS 系统为原型的安全操作系统。

Grenier 等指出，如果不进行系统的重新设计，以传统 Unix 系统为原型，很难开发出高于 TCSEC 标准的 B2安全等级的安全操作系统[44,45]，这一方面是因为用于编写 Unix 系统的 C 语言是一个非安全的语言，另一方面是因为 Unix 系统内部的模块化程度不够。Turing Plus 是一个可验证的语言，采用该语言编写的安全 TUNIS 系统的设计目标是 TCSEC 标准的 B3-A1等级。

安全 TUNIS 系统的可信计算基(TCB)由硬件、内核和可信进程等成份构成。安全 TUNIS 的设计借鉴了 Hydra 系统设计中的政策与机制分离的思想[46]，不同的是，Hydra 系统在内核中实现机制，在应用程序中实现政策，安全 TUNIS 系统把内核分割成两个部分，一部分实现安全政策，另一部分实现安全机制。

安全 TUNIS 实现改造过的 BLP 模型的安全政策，内核中实现安全政策的部分被设计成一个称为安全管理器的模块，安全管理器的职能是根据安全政策的规定执行安全判定。内核中除安全管理器以外的其它部分构成安全机制，安全管理器的工作需要安全机制提供的服务的支持。安全管理器与安全机制在安全 TUNIS 中的位置可用图2表示。

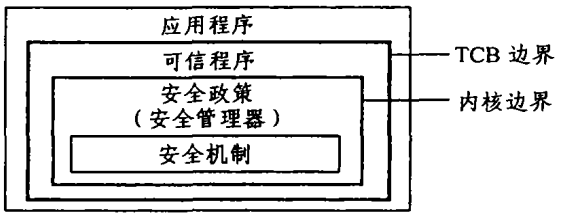

图2 安全 TUNIS 的安全管理器与安全机制

用户的所有请求都必须从安全管理器模块经过。安全管理器根据用户的请求确定应该启用的安全政策，借助安全机制检索出必要的安全信息，并利用安全信息进行安全判定，然后把通过判定的用户请求传递给安全机制执行。

安全政策与安全机制分离的方法，使得安全政策独立在一个小的安全管理器模块中，可以简化对安全管理器的形式化处理和验证过程，同时，也使得安全机制的描述与验证不受安全管理器的影响，从而最终可以简化 TCB 的验证工作，有利于高等级安全操作系统的实现。安全 TUNIS 原型系统的实现证明了这种分离设计方法的可行性，并提供了一个简单而结构规范的 TCB，它能满足 TCSEC 标准的 B3级要求。

4.6 ASOS 系统的开发

1990年，TRW 公司的 N. A. Waldhart 和 B. L. Di Vito 等发表了 ASOS 系统的设计与开发成果[47,48]。

ASOS 是针对美军的战术需要而设计的军用安全操作系统，由两类系统组成，其中一类是多级安全操作系统，设计目标是满足 TCSEC 标准的 A1级要求，另一类是专用安全操作系统，设计目标是满足 TCSEC 标准的 C2级要求。两类系统都支持 Ada 语言编写的实时战术应用程序，都能根据不同的战术应用需求进行配置，都具有容易在不同硬件平台间移植的特点。两类系统提供一致的用户界面。不但是主要针对基于 Ada 的应用而设计，ASOS 操作系统本身主要也是用 Ada 语言实现的。

ASOS 系统采用访问控制表(ACL)结构支持自主访问控制。对于每一个客体，ASOS 的 ACL 结构可以指定每一个用

户和用户组对它的访问权限，也可以明确指定给定用户或用户组不能对该客体进行访问。

ASOS 系统依据 BLP 模型实现防止信息泄露的强制访问控制，依据限制的 Biba 模型实现确保数据完整性的强制访问控制，限制的 Biba 模型规定，仅当主体的完整性等级支配客体的完整性等级时，允许主体写客体。ASOS 的强制访问控制除了支持安全性等级和完整性等级外，还支持访问等级，这里，访问等级是安全性等级和完整性等级的组合。

