南京大学

研 究 生 毕 业 论 文

（申请硕士学位）

论 文 题 目 Nutos 系统混合多策略视图安全模型

作 者 姓 名 夏 磊

学科、专业名称 计算机软件与理论

研 究 方 向 计算机安全

指 导 教 师 黄 皓

二○○七 年 五 月 二十 日

学　　号：**MG0433047**

论文答辩日期：**2007** 年 **5** 月　日

指 导 教 师：　　（签字）

Dissertation Submitted to Nanjing University

for the Master degree

Toward A Hybrid Multiple Policy Views Security Model for Nutos operating system

Master Candidate: **Lei Xia**

Supervisor: **Prof. Hao Huang**

Major: **Computer Science**

Speciality: **Computer Security**

Department of Computer Science and Technology

Nanjing University

May 2007

摘 要

随着信息系统的复杂化和网络互联技术的广泛应用，计算环境的多样性和复杂性也在显著地增加，实际应用对系统资源保护的需求也越来越多样化。为了满足不同的安全需求，一个好的安全策略模型的实施可以有效保证系统应用软件的完整性和机密性，限制攻击者的非法行为，阻止攻击者对系统信息的随意破坏和任意泄露，防止各种病毒、蠕虫在系统内部的随意感染和传播，减小病毒、蠕虫和特洛伊木马对系统安全的危害。

早期的操作系统多采用了单一的安全策略来提供强制访问控制。但是，单—的策略已不足以满足广泛的安全需求和策略交互的需要，操作系统需要多安全策略的支持框架以支持策略的多样性。

（1）提出一个混合多策略模型—MPVSM（Multiple Policy Views Security Model）。多级安全策略 MLS 虽然能很好地防止信息的非授权泄漏，保护信息的机密性，但它不考虑信息的完整性保护问题，并且不能有效解决信道控制等问题，在处理多种安全密级信息问题时采用的特权主体方法带来了很大的安全隐患。而信道控制和安全降级正是防火墙等安全应用软件需要解决的问题。

MPVSM 模型有机综合了 MLS、DTE 和 RBAC 等安全模型的功能和属性，消除了 MLS 模型的缺陷，使其可以方便地表达各种信道控制策略，同时细化了权限的分配，提高了策略表达能力，保证了可信主体在不违背最小特权原则下解决安全降级的问题。并综合了多个安全模型的特点，为实现多安全策略视图提供了一个灵活的框架。文中还给出了 MPVSM 模型的在我们的原型可信操作系统 Nutos 中的实现以及具体的策略配置实例。

（2）分析了基于微内核操作系统的分层完整性保障机制，并应用到我们实现的原型操作系统 Nutos 中。操作系统内核是各种应用层安全机制的基础，而 Unix、Linux 等单块结构操作系统的内核都是一个庞大、复杂的程序，所有代码和模块都共享同一个内核地址空间并具有同样的无限特权，一旦内核中某模块存在漏洞则可能危及整个系统的安全。基于微内核的 Nutos 操作系统将内核划分为微内核层和功能服务模块层，为操作系统建立了有纵深的防御体系，有效地隔离了各种内核安全威胁，抵御内核模块攻击等针对系统内核的攻击，保证了操作系统的整体安全性。Nutos 系统在有效地保障自身各

功能模块，包括安全决策模块和决策实施模块的完整性的基础上实现多安全策略，保障操作系统应用用户的各种安全性需求，解决了存在于主流操作系统中的各种安全缺陷和安全隐患。

关键词 多级安全策略；多策略视图；可信操作系统；职责分离；最小特权；机密性保护；完整性保护；微内核操作系统；操作系统完整性

ABSTRACT

With the development and widely employment of information technology, the diversity and complexity of the computing environments are sharply increasing. Various security requirements are coming up in variety of applications. To satisfy these security requirements, a variety of security models were proposed in last twenty years. A good security model is able to enforce the confidentiality and integrity assurance in user applications and systems effectively, stop and confine the malicious behavious from attackers, prevent system information from destroyed or leaked by malicious attackers, or infected by various malicious codes, such as computer viruses, worms and Troy horses, confine and minimize the damages to system by these malicious codes.

Previous trusted operating system usually enforced only one kind of mandatory access control model. However, single one of current access control models can not well satisfy these diverse security requirements and policy interactions needs. There must be an effective and fexible framework for enforcing multiple policy views in operating system to solve this problem that so many kinds of security requirments always exist in the same system.

1) A hybrid multi-policy security model--Multiple Policy Views Security Model (MPVSM) is proposed.

Multilevel Security models aim only at confidentiality assurance of the information, preventing the unauthorized leakage of information. However, they rarely consider the integrity assurance, and are weak in expressing channel control policies. Moreover, the trusted subjects introduced in MLS to handle the information “downward” flowing problem have brought many potential security flaws.

MPVSM has integrated the properties and functions of MultiLevel Security, Domain-Type and Role Based models into one unified model. It combines confidentiality and integrity lattices into a unified lattice, for confidentiality and integrity assurance. It has enhanced ability in expression of channel control policies. In addition, it makes permission management more fine-grained and enhances the ability of confining the

permission of the trusted subjects. MPVSM is also able to enforce multi-policy views between different applications in system in a flexible way. Our prototype Nutos operating system to implement the MPVSM is also introduced.

2) We analyze kinds of possible mechanisms provided by microkernel based operating system architecture that can be used to improve the integrity assurance ability and enforce the principles of least privilege and separation of duties on the operating system functional components. The prototype operating system, Nutos, is presented, as an example on how to use these mechanisms to enforce the goals.

The integrity assurance of the operating system components is the foundation of the security of user applications and the whole system. Current mainstream operating systems like Unix and Linux are mono-kernel, in which all of the functional components of a operating system are in the same address space, and one compoent can access data of other compoents without any control or confinement. Once there are security flaws in one of its components, and got used by attackers, it will bring huge damage to the security assurance of the whole system.

Nutos divides all of the functions of an operating system to one microkernel running in kernel mode and several functional components running in user mode. These components are running in their own address spaces separately and can only interaction with each other by message passing, which is controlled by the microkernel. Our in-depth defense for operating system effectively isolates potential attacks or other security threatens, prevents many kinds of attacks aiming at kernel, such as kernel module attacks, and therefore gurantee the integrity of the operating system functional components. Based on the integrity assurance of its functional components, especially the components that make access control decisions and enforcement the decisions, Nutos solves many security shortcomings and flaws existing in the current mainstream operating systems, and has the ability of multiple policies expressions for various security needs of diverse applications.

Key Words multiple policy; multi-policy views; trusted operating system; separation of duites; least privilege; confidentiality assurance; integrity assurance; microkernel operating system; operating system integrity

目　　录

图表目录

第一章 绪论

1.1 引言

随着 Internet 和信息技术的不断发展和大规模应用，计算环境的多样性和复杂性也在显著地增加。实际应用对系统资源保护的需求也越来越多样化，不同计算环境下的各种应用有着不同的安全需求。为了满足不同的安全需求，几十年来，产生了多种安全策略和安全模型。一个好的安全策略模型的实施可以有效保证系统应用软件的完整性和机密性，限制攻击者的非法行为，阻止攻击者对系统信息的随意破坏和任意泄露，防止各种病毒、蠕虫在系统内部的随意感染和传播，减小病毒、蠕虫和特洛伊木马对系统安全的危害。同时，也很容易对系统中的策略进行配置和管理。

多级安全策略 MLS(Multilevel Security)[Bell 75]是目前各种安全系统应用最为广泛的一类安全策略。MLS 策略规定主体不能读取高于其密级的客体，不能修改低于其密级的客体。它的目的是防止高密级信息泄漏给低密级的用户。在 MLS 系统中即使存在特洛伊木马并且其骗取了对高密级信息的访问权，也不可能将其读到的秘密信息泄漏给未授权用户。在 MLS 中任何主体都是不可信任的，即使它是处理高密级信息的用户，因此在 MLS 系统中一个用户不能同时处理多种密级的信息。

虽然 MLS 能很好地防止信息的非授权泄漏，保护信息的机密性，但是它不允许主体同时处理多种密级的信息，这使得不违背 MLS 很多功能无法正常实现。如防火墙软件若不违背 MLS 的规则就不能完成其正常的连接内外网络的功能。而从信息处理的角度看，一些信息处理程序必须同时处理多种安全等级的信息，如一个信息管理系统可以综合多种密级的信息提供给用户一个综合信息（如平均值），但不允许用户知道具体的各项秘密信息。MLS 的另一缺点是它没有考虑信息的完整性，而完整性保护同样非常重要[Amoroso 91, Eswaran 75, Lipner 82]。

Biba 模型采用与 MLS 相似的方法通过标记限制信息的流动来维护信息的完整性要求， Biba 模型在数学上与 MLS 等价，都是基于格的模型。而基于格的模型都无法实现可信信道控制等重要安全问题。本质原因在于基于格的信息流策略是传递的，即：若信息能从 A 流入 B，从 B 流入 C，则信息也能从 A 流入 C；而不能描述信息只能从 A 经

B 控制后流入 C 这样的安全策略。而一个防火墙系统实际上是一个信道控制系统，这样的信道控制模型无法用多级安全策略来描述。

计算环境的多样性和复杂性也在显著地增加导致实际应用对系统资源保护的需求也越来越多样化，不同计算环境下的各种应用有着不同的安全需求。目前普遍存在的安全需求包括数据的机密性保护、用户数据和系统数据/代码的完整性保护、系统可用性、信息传递的信道控制、用户或主体的职责分离和最小特权原则等。

为了满足不同的安全需求，几十年来，产生了多种安全策略和安全模型。目前应用最广泛的安全模型包括多级安全模型（以 BLP 和 Biba 为代表）、DTE 安全模型和 RBAC 安全模型等。

早期的操作系统多采用了单一的安全策略来提供强制访问控制，例如：Multics 系统[Organick 72]采用 BLP 模型来实现多级安全策略。但是，如上所述，单—的策略已不足以满足广泛的安全需求和策略交互的需要。操作系统需要多安全策略的支持框架以支持策略的多样性，例如同时支持 MLS、DTE 和 RBAC 等。

当前，支持多种安全策略的系统大都是用不同的安全模型实现不同的安全策略。如 SELinux[Zanin 04,Spencer 99]同时采用三个安全子模型：MLS、TE 和 RBAC。这三个子模型独立配置，安全决策必须同时满足三个安全模型所对应的安全策略。这种方式实现起来复杂，代码和各种数据结构关系繁杂，开发工作量大、周期长，系统配置和维护困难且容易出错，效率不高。并且无法保证三个模型对应安全策略的一致性，也无法检验可能出现的冲突。因此，开发一种与安全策略无关的安全模型来实现多安全策略具有很重要的意义。

作为策略中性的模型，RBAC 被看作是替代传统的自主访问控制和强制访问控制的一种新的模型。RBAC 模型把系统划分成用户、角色和权限三部分，在用户与角色、角色与权限之间建立关系。基于角色的访问控制模型提供了灵活的授权机制，便于权限管理和实施最小特权原则，可以实现自主访问控制，也可以实现强制访问控制。但是由于 RBAC 的结构所限，用 RBAC 来直接实现自主访问控制、多级安全策略[Loscocco 01]或 DTE 安全模型非常复杂且没有实际实现的意义。同时，目前文献中还没有发现一个用 RBAC 实现的安全系统，它具有类似 Bell-LaPadula 模型那样的形式模型。

本文综合了 MLS、DTE 和 RBAC 等安全模型的思想，构建了一个混合多策略模型—MPVSM（Multiple Policy Views Security Model）。MPVSM 模型通过对多种安全模型属性的有机组合，消除了 MLS 模型的缺陷，可以有效地实施信道控制，细化了权限的分配，提高了策略表达能力，保障了可信主体的最小特权原则的实现，并综合了多个安

全模型的特点，为实现多安全策略视图提供了一个灵活的框架。

MPVSM 基于多级安全策略模型，它将 MLS 机密性格和 Biba 完整性格统一起来，同时实现系统的机密性策略和完整性策略。在多级安全策略的基础上添加域属性，把处于同一安全级别中的主体划分到不同的域，利用 DTE 模型的控制机制来实现同一安全级别中的权限分配的进一步细化和权限分配的灵活性，控制处于同一安全级别下不同职责的主体的分权。另一方面，利用 RBAC 模型，为主体设置角色，为角色分配权限，这些权限是独立于多级安全策略 MLS 的。我们利用这种角色授权机制为特定的可信主体设置策略，这样可以避免特定的可信主体受到 MLS 策略影响而无法完成需要的功能，又使得分配给该主体的权限能够遵循最小特权原则，有效地限制可信主体的权限范围。由于 RBAC 的角色授权机制具有很强的策略表达能力和策略配置的方便性，能够提高有效实现最小特权原则的能力，大大增强权限分配和策略表达能力。

MPVSM 模型有机结合了 BLP、Biba、DTE 和 RBAC 等多种安全模型的属性和功能。它可以从不同侧面实现 BLP、Biba、DTE 和 RBAC 等安全模型的功能，并对这些模型的特点进行有机的综合，细化了权限分配，增强了权限分配的灵活性和策略的表达能力，有效地解决了特定可信主体的权限的管理问题，在很大程度上满足了各种应用的不同安全需求。

1.2 相关工作

1.2.1 安全模型

安全模型没有一个统一的定义，总的来说是对计算机系统的安全相关行为和特性的抽象描述，在模型中忽略了不必要的实现细节，是对安全策略的简单、抽象和无歧义的描述。安全模型可以是形式化的，也可以是非形式化的，但形式化安全模型更能无歧义的表达安全策略，并便于推理系统的安全行为，是进行形式化安全验证的基础。有一类安全模型与安全策略无关，是一种通用的策略无关的安全模型，如 HRU 模型[Harrison 76]、Take-Grant 模型[Lipton 77]、TAM 模型[Sandhu 91]、信息流模型[Denning 77,Denning 76]、不干扰模型[Goguen 82,McLean 94]。研究这些模型的目的是为了解决系统安全的一些一般性问题，如模型的安全可判定性问题，安全性证明问题，避免和识别隐蔽信道问题以及安全系统的组合问题等等。而另一类，也是目前研究最多的，是与安全策略相关的安全模型，有时也称为安全策略模型如：BLP 模型[Bell 75]、Biba 模型[Biba 77]、RBAC 模型[Sandhu 97, Sandhu 96]、DTE 模型[Walker 96,Badger 95]等。安全策略模型为安全策

略和它的实现机制之间的关联提供了一种框架。J. P. Anderson 强调，开发一个安全系统，必须首先建立系统的安全模型。

1.2.1.1 BLP 模型

BLP 模型[Bell 75]是根据军方的多级安全策略（MLS）设计的，它要解决的本质问题是对对机密性信息的访问控制。它是最早、也是目前最常用的基于 MLS 策略的安全模型。MLS 中规定仅当一个主体的安全级不低于客体的安全级时，才允许该主体读该客体；仅当一个主体的安全级不高于客体的安全级时，才允许该主体写该客体。

BLP 模型是一个状态机模型，并认为如果一个系统的初始状态是安全的，并且状态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。简单安全特性（ss-特性）、星特性（*-特性）、自主安全特性（ds-特性）以及基本安全定理构成了 BLP 模型的核心内容。