ASOS 项目在形式化验证中建立了两个层次的描述和证明，一个层次用于抽象的安全模型，一个层次用于形式化顶层描述。项目也提供了叙述式的顶层描述，以自然语言（普通英语）的形式描述安全核的接口。用于证明系统安全性的主要工具是 Gypsy 验证环境（GVE）[49]。项目还开发了一个在 GVE 中工作的流分析工具，用于分析系统设计中潜在的隐通道。

5 多政策时期

随着九十年代初 Internet 的影响的迅速扩大，分布式应用的迅速普及，单一安全政策的范型与安全政策多种多样的现实世界之间拉开了很大的差距。1993年，美国国防部在 TAFIM 计划中推出了新的安全体系结构 DGSA，DGSA 的显著特点之一是对多级安全政策支持的要求，这为安全操作系统的研究提出了新的挑战[50~52]，促使安全操作系统研究进入了一个新的时期。

5.1 国防部目标安全体系结构 DGSA

国防部的 TAFIM 计划为信息管理制定了一套技术上的体系结构框架，在国防部内强制实施，国防部目标安全体系结构 DGSA 是该体系结构框架的一个组成部分。在制定 DGSA 之前，国防部总结出了一组信息系统的安全需求，这组需求规定：

（1）国防部的信息系统必须支持多种安全政策下的信息处理，这些安全政策可以具有任意的复杂性和任意的类型，它们可能涉及非密级的敏感信息的处理和多种类别的密级信息的处理。

（2）国防部的信息系统必须受到充分的保护，以便能够在符合开放系统体系结构的多个网络的多个主机上进行分布式信息处理，包括进行分布式信息系统管理。

（3）国防部的信息系统必须支持在拥有不同的安全属性、按照不同的安全保护程度使用资源的用户之间进行信息处理，在必要的场合，这些用户可以包括那些使用非安全资源的用户。

（4）国防部的信息系统必须受到充分的保护，以便可以借助公共的（非军用的）通信系统进行连接。

以上这些规定是制定 DGSA 安全需求的基础。DGSA 的安全需求是一定层次上的抽象需求，这些需求包括：对多种信息安全政策的支持、开放系统的采用、充分的安全保护和共同的安全管理。

为了支持在一组用户间共享信息对象，同时，在一个共享信息的处理和通信环境中对这些信息对象进行充分的保护，DGSA 定义了信息域的概念。一个信息域由一些信息对象、一些用户和一个信息安全政策构成。支持多种安全政策的系统将拥有与安全政策数量一样多的信息域，这样的系统有能力处理数量非常多的信息域。

每一个信息对象落在一个信息域中，受所落信息域的信息安全政策控制。每个信息对象都有安全属性，同一个信息域中的信息对象拥有相同的安全属性，同一个信息域中的每一个信息对象的同一个安全属性具有相同的值。

在一个给定的信息域中，每一个用户所拥有的安全属性是相同的，但每个用户的每个安全属性的值可能不同。安全政策可以根据用户的安全属性的值区分不同用户的访问权限。

每个信息域可以拥有自己的信息安全政策，几个信息域也可以共享一个安全政策，共享一个安全政策的信息域可以通过它们所包含的用户或信息对象进行区分。

DGSA 是讨论安全政策及其实现问题的一个概念框架，它给安全操作系统开发者带来的挑战是，安全操作系统应该能够灵活地支持在一个计算平台上同时工作的多种安全政策，应该能够与不同平台上能够支持相同安全政策的其它操作系统进行互操作。

5.2 多政策环境中的安全政策支持范型

为解决在分布式多政策环境中支持用户定义的安全政策的问题，1992年，M. M. Theimer 等提出了访问控制程序（ACP）范型思想[53]，1993年，H. Hartig 等提出了看守员范型思想[54]。与传统的访问控制机制不同，CAP 和看守员范型都采用了基于算法的安全政策解决方法。

5.2.1 *访问控制程序范型* 在基于客户/服务器的分布式环境中，ACP 可以给充当客户的代表的中间实体进行细粒度的访问授权。一个 ACP 是一个程序，客户总是把它和一个请求绑定在一起传送给服务器。一个 ACP 描述一个客户针对每一个请求给中间实体所授予的权限。ACP 附带有数字签名，防止中间实体对它进行篡改。在服务器端，作为权限检查工作的一部分，服务器运行随请求一起收到的 ACP，如果 ACP 允许，则把访问权限授给中间实体。图3是 ACP 的示意。

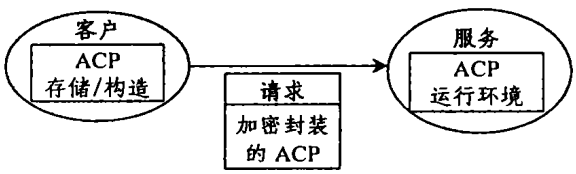

图3 与请求绑定的访问控制程序（ACP）

在 M. E. Kuhnhauser 给出的先进 ACP 范型中[55]，一个先进 ACP 构成一个装盛一个完整的安全政策的容器。在一个系统中，提交给一个安全政策实现组件的每一个实体都与一个对应的先进 ACP 相绑定，实体通信时，先进 ACP 与通信信息一起传送给目标对象，在目标对象端，ACP 中的安全政策被执行，以检查实体的操作与安全政策间的一致性。如果有多个实体被提交给同一个安全政策实现组件，所有这些实体则都与同一个先进 ACP 相绑定，以这种方式提交给一个安全政策实现组件的所有实体构成了该安全政策实现组件的域。

自主政策允许实体的属主选择或修改相关联的先进 ACP。先进 ACP 封装和强制实体关联实施的是强制政策。实施封装和应用实体关联时，先进 ACP 范型需要底层系统平台的引用监控机的支持，这样，先进 ACP 便把引用监控机的抗篡改和完全访问仲裁特性传播到用户定义的安全政策中。

5.2.2 *看守员范型* 看守员范型实现一个在分布式系统中支持用户定义的安全政策的对象模型[56]，在这个对象模型中，安全政策由算法和数据的组合表示。一个看守员是一个安全政策的保护外壳，能够为安全政策提供抗篡改性。一个看守员构成了一个政策中立的引用监控机的一个附加组件，在保持引用监控机的抗篡改、完全访问仲裁和可验证等特性的

同时，它为引用监控机吸收用户定义的安全政策提供了条件。过程调用的通信方式为支持多政策环境下的政策协作提供了良好的政策接口。

把安全政策施加到应用实体上的方法是把对应的看守员与应用实体关联起来。与同一个看守员关联的所有实体组成了相应安全政策的域，在这个意义上，一个安全政策就成了安全政策域中的每个实体的一个属性。有了安全政策属性的实体在通信中不再有自由，这样的实体的每一次通信都将被转向到相关联的看守员，如图4所示，结果，引起对政策操作的一次调用。政策操作与实体操作的关联依赖于政策的粒度，一个访问控制政策可以提供一个单一的操作，比如，Check-ReadAccess(Subject, Object)。一个安全政策可以定义与元政策进行互操作的操作。元政策是关于政策的政策[57]，它为多个安全政策的复杂共存建立了一个框架，这个框架包含有接口规则、互操作和冲突解决等。

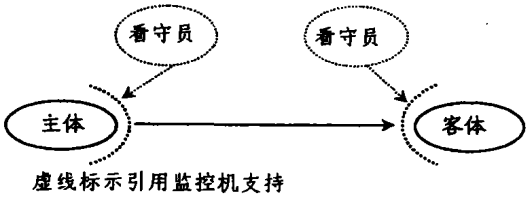

图4 保护着两个通信对象的看守员

看守员的思想可以方便地集成到多种多样的系统平台上，在每一种系统平台上，看守员是包含安全政策的普通的系统实体，所包含的安全政策是为它所关联的对象配备的。在BirliX安全体系结构中，看守员是普通的用户定义的抽象数据类型，在Unix系统平台上，看守员是普通进程(拥有由Unix文件实现的政策存储)，在OSF的分布式计算环境中，看守员被实现为DCE服务器。

5.3 基于Mach的DTOS安全操作系统

1997年完成的DTOS项目[58]属于Synergy项目[59]的一个组成部分，Synergy项目是操作系统研究的一个大项目，它的目标是为安全分布式系统开发一个灵活的、基于微内核的体系结构，激励安全操作系统厂商在下一代面向市场的操作系统中提供强大的安全机制。