虽然 BLP 被广泛应用，但它也有很多局限性。BLP 模型定义的安全性属性虽然是为了用于通用的计算机系统，但它以一组规则描述安全系统的行为。尽管这种基于规则的模型比较容易实现，但是它不能更一般地以语义的形式阐明安全性的含义。BLP 能够很好地防止信息的非授权泄漏，保护信息的机密性。但它没有考虑信息的完整性保护，而完整性保护在很多情况下同样非常重要[Amoroso 91, Eswaran 75, Lipner 82]。此外，BLP 模型不能用于信道控制[Rushby 92]，而类似的安全需求往往在一个实际的系统中是必须的，为了解决这些问题在 BLP 实现系统中只能引入一些可信主体来完成一些违背 BLP 模型规则但又是必需的任务，这显然违反了最小特权原则。

1.2.1.2 Biba 模型

BLP 模型只考虑了信息的保密问题而忽略了信息的完整性问题，没有采取任何措施制约对信息的非授权修改。为了实现信息完整性的安全要求，Biba 提出了一个类似于 BLP 的模型[Biba 77]，用来实现信息完整性的安全要求。在该模型中每个主客体都拥有一个完整性级别，低完整性的程序和信息不能破坏高完整性的信息。该模型的优势在于其简单性以及可以和 BLP 模型相结合弥补 BLP 不考虑完整性的缺点。

1.2.1.4 RBAC 模型

作为策略中性的模型，RBAC[Sandhu 97, Sandhu 96]被看作是替代传统的自主访问控制和强制访问控制的一种新的模型。RBAC 模型把系统划分成用户、角色和权限三部分，在用户与角色、角色与权限之间建立关系。基于角色的访问控制模型提供了灵活的

授权机制，便于权限管理和实施最小特权原则。

基本 RBAC(Role-Based Access Control)模型由 4 个基本元素组成：用户、角色、会话和权限。在一个系统中一个用户可以扮演多个角色，一个角色可以包含多个用户。而权限并不直接授予用户，而是将权限授予角色。基于角色的授权更利于对用户权限的管理。会话的引入使得模型能更好地实施最小特权原则，即在用户执行特定任务时只拥有完成该任务仅需的权限。

层次 RBAC 在基本 RBAC 的基础上引入了角色的层次关系，这种角色间的层次关系可以用来模型现实世界中职务或岗位的权限继承结构，可以简化系统安全策略的配置。约束 RBAC 在基本 RBAC 的基础上引入对系统各元素的约束机制。层次模型和约束模型可以结合成一个复合 RBAC 模型。

基于角色的访问控制模型的优点是提供了灵活的授权机制，并便于权限管理和实施最小特权原则，可以实现自主访问控制，也可以实现强制访问控制。但是由于 RBAC 的结构所限，用 RBAC 来直接实现自主访问控制、多级安全策略[Osborn 00]非常复杂且没有实际实现的意义。同时，目前文献中还没有发现一个用 RBAC 实现的安全系统，它具有类似 Bell-LaPadula 模型那样的形式模型。

1.2.1.5 DTE 模型

DTE（Domain and Type Enforcement）[Walker 96,Badger 95]模型是对 Boebert 和 Kain 提出的 TE（Type Enforcement）访问控制模型的改进版，并被应用于防火墙等系统。

TE 将系统视为各种主体和客体的集合，每一个主体都有一个安全属性“域”，而每一个客体也有一个安全属性“类型”，并用“域定义表”来描述各个域对不同类型客体的访问权限，而用“域交互表”来描述各个域之间许可的访问权限。并通过入口点程序来控制域的变迁或切换。

DTE 进行了一些改进：用一种高级语言来描述策略，方便了策略的配置并提高了策略的可重用性和兼容性；利用目录结构来设置 type 属性，不需要对每个文件设置；增加了域迁移的方式。DTE 能够很好地体现了最小特权和职责分离的安全原则，还可以有效地实现信道控制，保证信息流动的非传递性。然而，其访问控制策略的配置规则很复杂，使用 DTE 来实现多级安全策略的功能，保护系统中所有用户数据的机密性、完整性，其规则库将会很庞大，规则之间的关系也会很复杂，难以管理。同时 DTE 不是一个形式化的安全模型，很难验证其安全性。

1.2.1.6 不干扰模型

不干扰模型用“不干扰”概念来解释信息的流动，而不是用“读/写”这样语义模糊的操作来解释。如果将系统的任意输入序列中剔除所有用户 A 的输入后，系统对用户 B 的输出序列不变，则用户 A“不干扰”用户 B，即无信息从 A 流入 B。

系统的安全策略可以用“不干扰”关系来表达，如 MLS 策略可以表达为：Label(A) > Label(B) $\Rightarrow$ A 不干扰 B。

不干扰模型将系统描述为一个状态转换机，并严格定义了“不干扰”的语义。用“不干扰”关系来表达信息流策略，比限制“读写”操作的规则更接近信息流的本质，因此与 BLP 等模型只对各种“读写”操作进行监控相比，不干扰模型可以处理隐蔽信道问题。

不干扰模型提供了一个表达和验证安全策略的模式，但并不是一个建设性安全模型，并未提供一个简单易理解的访问控制规则和方法。另外不干扰模型的要求过于严格，一般系统很难完全满足。

1.2.2 安全操作系统

操作系统是计算机资源的直接管理者，它直接和硬件打交道并为各种应用软件提供接口，是计算机软件的基础和核心。操作系统的安全是整个计算机系统安全的基础，没有操作系统安全，就不可能真正解决数据库安全、网络安全和其他应用软件的安全问题[Loscocco 98]。AT&T 实验室曾经对美国 CERT 提供的安全报告进行过分析，结果表明，很多安全问题都源于操作系统的安全脆弱性[Smith 97]。因此对安全操作系统的研究具有极其重要的意义。

1.2.2.1 安全操作系统进展

安全操作系统的研究可以追溯到 1965 年美国贝尔实验室和麻省理工学院联合开发的 Multics[Organick 72]操作系统。虽然从商业角度看，Multics 并不成功，但从操作系统安全研究的角度看，Multics 迈出了安全操作系统设计和开发的第一步，为后来的安全操作系统的研制积累了大量的经验。特别是 Bell D.E.和 La Padula 提出的 BLP 安全模型在 Multics 中得到了成功的应用，随后一系列的安全操作系统相继被设计开发出来，如：Mitre 安全核[Schiller 73,Schiller 75]、UCLA 安全 UNIX [Popek 79,Walker 80]、KSOS[Ford 78,McCauley 79]以及 PSOS[Neumann 75,Feiertag 79]等。这一时期除了 BLP 模型外，引用监控器[Anderson 72,Lampson 74]、安全核[Lampson 76, Ames 83]以及可信计算基

（TCB）[Nibaldi 79-1,Nibaldi 79-2]等安全操作系统重要设计概念和理论被提出。

最早在操作系统实现 MAC 的机制是格[Bell 75, Saltzer 75, Weissman 69]。如 LOMAC [Fraser 99]使用了一种特殊的目标机制来实施 Biba 完整性。CMW [Berger 90]为每个客体分配了当前格和最大格两个标签，使用客体标签的动态变化来增加灵活性。

Asbestos [Efstathopoulos 05] 提供了标记和隔离机制来帮助实现那些包含软件错误的应用程序的安全执行。应用程序可以利用 Asbestos 内核提供的标记机制来灵活实现很多种类的策略，并且这种标记机制使得 MAC 更容易在现实系统中实现。

KernelSec [Radhakrishnan 06] 目的在于提高系统中实施的授权模型和安全策略的效力，提供一个有效的、健壮并且可分析的授权系统。

在基于权能的系统里，KeyKOS [Logic 89] 通过把进程隔离到每个单元并且使用引用监控器来监控权能在在单元之间跨越时的使用情况来实现多级安全。EROS（Extremely Reliable Operating System）[Shapiro 99] 成功地在现代硬件体系结构上实现了 KeyKOS 中提出的设计思想。Coyotes 内核[Shapiro 04] 是 EROS 的后续版本，主要致力于研究如何使用软件验证技术来实现验证内核的正确性和安全性的目标。

随着安全操作系统应用领域的扩大和各种安全策略模型的提出，仅支持单一安全策略的计算机系统已不能满足日益增长的各种应用需求，进入 20 世纪 90 年代后，人们开始研究对多策略和动态策略的支持。1993 年，美国国防部推出了 DGSA 安全体系结构，它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[Feustel 98,Halfmann 99]。1997 年完成的 DTOS（Distributed Trusted Operating System）[Secure 96,Secure 97,Fine 93]以 Mach 为基础，将策略执行单元和决策单元分离，能支持一系列安全策略，如 MLS、TE 等。1999 年结束的 Flask 是 DTOS 的后续项目[Spencer 99]，虽然 DTOS 结构独立于特定安全策略，能支持多策略，但它还无法支持动态安全策略。而 Flask 结构[Spencer 99]对 DTOS 结构进行了改造，克服了 DTOS 结构的不足，实现了动态安全策略。Flask 原型系统的安全服务器实现了由 MLS 和 TE 两个子策略组成的安全策略。2001 年发布的 Linux 安全增强型版本 SELinux[Zanin 04, Loscocco 01]为了支持策略灵活性采用了 Flask 安全体系结构，与 DTOS 和 Flask 不同，SELinux 并不是一个微内核操作系统。

1.2.2.2 引用监控器和安全核

引用监控器（Reference Monitor）和安全核（Security Kernel）概念，由 J. P. Anderson 于 1972 年首次提出，至今仍然是安全操作系统实现的重要基础理论。引用监控器是一种负责实施安全策略的软件和硬件的结合体，如图 1-1 所示。引用监控器根据访问控制数据库中的信息按照安全策略的要求来控制是否允许主体对客体的访问，并将重要的安全事件记录在审计文件中。访问控制数据库是动态的，体现了当前系统的安全状态，随着主体和客体的创建、删除以及访问权的修改而变化。引用监控器的关键是要对主体到客体的所有访问都要实施控制，即系统中主体没有可能绕过引用监控器而直接访问客体。

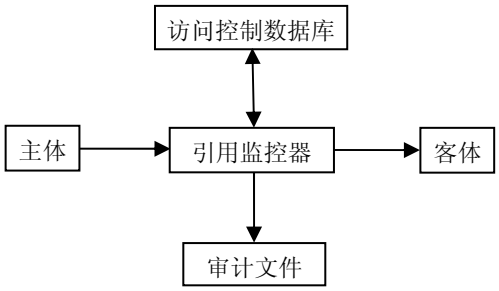

图 1-1 引用监控器

引用监控器只是一种理论上的概念，并不涉及它的具体实现方法。安全核是实现引用监控器概念的一种技术，其理论依据是：在一个大的操作系统内，只有相对比较小的一部分软件负责实施系统安全，通过对操作系统的重构，将与安全有关的软件隔离在操作系统的一个可信核内，而操作系统的大部分软件无需负责系统安全。安全核必须尽可能的小，以便对它的正确性进行检验。

安全核技术可以避免通常设计中固有的安全问题，大大提高用户对系统安全控制的信任度，但可能会影响系统的性能。因为该结构在用户程序和操作系统资源之间增加了新的层次。另一方面，要将操作系统中安全无关的功能全部排除在安全核外，以保持安全核的简单和小型并不是一件容易的事情，它往往与系统设计的其它因素相矛盾，如性能，甚至与设计安全核的基本原则完备性相矛盾。因此真正实现安全核的系统并不多，实现安全核的系统有：KSOS 和 UCLA 安全 Unix，而更多的安全操作系统为降低开发代价并不对操作系统内核进行分解，它们的安全核就是内核，如：LINVS IV、安全 Xenix、TMach、安全 TUNIS 以及各种安全增强型操作系统。

1.2.2.3 安全操作系统的一般开发方法

安全操作系统开发方法有多种，不同的开发方法在开发代价、兼容性、安全性、灵活性等方面各有不同，并无一种方法绝对优越，而需要根据需求来选择。这些方法可以归纳为以下 4 种基本方法。

(1)虚拟监控器安全核

在现有操作系统与硬件之间增加一个虚拟监控器作为安全核层，而原操作系统几乎不变地作为虚拟机来运行。在这种结构下，每个虚拟机都有一个唯一的标识符以及一个安全属性集合，安全核根据这个安全属性来监控各虚拟机对系统资源的访问，但安全核并不直接管理虚拟机中的进程操作。对于安全核来说，主体就是虚拟机，而操作系统传统的访问控制一般还是需要由操作系统来实施，而由安全核来实施原操作系统所不能提供的强制访问控制。采用该方法开发的系统有 KVM [Gold 84]，UCLA 安全 Unix，VAX VMM 安全核[Karger 90]以及 SVMM [Kelem 91]。该方法的优点是可以将安全核设计得足够小，并几乎完全兼容原有系统，但缺乏灵活性，安全核的管理粒度太大，不能适应复杂的上层应用的安全需求。

(2)安全核加仿真器

对现有操作系统的内核面向安全策略进行修改甚至重写，形成安全核，并在安全核与原操作系统用户接口之间增加仿真器软件层，以兼容现有的各种应用程序。仿真器的作用是将操作系统的功能翻译成安全核的调用功能，而不是作为一个完备的操作系统。该方法的问题是安全核实施的安全策略与仿真器接口之间往往不兼容：一些原有接口的功能本身就不安全，无法仿真；而一些安全核提供的安全功能无法用原有接口来表达。因此往往需要对原操作系统接口进行部分修改，在兼容性和安全性之间进行权衡。另外，由于操作系统的大部分功能需要在安全核中实现，仿真器方法设计的安全核往往比虚拟监控器安全核要大得多，但可以实现对进程、文件等细粒度主客体的安全控制。用该方法实现的系统典型的有：KSOS[Ford 78,McCauley 79]。

(3)对原系统内核的改造和增强

对现有操作系统的内核进行改造和扩充，加入安全机制使它支持新的安全策略，并尽量保持原有的系统接口（为了适应新的安全策略部分接口可能要进行修改和增加）。该方法由于受原有体系结构和接口的限制，且没有一个可验证的分离的小的安全核（安全核即操作系统内核），因此很难达到很高的安全级别，同时也存在要在兼容性和安全性之间进行权衡的问题。但这种方法不破坏原系统的结构，不增加软件层次，具有开发

代价小和附加性能开销小等优点。目前发布的多数安全操作属于这一类。

(4)重新构造新的安全操作系统

从头开始构建一个全新的安全操作系统，可以不受原有系统的限制，从设计的一开始就将安全作为重要因素加以考虑，因此可以排除各种现有操作系统的结构和功能中所固有的漏洞和不安全因素。1999 年完成的 EROS 内核就是一个典型的完全重新设计开发的系统。但总的来说，从头建立一个完整的安全操作系统代价太高，而且存在无法兼容现有的各种应用程序的问题，因此这样的系统多数还仅限于研究和军用。

1.2.3 微内核操作系统结构

与单内核的操作系统相对，微内核操作系统将系统的大多数功能模块如内存管理，进程管理、设备驱动和网络等以运行在用户模式下服务进程的形式实现。内核只实现那些上层功能所需的最基本的机制，如基本内存地址空间映射、消息传递、中断截获等。消息是用户进程与服务进程以及服务进程之间的唯一交互方式。

最初提出微内核操作系统结构并不是出于安全考虑。然而，我们可以将微内核结构思想用在安全操作系统的设计中，用来保护操作系统的完整性。首先，由于微内核的功能很少，代码很小，这样，无论从理论上还是工程上，微内核的自身的安全性都更容易得到保证。理论上，一个很小的内核更容易通过形式化的方法来验证它的安全性。工程上，一个功能较少、代码紧凑的内核更容易调试，更不容易出现实现上的安全漏洞。其次，将操作系统的大多数功能以用户态服务进程的形式实现，这些进程与内核和其他用户进程都处于不同的地址空间，这种物理上的隔离使得即使某个服务模块遭受攻击，也能把这种攻击控制在该进程范围之类，而不会影响到整个系统。第三，用户进程与服务进程以及服务进程之间都是通过消息进行交互，而所有消息传递必须经过微内核的仲裁，因此，各功能模块之间的交互通道可以得到有效的标识和控制。事实上，在下面的章节中会看到，通过对微内核和服务进程添加访问控制策略，针对系统内核的攻击变得几乎不可能。

早期微内核操作系统的典型代表有 Mach[Loepere 92]、DTOS[Secure 96, Secure 97]等，然而使得这些系统无法投入实际使用的重要原因都是效率问题，系统进程之间频繁的消息通讯导致系统整体效率大幅度的降低。一般认为微内核系统的消息通讯机制带来的效率问题是不可避免的，然而 Liedtke 等人[Liedtke 96]认为导致 Mach 等微内核操作系统效率问题真正原因在于它们的架构问题而非微内核本身的缺陷，Liedtke 认为早期

的微内核系统是从单内核系统演化过来，设计上还没有摆脱单内核系统的影响，没有实现真正意义上的微内核，这是导致效率低下的本质原因。基于这种思想，Liedtke 等人设计实现了微内核操作系统 L4，通过多种设计和实现方法[Liedtke 93, Liedtke 95]使得该系统的效率与同条件下的 Unix 相差无几。可见，效率问题并不是微内核架构操作系统不可逾越的问题，只要在架构和设计上经过精心的考虑，微内核效率低下的问题是可以在很大程度上得到解决的。

第二章 动机

随着 Internet 和信息技术的不断发展和大规模应用，计算环境的多样性和复杂性也在显著地增加。实际应用对系统资源保护的需求也越来越多样化，不同计算环境下的各种应用有着不同的安全需求。为了满足不同的安全需求，几十年来，产生了多种安全策略和安全模型。一个好的安全策略模型的实施可以有效保证系统应用软件的完整性和机密性，限制攻击者的非法行为，阻止攻击者对系统信息的随意破坏和任意泄露，防止各种病毒、蠕虫在系统内部的随意感染和传播，减小病毒、蠕虫和特洛伊木马对系统安全的危害。同时，也很容易对系统中的策略进行配置和管理。

多级安全策略 MLS(Multilevel Security)是目前各种安全系统应用最为广泛的一类安全策略。MLS 策略规定主体不能读取高于其密级的客体，不能修改低于其密级的客体。它的目的是防止高密级信息泄漏给低密级的用户。在 MLS 系统中即使存在特洛伊木马并且其骗取了对高密级信息的访问权，也不可能将其读到的秘密信息泄漏给未授权用户。在 MLS 中任何主体都是不可信任的，即使它是处理高密级信息的用户，因此在 MLS 系统中一个用户不能同时处理多种密级的信息。

虽然 MLS 能很好地防止信息的非授权泄漏，保护信息的机密性，但是它不允许主体同时处理多种密级的信息，这使得不违背 MLS 很多功能无法正常实现。如防火墙软件若不违背 MLS 的规则就不能完成其正常的连接内外网络的功能。而从信息处理的角度看，一些信息处理程序必须同时处理多种安全等级的信息，如一个信息管理系统可以综合多种密级的信息提供给用户一个综合信息（如平均值），但不允许用户知道具体的各项秘密信息。MLS 的另一缺点是它没有考虑信息的完整性，而完整性保护同样非常重要。

Biba 模型采用与 MLS 相似的方法通过标记限制信息的流动来维护信息的完整性要求， Biba 模型在数学上与 MLS 等价，都是基于格的模型。而基于格的模型都无法实现可信信道控制等重要安全问题。本质原因在于基于格的信息流策略是传递的，即：若信息能从 A 流入 B，从 B 流入 C，则信息也能从 A 流入 C；而不能描述信息只能从 A 经 B 控制后流入 C 这样的安全策略。而一个防火墙系统实际上是一个信道控制系统，我们

可以用图 2-1 来描述一个简单防火墙系统的信道控制功能。防火墙的外连模块不能和内连模块直接通信而必须经过防火墙的访问控制模块，在该模块的控制下外连模块和内连模块进行通信，这样的信道控制模型无法用多级安全策略来描述。

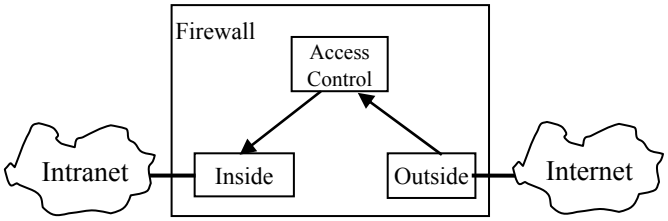

图 2-1 防火墙系统的信道控制功能示意图

主体需要同时处理多种安全级别信息以及信息的安全级别变化的例子还有很多。如在一个加密系统中，高密级的信息被加密程序加密后，包含秘密信息的密文应该是可以降低密级的，以便通过低密级安全域，这其实是加密的一个重要目的，但却违反 MLS 策略。

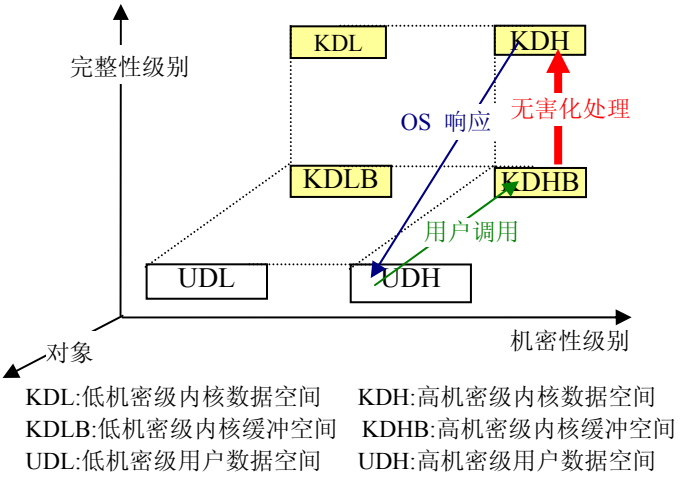

图 2-2 用户进程和操作系统之间的信息交互

另外一个典型的例子是用户和操作系统的数据交互。在 Biba 模型中,每个主体和每个客体都分配一个完整性级别，完整性越高代表信息和代码的可信度越高。而应用进程的完整性级别低于操作系统组件和操作系统数据文件。根据 Biba 的规则，应用进程的数据不能够流向操作系统。然而在实际系统中，虽然不允许用户进程破坏操作系统的完整性，但是，需要提供一条路径给用户进程来传递数据给操作系统，如系统调用的参数

等数据。图 2-2 所示为一个系统调用中，应用进程和操作系统之间的信息交互过程。操作系统的私有数据根据其机密性级别的不同分别保存在 KDL 和 KDH 中。KDLB 和 KDHB 是操作系统用来接收应用程序输入的缓冲数据区。UDL 和 UDH 是用户进程数据空间。在应用进程与操作系统的交互过程中，最重要的是保障操作系统关键数据的完整性。KDH 与 KDHB 是相互隔离的，只有操作系统相关组件才能对 KDH 区域进行写操作。用户进程的输入数据只能写入 KDHB。操作系统中的无害化处理代码对 KDHB 中的数据进行无害化处理后，把这些数据传入 KDH。用户进程无法直接地破坏 OS 数据的完整性。这样，我们就可以集中精力来验证无害化代码的正确性和可靠性。

目前，为了解决上述的这两类情况，很多实现 MLS 的安全系统采用的方法是将一些安全核外的主体视为可信主体，并把这些主体排除在 MLS 控制的范围以外，给它们特权使它们可以绕过系统的访问控制机制来直接访问各种资源，如 HP-UX/CMW 等。而这些能绕过访问控制机制的可信主体往往获得了比实际需要大得多的权限，是各种攻击者的主要攻击对象，成为系统的安全隐患。

此外，计算环境的多样性和复杂性显著地增加导致实际应用的安全需求越来越多样化，不同计算环境下的各种应用有着不同的安全需求。而早期的操作系统多采用了单一的安全策略来提供强制访问控制，例如：Multics 系统采用 BLP 模型来实现多级安全策略。但是，操作系统应用的多样化带来了安全需求的多样性，单—的策略已不足以满足广泛的安全需求和策略交互的需要。例如，普通用户最大的安全需求是机密性和完整性保障。而对于特权用户进程，职责分离和最小特权原则也很重要。对于大多数网络服务进程，如防火墙或网关，信道控制和用户信息的隔离更重要。

上述这些安全需求通常情况下是共存于同一个系统中的。而当前的每个安全模型都只侧重于其中某一种或几种安全需求，如，Bell-LaPadula(BLP)模型解决的本质问题是机密性保护，但它没有考虑信息的完整性保护和信道控制，也不能很好地支持职责隔离和最小特权原则的实现。Biba 主要针对信息完整性的安全要求。DTE 可以有效地实现信道控制，保证信息流动的非传递性。RBAC 能够很好地体现了最小特权和职责分离的安全原则。这些安全需求无法通过单个主流的安全策略模型来实现，系统中应该存在一种多策略视图，即针对不同的应用采用不同的策略模型。

当前，支持多种安全策略的系统采用的是简单叠加不同的安全模型来实现不同的安全策略。如 SELinux 同时采用两个安全子模型：MLS 和 TE。这两个子模型独立配置，安全决策必须同时满足两个安全模型所对应的安全策略。这种方式实现起来较机械、复杂，代码和各种数据结构关系繁杂，开发工作量大、周期长，系统配置和维护困难且容

易出错，效率不高。

作为策略中性的模型，基于角色的访问控制 RBAC 模型便于权限管理和实施最小特权原则。但是由于 RBAC 的结构，用 RBAC 来直接实现自主访问控制、多级安全策略[15]非常复杂，在实际系统中没有应用价值。

本文综合了 MLS、DTE 和 RBAC 等安全模型的思想，构建了一个混合多策略模型—MPVSM（Multiple Policy Views Security Model）。MPVSM 模型通过对多种安全模型属性的有机组合，消除了 MLS 模型的缺陷，可以有效地实施信道控制，细化了权限的分配，提高了策略表达能力，保障了可信主体的最小特权原则的实现，并综合了多个安全模型的特点，为实现多安全策略视图提供了一个灵活的框架。

MPVSM 基于多级安全策略模型，它将 MLS 和 Biba 这两个格叠加起来形成一个统一的格，同时实现系统的机密性策略和完整性策略。在多级安全策略的基础上添加域属性，把处于同一安全级别中的主体划分到不同的域，利用 DTE 模型的控制机制来实现同一安全级别中的权限分配的进一步细化和权限分配的灵活性，控制处于同一安全级别下不同职责的主体的分权。另一方面，利用 RBAC 模型，为主体设置角色，为角色分配权限，这些权限是独立于多级安全策略 MLS 的。我们利用这种角色授权机制为特定的可信主体设置策略，这样可以避免特定的可信主体受到 MLS 策略影响而无法完成需要的功能，又使得分配给该主体的权限能够遵循最小特权原则，有效地限制可信主体的权限范围。由于 RBAC 的角色授权机制具有很强的策略表达能力和策略配置的方便性，能够提高有效实现最小特权原则的能力，大大增强权限分配和策略表达能力。

MPVSM 模型有机结合了 BLP、Biba、DTE 和 RBAC 等多种安全模型的属性和功能。它可以从不同侧面实现 BLP、Biba、DTE 和 RBAC 等安全模型的功能，并对这些模型的特点进行有机的综合，细化了权限分配，增强了权限分配的灵活性和策略的表达能力，有效地解决了特定可信主体的权限的管理问题，在很大程度上满足了各种应用的不同安全需求。

第三章 MPVSM 模型

3.1 模型概述

MPVSM 模型包含有主体、用户、角色、域和客体等元素。包含用户-主体关系、用户-角色分配关系、主体-角色关系、角色-域授权关系。用户-主体关系给出了当前运行的每个主体所代表的用户，用户-角色分配关系给出了每个用户可以拥有的所有角色，主体-角色关系给出主体当前所具有的角色，每个用户分配多个角色，但主体每次只能代表一个角色运行，角色-权限关系把权限分配给角色，角色-域授权关系给出了角色允许进入的所有域，角色还拥有一个安全标记属性，代表了该角色的安全级别，同时，主体当前的安全级别由主体当前角色的安全级别决定，主体当前所处的域必须是已经授权给主体当前代表角色的域。每个客体都具有一个类型属性，指出该客体当前的类型，客体还拥有一个安全标记属性，给出该客体的安全级别。

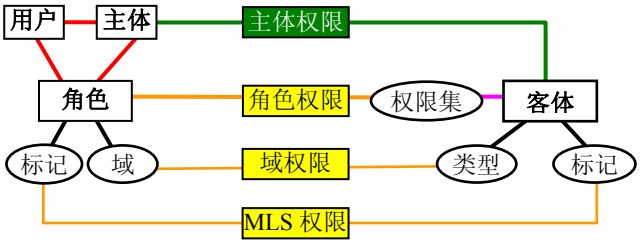

图 3-1 MPVSM 模型之元素-属性-关系示意图

每个运行的主体拥有的权限从三个方面得出：1) MLS 权限：根据主体代表的角色的安全级别(包括机密性级别和完整性级别)和被访问客体的安全级别，遵守多级安全访问控制规则的权限。2) 域权限：主体当前所处的域对某类型的客体的访问权限。3) 角色权限：主体代表的角色通过角色-权限关系获得的权限。

我们将 MLS 访问权限作为粗粒度的基础权限，指出主体具有读类权限或写类权限，而域访问权限是在 MLS 基础权限上对主体拥有的权限的进一步细化，指定各种细粒度的权限。角色访问权限是直接分配给角色的权限，这种权限是独立于前两种权限，这种

独立的权限定义方法便于我们对设计“特权程序”，并有效地对它的权限进行控制。

MPVSM 模型包含了三个方面的访问控制视图：1)多级访问控制：每个角色拥有一个安全标记，代表了该角色的安全级别，主体的安全级别由其当前代表的角色的安全级别决定。不同的安全级别拥有不同的权限，主体拥有的所有权限都不能超出该安全级别所允许拥有的权限的范围。2)域访问控制视图：主体可以在不同的域中运行，每个主体可以进入那些已经授权给了主体当前代表的角色的那些域，同时主体从当前域进入某个域还必须在域-域交互控制表中对该域具有“转换”权限。处在某个域的主体对处在某个类型的客体具有的权限由域-类型访问控制矩阵定义。3)角色控制视图：角色分配给用户，每个用户可以拥有多个角色，每个角色拥有不同的权限。主体代表着当前用户的某个角色运行，主体拥有该角色的所有权限。

多级访问控制和域访问控制是互补的关系，多级访问控制把系统中的权限进行粗粒度的划分，而域访问控制则对权限和权限划分进行细化，这样，既利用了多级访问控制的简洁性，又有域访问控制对权限分配的灵活性。角色访问控制是独立于前两种视图，它分配的权限独立于其它权限。角色访问控制用来处理那些特定的可信主体的权限控制问题，利用灵活的角色权限授权机制来实现可信主体的最小特权。

MPVSM 模型是可扩展模型。在我们的模型中，角色具有标记和域属性，客体具有标记和类型属性。这些属性可以扩展，从而丰富模型的表达能力。

3.2 模型中的基本元素

MPVSM 模型结构如图 3-1 所示，模型中包含的元素、属性和关系的定义如下：

定义 3.1 基本元素集合

- 用户集 U
- 主体集 S
- 客体集 O
- 角色集 R
- 域集 D
- 类型集 T
- 机密性标记集 C
- 完整性标记集 I
- 安全标记集 $SL \subseteq C \times I$，安全标记由机密性标记和完整性标记组成。

- 访问模式集 *P*，包括两个子集：读类模式集 *RP*={*read(r),execute(e),getattr(g)* **...** }；写类模式集 *WP*={*write(w), append(a), create(c), delete(d), setattr(s), sig-kill (sk),* **...**}。同时也可按目标对象的不同划分为客体访问模式 *AP* 和主体交互模式 *IP*。*P*=*RP*∪*WP*，同时，*P*=*AP*∪*IP*。其中，域交互模式 *transfer(t)*∈*IP*，*transfer* 表示主体从一个域转换到另外一个域的操作。
- 角色权限集 *CAP*⊆*P*×(O∪S), (p, t)∈*CAP* 的含义是：以方式 p 访问对象 t.