5.3.1 DTOS的设计目标 与传统基于TCSEC标准的开发方法不同，DTOS项目采用的是基于安全威胁的开发方法[60]，它的主要目的之一是为Synergy体系结构的最底层软件组件开发一个原型系统。DTOS原型系统以Mach为基础，具有以下设计目标：

(1)政策灵活性：DTOS内核应该能够支持一系列的安全政策，包括诸如国防部的多级安全政策的强制访问控制政策。为提高系统的政策变换能力，由政策变化引起的必要的系统变化应该能局限在一个单一的系统组件(即，安全服务器)的范围内。

(2)Mach兼容性：DTOS内核应该能够支持现有的所有Mach应用，使它们顶多只受到实施的安全政策的限制。特别地，如果安全政策允许所有的操作，那么，现有的Mach应用应该能够在不做任何改变的情况下运行。

(3)性能：DTOS内核的性能应该与它的母体Mach内核的性能相近。

5.3.2 DTOS的安全体系结构 DTOS安全体系结构的基础是一个由管理器和安全服务器构成的通用系统框架[61]，如图5所示，它通过安全判定与安全实施的分离支持安全政策的灵活性。安全判定由安全服务器履行，安全判定的结果由管理器实施。

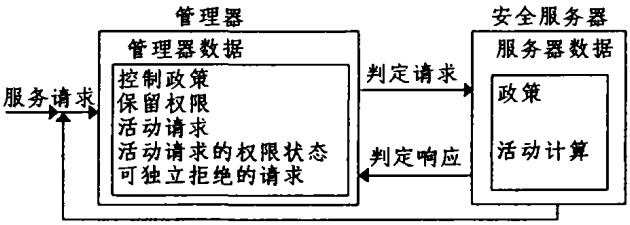

图5 DTOS安全体系结构通用框架

管理器是能够直接访问它所管理的客体的唯一主体，它接收来自各种客户主体的对客体进行操作的请求序列，并根据它的当前状态(有可能因来自安全服务器的政策信息的影响而变化)确定是否执行每个请求。安全服务器是系统中运行的进程，它根据一组安全规则进行安全判定。

管理器从其它主体(包括安全服务器)接收请求，并把安全请求发送给安全服务器，安全服务器把对安全请求的响应发送给管理器。

5.3.3 DTOS的原型系统 DTOS原型系统的管理器是CMU的Mach 3.0微内核的一个安全增强版本，它通过与安全服务器的协作，实施由安全服务器提供的安全判定，能够支持多种多样的安全政策，包括军用政策和民用政策，如多级安全(MLS)政策、基于标识的访问控制(IBAC)政策和类型裁决(TE)政策[62,63]等。已开发的安全服务器样例实现对MLS和TE政策的支持。DTOS的Lite Unix仿真环境提供安全Unix的职能。

DTOS实现的客体类型包括任务、线程和端口[64]。任务和线程代表系统中的活动主体，即进程。每个任务包含一组线程。服务器在系统中的实现体现为一个或多个任务。端口是任务传递消息的单向通信通道，如图6所示。

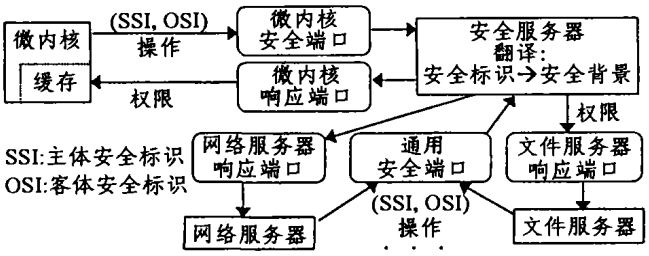

图6 DTOS原型系统安全服务器接口

任务利用权能来命名端口，每个权能描述向一个特定端口发送消息或从一个特定端口接收消息的权限，权能可以通过发送消息的方式从一个任务传递给另一个任务。

安全服务器通过微内核安全端口从微内核接收请求，或通过通用安全端口从其他服务器接收请求。它通过响应端口发送对请求的响应信息。(未完待续)