定义 3.2 用户-主体关系

- *US*⊆*U*×S，用户-主体关系，一对多的关系，每个用户可以有多个主体同时运行，而每个主体只能代表一个用户运行。
  - *user*：*S*→*U*，主体到用户的映射，该映射为单射，即每个主体只能代表一个用户运行。*user*(s)=u 当且仅当(u, s)∈*US*.

定义 3.3 用户-角色分配关系

- *UA*⊆*U*×*R*，用户-角色分配关系，多对多的关系，每个用户可以分配多个角色，每个角色也可以被分配给多个用户。
  - *UR*：*U*→2^R，用户到其所拥有的角色集合的映射，*UR*(u)={r∈*R*|(u,r)∈*UA*}.
  - *SR*：*S*→*R*，主体到角色的映射关系，为单射。任何时刻主体只以一个角色运行，且该角色是已经分配给主体当前代表的用户的角色，即：*SR* (s)∈*UR*(*user*(s)).

定义 3.4 角色安全属性

- *RL*：*R*→*SL*，角色的安全标记，该映射为单射。每个角色有且只有一个安全标记。
  - *Ssl*：*S*→*SL*，主体的安全标记，该映射为单射。主体安全标记为主体当前运行角色的安全标记，即：*Ssl* (s)= *RL*(*SR*(s)).

定义 3.5 角色-域授权关系

- *RD*⊆*R*×*D*，角色-域授权关系，多对多的关系。多个角色进入同一个域，一个角色也可以进入多个域。
  - *RDom*：(r:*R*)→2^D，角色到其所授权的域集合的映射，*RDom*(r)={d∈*D*|(r,d)∈*RD*}。
  - *SDom*：*S*→*D*，主体当前所在的域，该映射为单射，每个主体同一时刻只能在一个域中运行，并且该域已经授权给主体当前的运行角色，即：*SDom*(*s*)∈*RDom*(*SR*(s))。

定义 3.6 角色-权限分配关系

- *RCAP*⊆*R*×*CAP*，角色-权限关系，角色拥有的角色权限。

- *Rolecap*：$R \rightarrow 2^{CAP}$，每个角色的拥有所有角色权限的集合。*Rolecap*(r)={cap∈*CAP*|(r, cap)∈*RCAP*}。

定义 3.7 客体安全属性

- *OT*：$O \rightarrow T$，客体的类型属性，该映射为单射，每个客体只能属于一种类型。
- *OL*：$O \rightarrow SL$，客体的安全标记，该映射为单射，每个客体只有一个安全标记。

定义 3.8 域-类型/域-域访问控制矩阵

- *DTM*：$D \times T \rightarrow 2^{AP}$，域-类型访问控制矩阵。p∈*DTM*(d, t)的含义是：域d中的主体能以方式p访问具有类型t的客体。
- *DDI*：$D \times D \rightarrow 2^{IP}$，域-域交互控制矩阵，*transfer*∈*DDI*(d_1, d_2)表示域 d_1 中的主体可以转换到域 d_2.

定义 3.9 多级安全访问控制规则 *MLS_rule*：$SL \times SL \rightarrow 2^{P}$，*a*∈*MLS_rule*(ls,lo)的含义是：具有安全标记 *ls* 的主体对具有安全标记 *lo* 的客体具有 *a* 访问模式。该规则函数组合了 BLP 机密性和 Biba 完整性两个格。令 *ls*=(cs, is)，*lo*=(co, io).

- 当 cs≥co，允许读一类操作，即：*RP*⊆*MLS_rule*(ls, lo).
- 当 cs＜co，禁止读一类操作，即：∀p∈*RP*，p∉*MLS_rule*(ls, lo).
- 当 is≥io 时，允许写一类操作，即：*WP*⊆*MLS_rule*(ls, lo).
- 当 is＜io 时，禁止写一类操作，即：∀p∈*WP*，p∉*MLS_rule*(ls, lo).

3.3 权限规则

定义 3.10 主体-客体访问权限

- MLS访问权限map：主体s对客体o的MLS权限map(s, o)={p|p∈*MLS_rule*(*Ssl*(s), 、*OL*(o))}.
- 域访问权限dtp：主体s对客体o的域权限dtp(s, o)={p|p∈*DTM*(*SDom*(s), *OT*(o)).
- 角色访问权限rap：主体s对客体o的角色权限rap(s, o)={p|(o, p)∈*Rolecap*(*SR*(s))}.

定义 3.11 主体 s 对客体 o 拥有的访问权限 ap(s, o)= (map(s, o)∩dtp(s, o))∪rap(s, o).

定义 3.12 主体-主体交互权限

- MLS交互权限mip：主体s1对主体s2的MLS交互权限mip(s1, s2)= {p|p∈*MLS_rule*(*Ssl*(s1), *Ssl*(s2))}.
- 域交互权限ddp：主体s1对主体s2的域交互权限ddp(s1, s2)={p|*p*∈*DDI*(*SDom*(s1),

SDom(s2))}.

- 角色交互权限rip：主体s对客体o的角色权限rip(s1, s2)={p|(s2, p)∈*Rolecap*(*SR*(s1))}.

定义 3.13 主体 s1 对主体 s2 的交互权限 ip(s1,s2)= (mip(s1,s2)∩ddp(s1,s2))∪rip(s1,s2).

3.4 模型的限制条件

限制条件 1 任何状态下，客体都有且仅有一个类型属性、安全标记属性，即：

dom(*OT*)=*O*, dom(*OL*)=*O*.

限制条件 2 任何状态下，主体都只能处于一个域当中、并且该域是主体当前角色拥有的域，即：

dom(*SDom*)=*S*, *SDom*(s)∈*RDom*(*SR*(s)).

限制条件 3 任何状态下，主体都只代表一个用户，并且以一个角色执行，该角色必须是已经分配给用户的角色，即：

dom(*user*)=*S*, dom(*SR*)=*S*, *SR*(s)∈*UR*(*user*(s)).

限制条件 4 任何状态下，主体都只有一个安全标记，该安全标记为主体当前角色的安全标记，即：

dom(*Ssl*)=*S*, *Ssl* (s)= *RL*(*SR*(s)).

3.5 形式化系统描述

上节给出了 MPVSM 模型中的基本元素、元素之间的关系和属性的定义。本节给出 MPVSM 模型中的状态、状态的安全属性及系统规则、安全系统的定义，最后还给出了系统的安全定理。

定义 3.14 系统状态集 V。系统状态 v∈V 是一个四元组(b, c, m, e, f),其中 b∈P(*S*×*O*×*P*)，表示主体 s 可以以方式 p 来访问客体 o；c∈P(*S*×*S*×*SP*)，表示主体 s1 可以以 p 方式与主体 s2 交互；m 为(*DTM*,*DDI*,*MLS_rule*)，其中 *DTM* 和 *DDI* 分别对应该状态下系统的域-类型访问控制矩阵和域-域交互控制矩阵；*MLS_rule* 对应该状态下系统的多级安全访问控制规则；e 为(*U*, *S*, *O*, *R*, *D*, *T*)，分别对应该状态下系统的用户集、主体集、客体集、角色集、域集和类型集； f 为(*US, UA, SR, RD, SDom, RL, Ssl, OT, OL, RCAP*)，分别对应该状态下系统的各种关系和映射。

定义 3.15 系统状态的各种安全属性

- 一个状态(b,c,m,e,f)满足**多级安全属性**当且仅当：∀(s,o,p)∈b, p∈*MLS_rule*(*SL*(s), *OL*(o))并且∀(s_1,s_2,p)∈c, p∈*MLS_rule*(*SL*(s_1), *SL*(s_2)).
- 一个状态(b,c,m,e,f)满足**域安全属性**当且仅当：∀(s,o,p)∈b, p∈*DTM*(*SDom*(s),*OT*(o))并且∀(s_1,s_2,p)∈c, p∈*DDI*(*SDom*(s_1), *SDom*(s_2)).
- 一个状态(b,c,m,e,f)满足**角色安全属性**当且仅当： ∀(s,o,p)∈b, (o,p)∈*Rolecap* (*SR*(s))并且∀(s_1, s_2,p)∈c, (p, s_2)∈*Rolecap*(*SR*(s)).

定义 3.16 一个状态是安全的当且仅当该状态同时满足以下三个条件：

1）满足模型**限制条件 1-4**。

2）满足**域安全属性**和**多级安全属性**或满足 3）。

3）满足**角色安全属性**。

定义 3.17 状态(b, c, m, e, f)到状态(b*, c*, m*, e*, f*)的转换满足**域转换安全属性**当且仅当满足以下条件：

1） ∀s∈(*S**∩*S*), 如果 *SDom*(s)≠*SDom**(s)，则 *transfer*∈*DDI*(*SDom*(s), *SDom**(s))

定义 3.18 状态序列 z∈Z 是一个**安全状态序列**当且仅当：

1） ∀i∈N，z_i 是安全的状态。

2） ∀i∈N，z_{i-1} 到 z_i 的状态转换满足域转换安全属性。

定义 3.19 *RA*：请求集合。*DE*：判断集合。对请求作出的判定结果，包括：YES(允许)、NO(拒绝)、ILLEGAL(非法请求)、ERROR(错误)。系统动作集 W⊆*RA*×*DE*×*V*×*V*，表示一个请求被发出，针对该请求作出的决策，以及引起的系统状态的转换。

设定 N 为自然数集合。$X=RA^N$，该集合的元素 x 为一个请求的序列。$Y=DE^N$，该集合的元素 y 为一个决策的序列。$Z=V^N$，该集合的元素 z 为一个状态的序列。x、y、z 的第 i 个元素表示成 x_i、y_i、z_i.

定义 3.20 系统定义为∑(RA, DE, W, z_0)⊆*X*×*Y*×*Z*，z_0 是系统初始状态。(x, y, z)∈∑(RA, DE, W, z_0)当且仅当，对于所有 i∈N, (x_i, y_i, z_i, z_{i-1})∈W.

定义 3.21 (x,y,z)∈∑(RA, DE, W, z_0)是安全的当且仅当 z 是一个安全状态序列。一个系统∑(RA, DE, W, z_0)是安全的当且仅当： ∀(x,y,z)∈∑(RA, DE, W, z_0)是安全的。

定义 3.22 规则是一个函数 ρ：RA×V→DE×V。给定一个状态和请求，规则给出对该状态的判定结果和转换后的状态。(ra, de, v, v*)∈*RA*×*DE*×*V*×*V* 是系统∑(RA, DE, W, z0)的一

个规则当且仅当，存在 $n\in N$，$(ra, de, v, v^*)=(x_n, y_n, z_n, z_{n-1})$.

定义 3.23 一个规则 ρ 是安全的当且仅当：

1）对于所有 $\rho(ra, v)=(de, v^*)$，如果 v 是一个安全状态，那么 v*也是一个安全状态。

2）对于所有 $\rho(ra, v)=(de, v^*)$，如果 v 是一个安全状态，那么 v 到 v*的状态转换满足域转换安全属性。

下面给出了系统安全定理：

定理 3.1 $\sum(RA, DE, W, z_0)$是一个安全的系统当且仅当 z_0 是一个安全状态并且 W 中的规则都是安全的。

证明 假设$\sum(RA, DE, W, z_0)$是不安全的。令$(x, y, z)\in\sum(RA, DE, W, z_0)$是不安全的，则z为不安全的状态序列，因此z中存在不安全状态或是z中存在状态转换不满足域转换安全属性。

1）设z_t是z中不安全的状态中t最小的一个状态。因为z_0是安全的，故t>0。根据我们的假设，z_{t-1}是安全的，并且根据$\sum(R,D,W,z0)$的定义，$(x_t,y_t,z_t,z_{t-1})\in W$。并且存在一个唯一的规则$\rho\in\omega$，$\rho(x_t,z_{t-1}) = (y_t,z_t)$，因为ρ是安全的，$z_{t-1}$是安全的，根据定义，$z_t$也是安全的，与前提假设相矛盾。

2）假设z_{t-1}到z_t的转换不满足域转换安全属性，同样，存在一个唯一的规则$\rho\in\omega$，$\rho(x_t,z_{t-1}) = (y_t,z_t)$，因为ρ是安全的，所以$z_{t-1}$到$z_t$的转换满足域转换安全属性，所以假设不成立。

综上，假设不成立，因此$\sum(RA, DE, W, z_0)$是安全的。

3.6 模型规则

规则 1 request_access(s, o, a)

请求类型为$R^{(1)}$，含义是：主体s请求以a方式访问客体o。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $(a, o)\in Rolecap(SR(s))$

(b) $a\in DTM(SDom(s), OT(o))\ \wedge\ a\in MLS\_rule(SL(s), OL(o))$

如果条件(a)或(b)之一满足，决策输出YES．下一个状态变化如下：

- $b^*=b\cup\{(s, o, a)\}$
- 其他变量保持不变。

如果上述条件不成立，决策输出NO; 下一状态保持原状。

规则 2 release_access(s, o, a)

请求类型为$R^{(2)}$：主体s请求释放以a方式访问客体o。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) (s, o, a)∈b;

如果条件(a)满足，决策输出YES。下一个状态变化如下：

- b*=b\{(s, o, a)}
- 其他变量保持不变。

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 3 request_interact(s1, s2, i)

请求类型为$R^{(3)}$：主体s1请求以方式i与主体s2交互。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) (*SDom* (s1), *SDom* (s2),i)∈*DDI* ∧ i∈*MLS_rule*(*SL*(s1), *SL*(s2))

(b) (i, s2)∈*Rolecap*(*SR*(s1))

如果条件(a)或(b)之一被满足，决策输出YES。下一个状态变化如下：

- c*=c∪{(s1, s2, i)}
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 4 release_interact(s1, s2, i)

请求类型为$R^{(4)}$：主体s1请求释放以方式i与主体s2交互。如果请求方式不恰当，则下一状态保持原状，决策输出ILLEGAL；否则，检验如下条件：

(a) (s1, s2, i)∈c;

如果条件(a)满足，决策输出YES．下一个状态变化如下：

- c*=c\{(s1, s2, i)}
- 其他变量保持不变。

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 5 create_object(s, t, ob, reob)

请求类型为$R^{(5)}$：主体s请求创建类型为t的新客体ob，reob为相关客体。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) (*SDom*(s), t, *creat*)∈*DTM*

(b) $creat \in MLS\_rule(SL(s), SL(reob))$

如果条件被满足，决策输出YES．下一个状态变化如下：

- $O^* = O \cup \{ob\}$
- 其他变量保持不变。

如果上述条件不成立，决策输出NO；下一状态保持原状．

规则 6 delete_object(s, o)

请求类型为$R^{(6)}$：主体s请求删除客体o。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $(o, delete) \in Rolecap(SR(s))$

(b) $delete \in DTM(SDom(s), OT(o)) \wedge delete \in MLS\_rule(SL(s), OL(o))$

(c) $\forall s \in S, a \in P: (s, o, a) \notin b$

如果条件(a)-(c)同时满足，决策输出YES．下一个状态变化如下：

- $O^* = O \backslash \{o\}$
- 其他变量保持不变。

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 7 request_transition(s, d)

请求类型为$R^{(7)}$：进程s请求转换到域d。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $SDom(s) \neq d$

(b) $transfer \in DDI(SDom(s), d) \wedge d \in RDom(SR(s))$

(c) $\forall o \in O, a \in AP:$ if $a \in DTM(SDom(s), OT(o))$ then $(s, o, a) \notin b$

(d) $\forall s1 \in S, i \in IP:$ if $i \in DDI(SDom(s), SDom(s1))$ then $(s, s1, i) \notin c$

(e) $\forall s1 \in S, i \in IP:$ if $i \in DDI(SDom(s1), SDom(s))$ then $(s1, s, i) \notin c$

如果条件(a)-(e)同时满足，决策输出YES。下一个状态变化如下：

- $SDom^* = (SDom \backslash \{(s, SDom(s))\}) \cup \{(s, d)\}$
- 其他变量保持不变。

如果上述条件不成立，决策输出NO; 下一状态保持原状。

规则 8 request_change_role(s, r, d)

请求类型为$R^{(8)}$：主体s请求转换到角色r，同时将域切换到d。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $r \in UR(user(s)) \wedge d \in RDom(r)$

(b) $\forall o \in O, a \in AP$: if $(a, o) \in Rolecap(SR(s))$ then $(s, o, a) \notin b$

(c) $\forall o \in O, a \in AP$: if $a \in DTM(SDom(s), OT(o))$ then $(s, o, a) \notin b$

(d) $\forall s1 \in S, i \in IP$: if $i \in DDI(SDom(s), SDom(s1))$ then $(s, s1, i) \notin c$

(e) $\forall s1 \in S, i \in IP$: if $i \in DDI(SDom(s1), SDom(s))$ then $(s1, s, i) \notin c$

如果条件(a)-(e)满足，决策输出YES．下一个状态变化如下：

- $SR^* = (SR \setminus \{(s, SR(s))\}) \cup \{(s, r)\}$
- $SDom^* = (SDom \setminus \{(s, SDom(s))\}) \cup \{(s, d)\}$
- 其他变量保持不变。

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 9 add_role(s, r)

请求类型为$R^{(9)}$：主体s请求添加角色r。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $SR(s)$=secadmin_r ∧ $SDom(s)$=secadmin_d

(b) $r \notin R$

如果条件(a)和(b)同时满足，决策输出YES．下一个状态变化如下：

- $R^* = R \cup \{r\}$
- 其他变量保持不变。

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 10 add_domain(s, d)

请求类型为$R^{(10)}$：主体s请求添加域d。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $SR(s)$=secadmin_r ∧ $SDom(s)$=secadmin_d

(b) $d \notin D$;

如果条件(a)和(b)同时满足，决策输出YES．下一个状态变化如下：

- $D^* = D \cup \{d\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 11 add_type(s, t)

请求类型为$R^{(11)}$：主体s请求添加类型t。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $SR(s)$=secadmin_r ∧ $SDom(s)$=secadmin_d

(b) t $\notin T$;

如果条件(a)和(b)满足，决策输出YES．下一个状态变化如下：

- $T^*=T \cup \{t\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 12 delete_role(s, r)

请求类型为$R^{(12)}$：主体s请求删除角色r。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(b) r$\in R$

(c) $\forall$ s$\in S$, $SR(s)\neq$r

(d) $\forall$ u$\in U$, (u, r) $\notin UA$

如果条件(a)-(d)同时满足，决策输出YES．下一个状态变化如下：

- $R^*=R \setminus \{r\}$
- $\forall$ sl$\in SL$, (r, sl)$\in RL$, $RL^*=RL \setminus \{(r, sl)\}$
- $\forall$ d$\in D$, (r, d)$\in RD$, $RD^*=RD \setminus \{(r, d)\}$
- $\forall$ $cap\in CAP$, (r, cap)$\in RCAP$, $RCAP^*=RCAP \setminus \{(r, cap)\}$
- 其他变量保持不变。

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 13 delete_domain(s, d)

请求类型为$R^{(13)}$：主体s请求删除域d。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(b) d$\in D$

(c) $\forall$ s$\in S$, $SDom(s) \neq$ d

(d) $\forall$ r$\in R$, (r, d) $\notin RD$

如果条件(a)-(d)同时满足，决策输出YES．下一个状态变化如下：

- $D^*=D \setminus \{d\}$
- $\forall$ t$\in T$, ap$\in 2^{AP}$, $DTM^*=DTM \setminus \{(d, t, ap)\}$
- $\forall$ d1$\in D$, ip$\in 2^{IP}$, $DDI^*=DDI \setminus \{(d, d1, ip)\}$
- $\forall$ d1$\in D$, ip$\in 2^{IP}$, $DDI^*=DDI \setminus \{(d1, d, ip)\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 14 delete_type(s, t)

请求类型为$R^{(14)}$：主体s请求删除类型t。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $SR(s)=secadmin\_r \wedge SDom(s)=secadmin\_d$

(b) $t \in T$

(c) $\forall o \in O, OT(o) \neq t$

如果条件(a)-(c)同时满足，决策输出YES．下一个状态变化如下：

- $T^*=T \setminus \{t\}$
- $\forall d \in D, ap \in 2^{AP}, DTM^*=DTM \setminus \{(d, t, ap)\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 15 add_DTM(s, (d, t, a))

请求类型为$R^{(15)}$：主体s在域-类型访问控制矩阵中为d域对t类型的访问方式中添加a方式。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $d \in D \wedge t \in T \wedge a \in OP$

(b) $SR(s)=secadmin\_r \wedge SDom(s)=secadmin\_d$

(c) $a \notin DTM(d, t)$

如果条件(a)-(c)同时满足，决策输出YES．下一个状态变化如下：

- $DTM^*=(DTM \setminus \{(d, t, DTM(d, t))\}) \cup \{(d, t, (DTM(d, t) \cup a))\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 16 add_DDI(s, (d1, d2, i))

请求类型为$R^{(16)}$：主体s在域-域交互控制矩阵中为d1域对d2域的交互方式中添加i方式。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $d1 \in D \wedge d2 \in D \wedge i \in IP$

(b) $SR(s)=secadmin\_r \wedge SDom(s)=secadmin\_d$

(c) $i \notin DDI(d1, d2)$

如果条件(a)-(c)同时满足，决策输出YES．下一个状态变化如下：

- $DDI^*=DDI \setminus \{(d1, d2, DDI(d1, d2))\} \cup \{(d1, d2, (DDI(d1, d2) \cup i))\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 17 del_DTM(s, (d, t, a))

请求类型为$R^{(17)}$：主体s在域-类型访问控制矩阵中删除d域对t类型的a访问方式。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $d \in D \ \wedge \ t \in T \ \wedge \ a \in DTM(d, t)$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $\forall s \in S, o \in O$: if $SDom(s)=d \wedge OT(o)=t$ then $(s, o, a) \notin b$

如果条件(a)-(c)同时满足，决策输出YES。下一个状态变化如下：

- $DTM^*=DTM \setminus \{(d, t, DTM(d, t))\} \cup \{(d, t, (DTM(d, t) \setminus a))\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 18 del_DDI(s, (d1, d2, i))

请求类型为$R^{(18)}$：主体s在域-域交互控制矩阵中删除d1域对d2域的a交互方式。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $d1 \in D \ \wedge \ d2 \in D \ \wedge \ i \in DDI(d1, d2)$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $\forall s1 \in S, s2 \in S,$ if $SDom(s1)=d1 \wedge SDom(s2)=d2$ then $(s1, s2, i) \notin c$

如果条件(a)-(c)同时满足，决策输出YES。下一个状态变化如下：

- $DDI^*= DDI \setminus \{(d, t, DDI(d1, d2))\} \cup \{(d, t, (DDI(d1, d2) \setminus i))\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 19 add_role_permission(s, (t, a, r))

请求类型为$R^{(19)}$：主体s在给角色r添加权限(a, t)。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $r \in R \ \wedge \ t \in O \cup S \ \wedge \ a \in P$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $(r, (a, t)) \notin RCAP$

如果条件(a)-(c)同时满足，决策输出YES。下一个状态变化如下：

- $RCAP^*= RCAP \cup \{(r, (a, t))\}$

- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 20 delete_role_permission(s, (t, a, r))

请求类型为$R^{(20)}$：主体s删除该角色r的权限(a, t)。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $r \in R \ \wedge \ t \in S \cup O \ \wedge \ a \in P$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $\forall s \in S, t \in S \cup O$: *if* $SR(s)=r \wedge (r, (a, t)) \in RCAP$ then $(s, t, a) \notin b \wedge (s, t, a) \notin c$

如果条件(a)-(c)同时满足，决策输出YES。下一个状态变化如下：

- $RCAP^* = RCAP \backslash \{(r,(a, t))\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 21 add_user_role(s, (u, r))

请求类型为$R^{(21)}$：主体s把角色r分配给用户u。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $r \in R \ \wedge \ u \in U$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $(u, r) \notin UA$

如果条件(a)-(c)同时满足，决策输出YES。下一个状态变化如下：

- $UA^* = UA \ \cup \{(u, r)\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 22 delete_user_role(s, (u, r))

请求类型为$R^{(22)}$：主体s删除分配给用户u的角色r。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $(u, r) \in UA$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $\forall s \in S, SR(s) \neq r$

如果条件(a)-(c)同时满足，决策输出YES。下一个状态变化如下：

- $UA^* = UA \backslash \{(u, r)\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 23 add_role_domain(s, (r, d))

请求类型为$R^{(23)}$：主体s把域d分配给角色r。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $r \in R \ \wedge \ d \in D$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $(r, d) \notin RD$

如果条件(a)-(c)同时满足，决策输出YES。下一个状态变化如下：

- $RD^* = RD \cup \{(r, d)\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 24 delete_role_domain(s, (r, d))

请求类型为$R^{(24)}$：主体s把分配给角色r的域d撤销。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $r \in R \ \wedge \ d \in D$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $(r, d) \in RD$

(d) $\forall s \in S, \quad if\ SR(s)=r,\ then\ SDom(s) \neq d$

如果条件(a)-(d)同时满足，决策输出YES。下一个状态变化如下：

- $RD^* = RD \backslash \{(r, d)\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

规则 25 change_type(s, (o, t))

请求类型为$R^{(25)}$：主体s请求改变客体o的类型为t。如果请求方式不恰当，决策输出ILLEGAL，下一状态保持原状；否则，检验如下条件：

(a) $o \in O \ \wedge \ t \in T$

(b) $SR(s)$=secadmin_r $\wedge$ $SDom(s)$=secadmin_d

(c) $\forall s \in S, a \in SP: \quad if \quad a \in DTM(SDom(s), OT(o)) \quad then \quad (s, o, a) \notin b$

如果条件(a)-(c)满足，决策输出YES．下一个状态变化如下：

- $OT^* = (OT \backslash \{(o, OT(o))\}) \cup \{(o, t)\}$
- 其他变量保持不变

如果上述条件不成立，决策输出NO；下一状态保持原状。

定理 3.2 本模型中定义的25条迁移规则是安全的。

证明 要证明这个定理仅需根据每条规则的条件，逐一验证规则转换后状态是安全的并且状态转换过程满足域转换安全属性即可。由于验证方法是类似的，限于篇幅，本文只给出规则1和规则7的安全性验证。

规则1验证 假设满足规则条件(a)和(b)，规则前后状态分别是v和v*，v为安全状态。由规则知v*与v的唯一区别是集合b发生了变化: b*\b={(s, o, a)}。因为满足条件(a)和(b)，所以(s, o, a)满足域安全属性和多级安全属性，并且同时至少还满足自主安全属性或角色安全属性之一，由于v是安全状态，所以v*是安全状态。此外，没有任何主体的域发生了转换，因此v到v*的状态转换满足域转换安全属性。由此得出该状态转换规则是安全的。

规则7验证 假设满足规则条件(a)和(b)，规则前后状态分别是v和v*，v为安全状态。因为满足条件(a)和(c)-(e)，v*与v的唯一区别是关系集合*SDom*发生了变化：$\exists s\in(S^*\cap S)$, $SDom(s)\neq SDom^*(s)$。由于满足条件(b)，$transfer\in DDI(SDom(s), SDom^*(s))$，所以状态转换满足域转换安全属性。同时，由于$SDom^*= (SDom\backslash\{(s, SDom(s))\})\cup\{(s, d)\}$，因此 $dom(SDom^*)=S$, $SDom^*(s)\in RDom^* (SR^*(s))$，满足限制条件2。状态的其他集合和变量没有发生变化，且v是安全状态，因此v*是安全状态。由此得出状态转换规则7是安全的。

第四章 原型系统实现

4.1 系统结构

本文构建的 MPVSM 模型目前正在我们的原型操作系统 Nutos 中使用，Nutos 原型系统在 Minix3.0[Herder 06-1, Herder 06-2]操作系统基础上构建。Minix3.0 是基于微内核的操作系统，包含运行于内核态的微内核，提供系统的基本运行机制和功能，运行于用户态的资源管理、进程管理、网络管理等操作系统服务进程以及驱动程序。在 Minix 3.0 的基础上，我们添加了一个运行于用户态的策略服务器，负责管理维护系统中的所有与强制访问控制决策相关的安全策略和安全属性信息。同时在原来的资源管理、进程管理、网络管理进程中添加引用监控器模块。为了提高系统的性能，还对其他模块包括微内核进行了很多的修改和重设计。系统结构如图 4-1 所示。

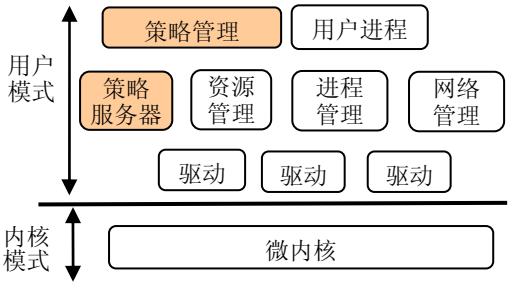

图 4-1 Nutos 系统结构图

微内核为上层提供最基本的功能和安全机制。进程管理负责管理系统中的主体（进程）的加载、调度、运行、交互、结束等过程，对主体的角色、域转换和主体之间的交互实施控制。资源管理器管理系统中的所有客体资源包括文件、设备等静态客体和 socket 等动态客体，并对其实施访问控制。系统中的所有安全策略信息、主体及客体的安全属性都由安全服务器来维护，策略服务器根据主客体的安全属性及安全策略来作出访问控制决策。驱动程序进程与设备交互，可以根据设备的使用动态加载或结束。

4.2 策略服务器

4.2.1 主要功能

策略服务器(Policy Server)是整个系统安全功能的核心，是系统中安全决策中心。它为系统的访问控制提供安全决策，管理维护系统中的各种主客体安全属性，安全策略信息。

策略服务器为系统中的所有资源包括文件、设备等静态客体和 socket 等动态客体以及进程等主体分配与管理安全属性。并以此属性为据，根据系统的安全策略做出访问决策。

策略服务器功能包括为进程管理、资源管理进程提供访问控制决策支持，维护系统安全策略信息，提供添加和修改系统安全策略的接口。具体包括：

(1) 维护系统安全策略信息
(2) 维护系统中所有主体的安全属性
(3) 维护所有客体的安全属性
(4) 为主体对资源的访问提供访问控制决策
(5) 为进程之间的相互交互提供交互决策支持
(6) 为进程的安全属性转换提供接口
(7) 为资源的安全属性改变提供接口
(8) 提供添加和修改系统安全策略的接口

4.2.2 策略服务器功能动态过程

4.2.2.1 用户进程读取客体

图 4-2 给出了用户进程进行客体（文件）读取操作时，操作系统的处理流程。其中关键步骤如下：

①：用户进程请求访问资源
②：把请求交给引用监控器
③：引用监控器收集主体和客体的安全属性，发送给策略安全服务器请求访问控制决策
⑤：读取客体资源
⑥：资源管理进程调用相关驱动进程读取相应物理设备
⑦：驱动进程请求微内核进行实际的 I/O 操作

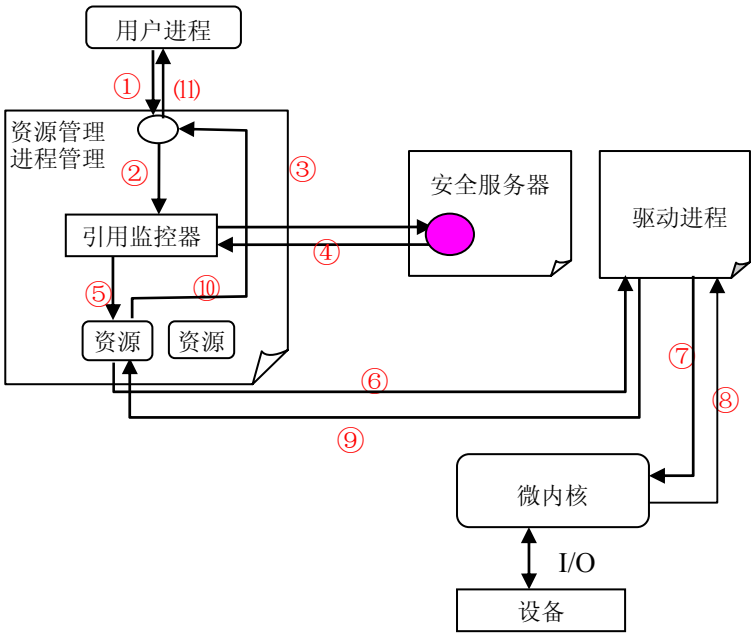

图 4-2 用户进程读取客体资源流程图

4.2.2.2 修改系统安全策略

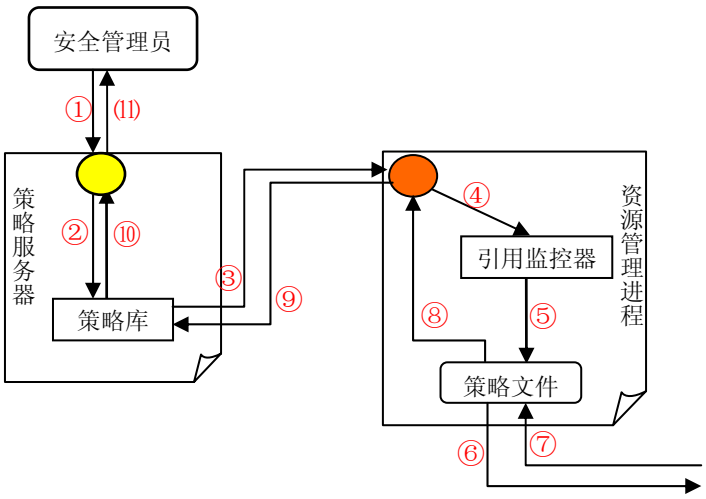

图 4-3 修改系统安全策略流程图

图 4-3 给出了安全管理员进程对系统的安全策略进行配置修改操作时，操作系统的

处理流程。其中关键步骤如下：

①：安全管理员进程调用安全服务器系统调用接口

②：安全服务器检查策略修改文件的合法性及一致性，满足后修改安全服务器种的策略库

③：安全服务器请求资源管理进程将新修改的安全策略保存到策略文件中

④：请求经过引用监控器，引用监控器判断为安全服务器调用，跳过安全检查

⑤：读取策略文件

4.2.3 策略服务器结构

策略服务器负责解释和维护系统中的所有主体和客体的安全属性。并以此属性为据，根据系统的安全策略做出访问控制决策。修改主体和客体安全属性的工作也由安全服务器来完成，策略服务器提供修改安全属性的接口供其他功能模块进程调用。

策略服务器内部结构如图 4-4 所示。策略服务器提供的接口包括：访问控制决策接口；创建安全属性接口；修改安全属性接口；策略配置接口。

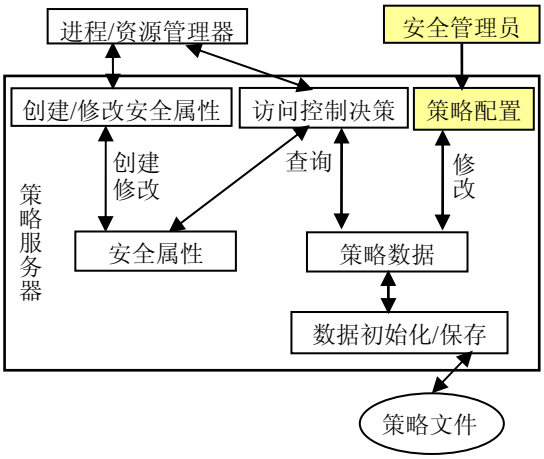

图 4-4 策略服务器内部结构设计

系统的安全策略信息以文件形式保存在文件系统的/security/文件夹下，/security 文件夹对用户不可见，只有安全服务器进程才有权读写该文件夹下的配置文件。当安全服务器启动时，读入安全信息并建立策略数据库，初始化各种元素表。当安全管理员修改安全策略配置后，首先修改安全服务器中对应的数据结构表，然后再把修改后的安全策略保存回策略文件中。

4.2.4 策略服务器接口

1）访问控制决策接口

功能：对系统中的进程访问文件或进程间通信时的访问控制提供访问决策。

调用者：进程管理和资源管理。

2）创建安全属性接口

功能：为系统中新创建的主体（进程）和客体（文件）创建对应的安全属性。

调用者：进程管理和资源管理。

3）修改安全属性接口

功能：为系统中现存的主体（进程）和客体（文件）提供修改对应安全属性，如修改客体安全属性、进程的角色变换、进程的域迁移等。

调用者：进程管理和资源管理。

4）安全策略配置接口

功能：配置系统的安全策略。

调用者：拥有安全管理员角色并处于安全管理域的特权进程。

4.2.5 使用的其它模块接口

资源管理模块提供给安全服务器的接口

open、read、write、unlink。调用资源管理器正常的文件打开/创建/读/写/删除等接口。资源管理器在这些接口函数处理时判断调用者，如果调用者为策略服务器，则绕过安全检查，安全策略信息文件存放在系统根目录的/security 子目录下，这些接口只能对该子目录下的文件进行操作。

4.2.6 安全属性存放

4.2.6.1 安全策略信息存放

系统安全策略信息以 XML 格式的文件保存在硬盘中。每个文件系统的静态安全信息保存在该文件系统的/security/文件夹下。/security 文件夹对用户不可见。

表 4-1 /security/下对应的安全策略文件列表

文件名	文件内容
user.sec	用户配置信息
role.sec	角色配置信息
domain.sec	域配置信息
type.sec	客体类型配置信息
mls.sec	安全标记信息
ddt.sec	域/类型访问控制矩阵
dit.sec	域交互控制矩阵

4.2.6.2 安全属性信息存放

客体的安全属性保存在客体文件对应的索引节点结构和相应的扩展磁盘块中。主体的安全属性在系统运行过程中动态建立。进程管理模块在每个进程的描述符中保存一个安全标识符，策略服务器负责将该标识符映射到对应进程的安全属性结构。

主体（进程）安全属性在系统运行时动态建立，无需存放在磁盘中。客体（文件）对应的安全属性保存于每个文件的索引节点（inode）的相关扩展域中，如果该扩展区域空间不够存放安全属性，则重新分配磁盘块存放，在索引节点中保存指向该磁盘块的指针（详见资源管理模块详细设计）。结构见下图：

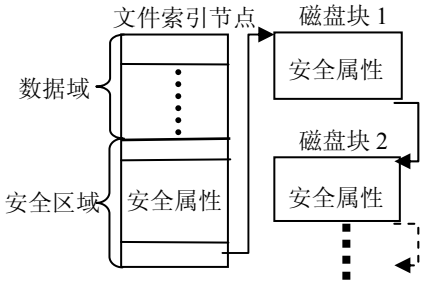

图 4-5 索引节点结构图

4.2.6.3 未标记文件系统的处理

如果挂接的文件系统没有标记安全属性，则以挂接点目录的安全属性作为被挂接文件系统根目录的安全属性。

4.2.6.4 无安全属性客体的处理方式

某个文件如果没有对应安全属性，以其所在目录的安全属性为默认属性，依次类推。

4.2.7 安全策略文件表示

策略服务器中的安全策略信息在系统关闭时以 XML 格式存储于静态存储器中。MMVSF 中一些元素的 XML 表示形式如下例所示：

```
<?XML version="1.0" ?>
<Nutos_Secutity_elements
xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
xsi:SchemaLocation="mmvsf.xsd">
  <user uid=”admin_u” description=”system administrator” />
  <role rid=”user_r” description=”general role” />
  <role rid=”sysad_r” description=”Administrator role” />
  <domain did=”syscon_d” description=”system configure domain” />
  <domain did=”gen_d” description=”general domain” />
  <type tid=”syslog_t” description=”system configure type” />
  <type tid=”gen_t” description=”general file type” />
  <conflab cid=”unclass_c” description=”unclassfied” />
  <conflab cid=”sec_c” description=”secret” />
  <intlab iid=”sh_i” description=”share” />
  <intlab iid=”tin_i” description=”top integrity” />
  <accmode mid=”r” description="read" type=”object” />
  <intmode mid=”sig_k” description="sig kill" type=”process” />
  <intmode mid=”tran” description="transform" type=”process” />
</Nutos_Secutity_elements>
```

用户-角色分配关系的 XML 表示形式如下：

```
<user_role_assign>
    <user>admin_u</user>
    <role>user_r</role><role>sysad_r</role>
<user_role_assign>
```

角色-域授权关系和角色安全属性的表示形式如下：

```
<role_attrbite>
    <role>sysad_r</role>
    <domain>sysconf_d</domain><domain>genr_d</domain>
    <conflab>unclass_c</conflab><intlab>tin_i</intlab>
    <permission>
        <accmode>r</accmode><object>/root/log.t</object>
    </permission>
```

```
        <permission>
            <intmode>sig_k</amode><subject>10001</subject>
        </permission>
    </role_attrbite>
```

DTM 和 *DDI* 矩阵中的元素表示如下：

```
    <domain_type_item>
        <domain>sysconf_d</domain>
        <type>syslog_t</type>
        <accmode>r</accmode><accmode>w</accmode>
    </domain_type_item>
    <domain_domain_item>
        <domain>sysconf_d</domain>
        <domain>gen_d</domain>
        <intmode>sig_k</intmode><intmode>tran</intmode>
    </domain_domain_item>
```

4.3 引用监控器

Nutos 中的引用监控器实现为文件服务器、进程管理和网络管理进程中的一个不可被旁路的模块。引用监控器截获用户进程访问系统与网络资源或与其它进程交互的请求，获取相关资源安全属性，发送访问控制决策请求给策略服务器，等待策略服务器的决策结果，并根据结果允许或禁止用户进程的操作。整个过程如图 4-6 所示。

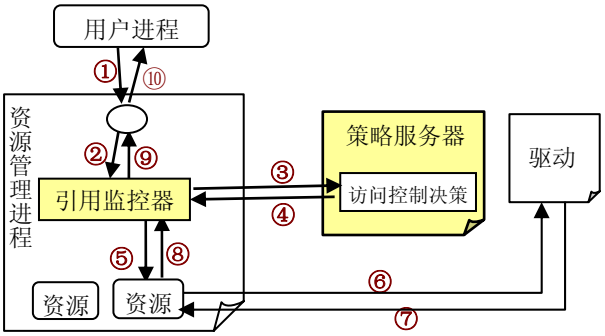

图 4-6 访问控制图

4.4 系统完整性的层次保障分析

4.4.1 微内核结构对系统完整性保护的分析

微内核结构使得微内核和各个功能服务进程处于不同的地址空间，其中微内核的段页表管理和消息传递是微内核提供的最基本的安全机制。段页表实现了进程物理地址空间上的隔离，使得直接跨进程的读写操作在硬件级上被禁止。将段表页表保存在微内核中，其他操作系统服务进程或用户进程不能通过篡改段表页表来破坏微内核的进程内存空间隔离功能。微内核保证了进程的区域隔离，只要微内核的完整性得到保证，则功能服务进程的内存空间中运行代码的完整性就可以得到保证。

微内核控制着进程之间消息的实际发送过程。可以对系统各功能服务进程进行标识，微内核以此标识来控制用户进程与功能服务进程及功能服务进程之间的消息传递。只要微内核的完整性得到保证，恶意进程则无法通过恶意消息或冒充功能服务进程来破坏系统的完整性。以这种方式，微内核为功能服务进程之间的消息传递提供了一条可信的数据传递路径。

Nutos 系统基于微内核结构，微内核运行于内核特权模式，提供基本的功能和安全机制，系统的大部分功能通过运行在用户模式的系统功能服务进程（简称系统进程）实现。操作系统的功能实际上是由很多逻辑模块来共同完成的，典型划分包括：进程管理、内存管理、文件系统、设备管理、网络系统等。Nutos 系统采用了另外的功能分类，操作系统的主要功能是管理系统中的主体（进程）和客体（系统中的各种资源）并控制主体对客体的访问及主体之间的交互。基于这种思想，Nutos 系统功能划分为微内核、进程管理、资源管理和策略安全服务器及单独的驱动程序。微内核为上层提供最基本的功能和安全机制。进程管理主要负责管理和控制进程的加载、调度、结束及进程间的交互以及进程的虚拟地址空间的分配与管理。资源管理器管理系统中的资源包括文件、设备等静态客体和 socket 等动态客体并对其实施访问控制。策略安全服务器则负责整个系统的安全决策，并管理维护系统的安全策略信息。驱动程序进程与物理设备交互，根据设备的使用动态加载或结束。

其中微内核的段页表管理和消息传递既是最基本的系统功能，也是微内核提供的最基本的安全机制，是整个操作系统乃至系统平台上各种应用安全的基石。段页表实现了进程物理地址空间上的隔离，使得直接跨进程的读写操作在硬件级上被禁止。将段表页表保存在微内核中，其他操作系统服务进程或用户进程不能通过篡改段表页表来破坏微

内核的进程地址空间隔离机制。由于微内核保证了安全区域隔离，资源管理进程不能篡改进程管理的代码和数据，同样进程管理也不能通过破坏资源管理进程来访问各种安全敏感文件，从而形成了有效的隔离保护体系，不会因为个别系统模块的漏洞导致整个系统安全机制被破坏。

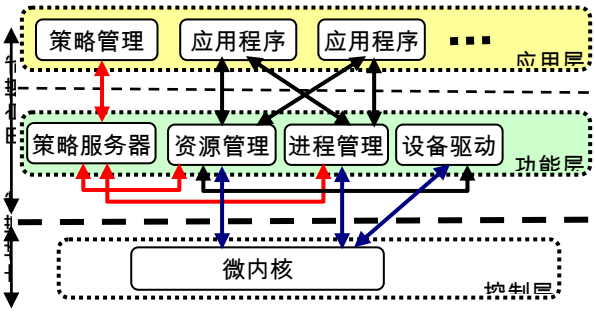

图 4-7 Nutos 系统消息流

消息是用户进程与系统进程以及服务进程与安全微内核之间唯一的通信方式。用户进程通过发送消息给服务进程的方式请求操作系统功能服务，服务进程发送消息给微内核请求微内核的功能服务，服务进程相互之间发送消息请求功能服务，调用返回值通过回复消息返回给发送进程。图 4-7 中的箭头表示进程之间允许的消息流。消息发送过程由微内核控制，每条消息都包含发送进程和接受进程的标识，只有指定进程间的消息流才被允许。微内核为每个功能服务进程分配唯一的标识符，对每条消息的发送进程和接收进程标识进行检查并控制是否允许消息发送。这样，可以有效地保障进程间交互通道的安全性。

在操作系统各系统进程实现物理隔离的基础上，利用域对操作系统内核各功能模块进行划分。系统中定义的域定义表和域交互表分别描述各个域对不同类型客体和各个域之间的访问权限。

我们把 Nutos 的服务进程和微内核划分到不同的域中，一种划分方法见图 4-8，每个系统服务进程包括微内核都有一个对应的域：安全服务器进程处于 secser_d 域，文件系统进程处于 rm_d 域，进程管理进程处于 pm_d 域，所有驱动进程都处于 driver_d 域中，安全微内核则在 kernel_d 中。操作系统自身运行所需要的资源如重要的数据结构、代码数据文件以及配置文件划分为对应的不同的类型。安全策略指定域对类型的访问操作权限。如，pm_t 类型的文件只能供 pm_d 域中的进程访问，其他任何域对该类型的文件没有任何访问权限。同时，安全策略指定域与域之间的交互规则。一个域中的进程是否允许另外一个域中的进程发送消息请求服务由系统的策略决定。这样，系统服务进程运行

所需要的数据文件和配置文件就不会被非法修改，从而服务进程的行为不会被恶意进程所改变，保证了系统服务进程的完整性。同时，用户消息的传递在域交互规则控制下，必须经过指定路径去访问系统的资源，防止用户绕开资源管理进程而直接访问驱动程序而读写系统文件，保证了进程间通信信道的完整性。

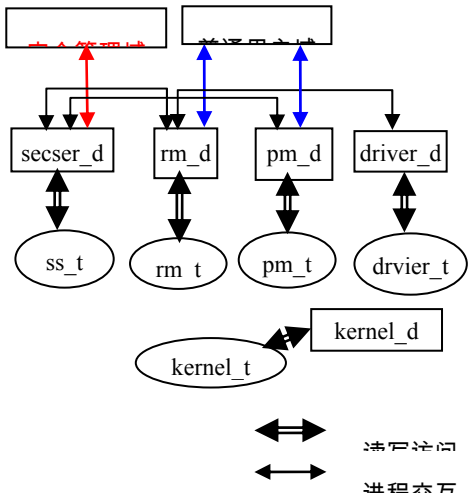

图 4-8 系统进程域的划分

系统的安全策略规则只有处于安全管理域中的进程才能修改，而由内核的隔离机制保证了安全管理域进程运行代码的完整性。这样，就防止了除了安全管理员之外的非法用户进程对系统安全策略的修改，保障了系统安全策略信息的完整性。

这样，从微内核的安全隔离机制，层层往上，就形成了一个纵深的完整性保护体系。

4.4.2 分权和最小特权的实现

职责分离和最小特权是两个重要的安全原则，前者是指系统分配给用户的权限是用户完成工作所必需的权限的最小集合；后者是指系统中不同的职责尽量由不同的用户承担，避免权限过度集中。这个定义在操作系统的各个功能模块上同样适用。在 Nutos 系统中，微内核提供的功能服务进程隔离机制使得操作系统各功能模块之间的职责分离和最小特权原则可以得到有效的实施。其中，微内核只实现基本的安全机制和系统功能，并不关心具体的策略。而进程管理只负责系统中进程活动周期的管理及安全决策的实施。资源管理负责管理系统中的所有资源并对资源实施访问控制。安全策略服务器管理系统的安全策略信息和系统所有资源的安全属性，并对系统的访问控制提供控制决策。微内核是机制的提供者，安全策略服务器是访问控制的决策者，资源管理或进程管理则

是访问控制的实施者。在职责分离的基础上，对每个服务进程拥有的权限加以分析，控制，就可以实现每个服务进程的最小特权。

同时，安全微内核只提供实现操作系统功能和安全的基本机制，而上层服务进程实现各种功能和安全策略。这样有几点好处：第一、机制和策略的分开使得微内核实现起来很小也很灵活，即提高系统效率，也便于对微内核进行形式化验证。第二、机制和策略的分开有利于灵活性和可移植性，安全微内核提供最基本机制，上层服务可以实现不同的策略，改变策略只要修改或更换服务进程即可，甚至可以同时存在多个版本的服务进程，从而同时实现多种策略。

4.4.3 引用监控器的不可旁路性

利用进程多保护域隔离机制[4，5，6]对资源管理进程中的代码和数据进行保护域划分。把引用监控器代码和影响引用监控器的数据结构放在单独的段中，然后把这些段放进高特权的保护域中，设置引用监控器为资源管理的入口段。这样，可以有效保护引用监控器代码不可被篡改，更重要的是保护引用监控器不被旁路。结构如图 4-9 所示。

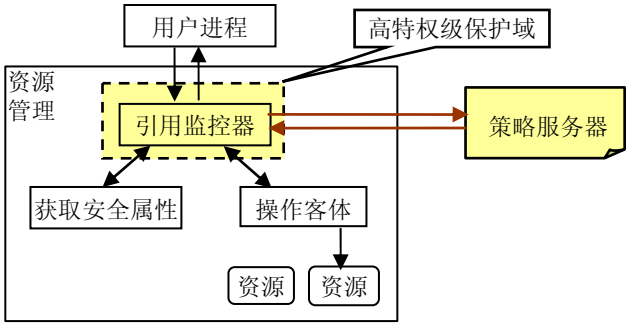

图 4-9 引用监控器保护域划分

4.4.4 防内核模块攻击分析

内核插入模块攻击也是另一种典型的针对 Linux 操作系统的攻击。一个典型的内核攻击例子就是通过内核模块机制插入木马并将其隐蔽起来。攻击者首先获得主机的管理员权限，然后在内核中插入木马程序，木马程序修改系统调用表，截获系统调用来更好的隐藏自己。虽然从 Linux2.4.8 内核之后，系统调用表已经不再对外导出，但是仍然有其他的方法修改系统调用。尽管 Linux 开发者们早就意识到内核模块（Linux kernel Module）机制的不安全性，并且多次改进其使用方式。但是作为系统的一个主要功能，特别是为了提供动态插入驱动程序的功能，系统又无法放弃内核模块机制。

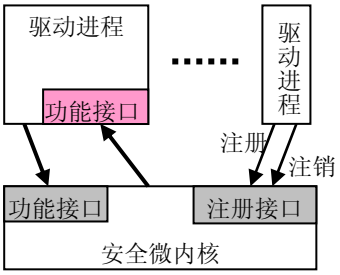

图 4-10 驱动进程动态添加/注销

而在基于微内核结构的 Nutos 系统中，每个驱动程序都以单独的进程运行，并使用安全微内核提供的相应的接口来完成最终的 I/O 操作，同时驱动进程提供功能接口给安全微内核和其他系统服务进程，过程如图 4-10 所示。每次添加新的驱动进程，都会在安全微内核中进行注册，以供其他服务进程使用。结束驱动进程时会进行注销。动态加载的驱动进程只能访问安全微内核或其他系统服务进程的指定接口。这样，动态添加设备驱动只需要添加驱动进程并且在内核注册即可，无须在微内核中提供插入模块支持。既解决了动态加载驱动程序的功能需求，又不会产生因为加载内核模块带来的许多安全问题。插入模块攻击对该系统也无法奏效。

4.4.5 内核缓冲区溢出攻击的防止

缓冲区溢出是指通过往程序的缓冲区写超出其长度的内容，造成它的溢出，从而破坏程序的数据区域(堆或者栈、或者静态数据区等)，使程序转而执行其它指令，以达到攻击的目的。

引起缓冲区溢出的实质原因是在大多数体系结构的计算机中，程序的代码区和数据区并不完全隔离，在 CPU 看来，代码区和数据区并没有太大的区分，均可以当成代码执行。因此控制程序执行的关键数据结构易受数据区的影响。当前许多重要的系统都是由 C / C++开发的，由于 C / C++是一种效率优先的语言，对数据类型检查较弱，程序风格自由，并且数组的越界在编译器中不进行检查，如果开发者对数组和指针的引用不进行边界检查，就有可能导致缓冲区溢出漏洞的产生。如果这些漏洞再被攻击者加以利用，轻则导致程序崩溃，产生拒绝服务攻击；重则导致操作系统的权限泄漏，使入侵者控制整个计算机系统等。缓冲区溢出攻击使任何人都有可能取得主机的控制权，所以它代表了一类极其严重的安全威胁。而针对操作系统内核的缓冲区溢出类攻击则是黑客对类 Unix 系统(如 Unix，Linux，FreeBSD 等)和 Windows 系统进行系统攻击的一种主要手

段[1]。

如上所述，缓冲区溢出攻击问题存在的原因是在当前计算机体系结构和主流操作系统中，数据段和堆栈段都是可以执行的。但是，如果简单的禁止数据段和代码段的执行又会带来很多功能上的限制。因此在 Nutos 系统中，微内核和各功能模块进程中都增加了独立的数据段用来接收外来的数据或参数，该数据段在进程地址空间中的权限设置为不可执行，内核可以针对某些外来数据进行无害化处理后再进一步使用。如此，可以解决大多数针对微内核和系统进程的缓冲区溢出攻击。

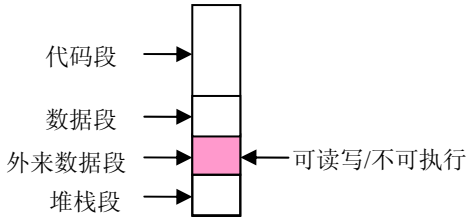

图 4-11 进程地址空间分段示意图

4.5 系统性能改进

由于 Nutos 采用的是微内核的操作系统结构，因此对性能会有一定的影响。但是我们在设计和实现的过程中参考采用了很多提高性能的技术和设计方法，包括改善微内核 IPC 的相关技术[Liedtke 93, Liedtke 93]，以及引用监控器和安全服务器间多消息捆绑发送方法等，使得最终的性能有很大的改善。最终系统的性能分析见 4.6，性能分析结果也显示了我们的设计的可用性。

4.5.1 大数据传递

在服务进程之间和服务进程与用户进程之间经常需要传递大块的数据，如文件读写数据、网络数据包等。而进程之间的地址空间是相互独立的，无法直接访问其他进程的地址空间，而大块数据的多次内存拷贝又会显著降低系统的性能。

Nutos 系统中服务进程有多个数据段，这些段的描述符都保存在相应进程的表项中。每个进程提供一个共享数据段，发送进程将数据拷贝到自己的共享数据段，再发送消息时请求内核将该数据段对应的物理内存空间映射到目标进程地址空间，目标进程就可以直接读取该空间的数据。这样避免了数据多次在内存之间的拷贝。

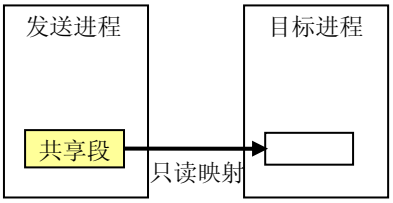

图 4-12 进程间共享数据段

4.6 系统性能分析

表 4-2 几个系统的性能对比(单位：微秒)

系统调用	Minix 3.0	Nutos	比率
open+ close	3.914	4.215	107.7%
write (1K)	4.795	5.064	105.6%
read (1K)	3.418	3.626	106.1%
read (1 M)	3061.9	3068.0	100.2%
creat	4.873	5.531	113.5%
del	5.634	6.057	107.5%
lseek	1.751	1.823	104.1%
mkdir	7.795	8.933	114.6%

我们分别测试了 Minix3.0 和 Nutos 下相关系统调用的速度，并进行了对比。测试方法为每次循环调用某个系统调用打开或读写不同文件 1000 次到 100000 次，记录整个调用时间，再除以调用次数获得该调用的单位时间。我们对每个系统调用都进行了 50 次以上的调用计算，取平均值作为最终结果。

测试基于 P4 CPU 1.70GHz，256M 内存，40GB IDE 硬盘的硬件环境，所有的测试系统都是空闲系统，没有其他用户进程同时进行。测试结果如表 4-1 所示。最终结果显示系统的性能损失在 10%以内，也显示了我们的设计的可用性。

第五章 多策略模型配置实例

5.1 模型配置实例

5.1.1 操作系统与用户进程交互的配置实例

我们利用 MPVSM 为 Nutos 设计了用户进程与操作系统的交互控制策略。如图 5-1 所示。用户进程与操作系统的数据交互通过各自的 buffer 数据存储对象来完成。用户发送数据给操作系统时，将这些数据写入操作系统的 buffer 区域，操作系统回复用户进程时，将回复数据写入用户的 buffer 区域即可。不允许用户进程写操作系统的其它数据区域。

为实现以上策略，我们为每个模块分配一个安全属性{role, domain}，role 表示运行角色，domain 表示运行域。为每个存储对象设置相应的安全属性，安全属性{(c, i), t)表示对象的机密性级别为 c，完整性级别为 i，类型为 t。其中，*Rolecap*(usr_r)={(*w*, kerbuffer)}，表示角色 usr_r 拥有写 kerbuffer 数据对象的权限，角色 usr_r 的安全级别为(0,1)，授权域为{usr_d}。*Rolecap*(ker_r)= Φ，ker_r 没有角色权限，角色 ker_r 的安全级别为(0,2)，授权域为{ker_d}。其中的域-类型和域-域控制矩阵如表 5-1 所示。

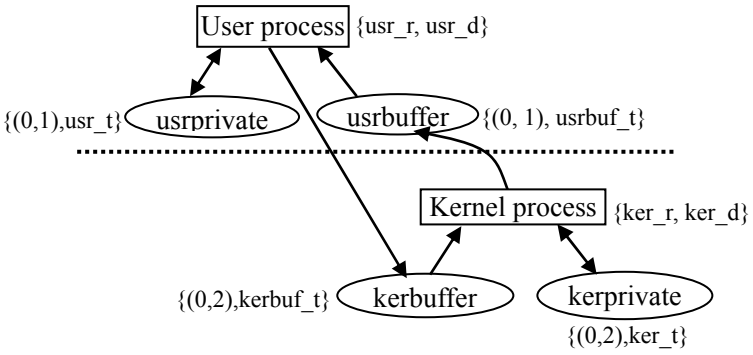

图 5-1 用户进程-操作系统交互模型

表 5-1 DTM 矩阵

	ker_t	kerbuf_t	usr_t	usrbuf_t

ker_d	*r,w*	*r*		*w*
usr_d			*r,w*	*r*

通过不同的安全级别把内核私有数据空间和用户数据空间隔离开来，同一安全级别的 usrbuffer 和 usrprivate 通过划分到不同类型，使得 User process 对两个数据空间的权限不同。而根据安全级别限制，User process 无法对 kerbuffer 进行写操作，而这个操作又是功能需要，因此，赋予用户角色 usr_r 单独的写 kerbuffer 的权限，该权限独立于 MLS 和域权限，即满足了写多安全级别信息的需要，又不用担心 User process 获得过多的权限而带来安全隐患。

5.1.2 防火墙配置实例

我们还设计了一个第二章中提到的简单的防火墙系统信息流策略模型，如图 5-2 所示。该防火墙系统的安全策略要求从外网进入系统的数据必须通过访问控制模块的检查才能进入内网，反之亦然。在模型中该策略的解释为：所有从 Outside 模块流入 Inside 模块的信息，和所有从 Inside 模块流入 Outside 模块的信息必须经过 Access control 模块。除此之外，系统还要求所有模块可以读取系统配置信息但不能对其修改；所有模块可以追加日志信息但不能读取该信息。

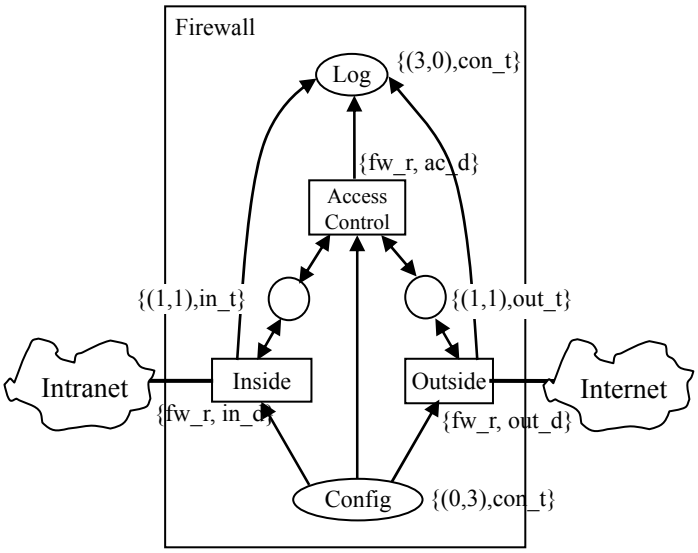

图 5-2 防火墙策略配置实例图

为实现以上信息流策略，我们的配置如图 5-2 所示。其中的域-类型和域-域控制矩阵如表 5-2 所示。其中，*Rolecap*(fw_r)=Φ，角色 fw_r 没有角色权限。角色 fw_r 的安全

级别为(1,1)，授权域为{ac_d, in_d, out_d}，这三个域之间不能相互转换。

通过提高日志信息的机密性使得主体不能读取日志，提高配置信息的完整性使得主体不能修改配置。通过把同一安全级别的主体划分到不同的域，而同一安全级别的客体划分到不同的类型，并通过域-类型访问控制矩阵细化访问权限，以此实现内外网络信息流动通道的控制。

表 5-2 DTM 和 DDI 矩阵

	in_t	out_t	con_t	in_d	out_d	ac_d
in_d	*r,w*		*r,a*			
out_d		*r,w*	*r,a*			
ac_d	*r,w*	*r,w*	*r,a*			

5.2 应用层实施多策略模型视图

MPVSM 模型经过某种退化可以在系统中分别独立实现 BLP、Biba、DTE 和 RBAC 模型的功能。也可以在同一系统的不同用户群之间实现多安全策略视图。下面给出的就是这两种方式的实现方法：

5.2.1 实施多级安全策略

用 MPVSM 模型实现基于格的多级安全模型时，只需按如下方式配置规则：

(1) $|R|=|SL|$，即系统中的角色数量等于系统中的安全标记的种类数。系统中的角色和安全标记一一对应，每个角色对应一个安全标记。

(2) $D=\{gen\_d\}$, $T=\{gen\_t\}$ 即系统只有一个域和一个类型。$RD=\{(r,gen\_d)|r\in R\}$，即：$Rdom(r)=\{gen\_d\}$，$r\in R$，就是所有角色的授权域都只有 *gen_d* 这个域。同样所有客体的类型都是 *gen_t*，即：$OT=\{(o, gen\_t)|\ o\in O\}$。

(3) $DTM=\{(d, t, p)|d\in D, t\in T, p\in AP\}$，即 *gen_d* 域的主体对 *gen_t* 类型的客体具有所有域访问权限。$DDI=\{(d1, d2, p)|\ d1\in D, d2\in D, p\in IP\}$，即 *gen_d* 域中的主体之间拥有所有的域交互权限。

(4) $RCAP=\Phi$，$Rolecap(r:R)=\Phi$，所有角色的角色权限为空。

5.2.2 实施 DTE 安全策略

同样也可以实现 DTE 安全策略的功能，只需按如下方式配置规则：

(1) $R=\{gen\_r\}$，系统只有一个角色。$UA=\{(u, gen\_r)|u\in U\}$，该角色被指派给所有的用户。

(2) $RD=\{(gen\_r,d)|d\in D\}$，即系统中的所有域都授权给系统中唯一的角色--*gen_r* 角色。

(3) $C=\{Only\_C\}, I=\{Only\_I\}, SL=\{(Only\_C, Only\_I)\}$, 即系统只有一个安全标记。$RL=\{(r, (Only\_C, Only\_I))|r\in R\}$，每个角色都只有该安全标记。

(4) $RCAP=\Phi$, $Rolecap(r:R)=\Phi$, *gen_r* 角色的角色权限为空。

5.2.3 实施 RBAC 安全策略

用 MPVSM 模型可以实现 RBAC 模型的功能，只需按如下方式配置规则：

(1) $D=\{gen\_d\}, T=\{gen\_t\}$，系统只有一个域和一个类型。$RD=\{(r,gen\_d)|r\in R\}$，即 *gen_d* 域授予给所有的角色。所有客体都是 *gen_t* 类型，即：$OT=\{(o, gen\_t)|o\in O\}$。

(2) $DTM(gen\_d, gen\_t)=\Phi$，即 *gen_d* 域的主体对 *gen_t* 类型的客体不具有任何域访问权限。$DDI(gen\_d, gen\_d)=\Phi$，即 *gen_d* 域中的主体之间不具有任何域交互权限。

(3) $C=\{Only\_C\}$，$I=\{Only\_I\}$，$SL=\{(Only\_C, Only\_I)\}$，即系统只有一个机密级别和一个完整级别。$RL=\{(r,(Only\_C, Only\_I))|r\in R\}$，每个角色的安全级别都是$(Only_C, Only_I)$。

5.2.4 在同一系统不同用户群中实施不同的安全策略

如前所述，多安全策略视图在很多系统中是需要的。假设系统中的所有用户分为三个部分：Grpa、Grpb、Grpc。每个部分的用户所见到的安全策略是不同的，如 Grpa 的用户认为系统的安全策略是多级安全策略，而 Grpb 的用户则认为系统是采用 RBAC 安全策略，而 Grpc 的用户则认为系统的安全策略是 DTE。下面给出了用 MPVSM 模型实现这种多策略视图的方法：

(1) U=Grpa∪Grpb∪Grpc，并且 Grpa、Grpb、Grpc 两两互不相交。

(2) R= mls_rs∪rbac_rs∪{dte_r}. mts_rs 为 MLS 模型对应的角色集，rbac_rs 为 RBAC 策略对应的角色集。dte_r 为 DTE 策略对应的角色。

(3) D= {mls_d}∪{rbac_d}∪dte_ds。

(4) $(u,r)\in UA\wedge(u,r')\notin UA$，其中 u∈Grpa，r∈mls_rs，r'∉mls_rs。即 Grpa 中用户仅仅指派角色集 mls_rs 中的角色。$(u,r)\in UA\wedge(u,r')\notin UA$，其中 u∈Grpb，r∈rbac_rs，r'∉rbac_rs。即 Grpb 中用户仅仅指派角色集 rbac_rs 中的角色。同样，$(u,dte\_r)\in UA\wedge(u,r)\notin UA$，其中 u∈Grpc，r≠dte_r。Grpc 中所有用户都仅指派角色 dte_r。

(5) |mls_rs|=|*SL*|，即 mls_rs 中的角色数量等于系统中的安全标记数。mls_rs 中的角色和 *SL* 中的安全标记一一对应，每个角色对应一个安全标记。rbac_rs 集合中的角色对应的安全标记为所有标记中最低级别的。即：∀r∈rbac_rs，*RL*(r)=(cs, is)，其中，∀c∈*C*，cs≤c；∀i∈*I*，is≤i。角色 dte_r 对应的安全标记为所有标记中最高级别，即：*RL*(dte_r)=(cs, is)，其中，∀c∈*C*，cs≥c；∀i∈*I*，is≥i。

(6) $(r, mls\_d)\in RD\wedge(r, d)\notin RD$，其中 r∈mls_rs，d≠mls_d。即 mls_rs 中的角色的授权域仅仅为 mls_d。$(dte\_r, d)\in RD\wedge(r', d)\notin RD$，其中 r'≠dte_r，d∈dte_ds。即 dte_r 角色的授权域为整个 dte_ds 集合。同样，$(r, rbac\_d)\in RD\wedge(r, d)\notin RD$，其中 r∈rbac_rs，d≠rbac_d。即 rbac_rs 中的角色的授权域为 rbac_d.

(7) $(mls\_d,t,p)\in DTM$，t∈*T*，p∈*P*，即 mls_d 域中的主体对任何类型的客体都具有所有的域访问权限。*DDI*(mls_d, d)= *IP*/{*transfer*}, d∈*D*，mls_d 域中的主体不能转换到其他任何域中去，但对其他域的主体具有除 *transfer* 之外的所有交互权限。

(8) $(rbac\_d,t,p)\notin DTM$，t∈*T*，p∈*P*，即 rbac_d 域中的主体对任何类型的客体不具有任何域访问权限。*DDI*(rbac_d,d)=Φ，d∈*D*，rbac_d 域的主体不能转换到其他任何域中去。

(9) $(r, c)\notin RCAP$, r∈mls_rsU{dte_r}，mls_rs 中的角色和角色 dte_r 的角色权限为空。

使用以上规则之后，呈现给用户群 Grpa 的安全策略是基于机密性格和完整性格的混合多级安全策略，呈现给用户群 Grpb 的是 RBAC 安全模型，而呈现给用户群 Grpc 的则是 DTE 安全模型。

5.3 Nutos 系统层的安全策略配置

这章给出了 Nutos 系统层的安全策略配置例子。

系统中的用户 $U=User\_U \cup Pri\_U \cup \{sys\_usr\}$。系统中的角色包含三个子集：$R = User\_R \cup Pri\_R \cup Sys\_R$. 系统中的域和类型也都分别包含三个子集：$D = User\_D \cup Pri\_D \cup Sys\_D$, $T = User\_T \cup Pri\_T \cup Sys\_T$。

Pri_R 中的角色是可以运行特权程序的特权角色。Sys_R 中的角色是操作系统功能进程的运行角色。所有其它的应用层角色都在 *User_R* 中。*User_R* 中的角色只能分配给 *User_U* 或 *Pri_U* 中的用户。*Pri_R* 中的角色只能分配给 *Pri_U* 中的用户，*Sys_R* 中的角色只能分配给用户 sys_usr。

User_D 中的域只能授权给 *User_R* 或 *Pri_R* 中的角色，同时，*Pri_D* 中的域只能授权给 *Pri_R* 中的角色，*Sys_D* 中的域只能授权给 *Sys_R* 中的角色。

User_U 中的用户或 *User_R* 中的角色运行的进程的数据空间对象的类型是 *User_T* 和 *Userbuf_T* 的类型，根据该对象是进程的私有对象还是为其它进程的数据输入的缓冲区对象。同样，特权用户进程的数据对象分配的类型都在 *Pri_T* 中，操作系统功能服务进程的数据对象的类型都在 *Sys_T* 中。

Pri_D、*Sys_D*、*Pri_R*、*Sys_R*、*Pri_T* 和 *Sys_T* 六个集合是静态配置的，在系统运行过程中是不能改变的。这些配置在系统编译时确定下来，配置如下：*Pri_D* = {*admin_d*, *opt_d*}, *Sys_D*={*rs_d*, *pm_d*, *driver_d*, *ss_d*, *kernel_d*}, *Pri_R*={*sysopt_r*, *sysconfig_r*, *secopt_r*, *secconfig_r*, *netopt_r*, *netconfig_r*}, Sys_R={*sys_r*, *sec_r*, *kernel_r*}, *Pri_T*={*admin_t*, *adminbuf_t*, *opt_t*, *optbuf_t*}, Sys_T={*rs_t*, *rsbuf_t*, *pm_t*, *pmbuf_t*, *driver_t*, *driverbuf_t*, *ss_t*, *ssbuf_t*, *kernel_t*, *kernelbuf_t*}.

系统机密性级别集合 $C = User\_C \cup \{sys\_con\}$, *User_C* 是为应用进程角色定义的机密性标记集合。*User_C* 中的每个安全标记都大于 *sys_con*. *User_C* 集合中至少有一个元素 $usr\_con \in User\_C$。同样，系统中的完整性标记集合 $I=User\_I \cup \{pri\_int, sys\_int, sec\_int, kernel\_int\}$, *User_I* 是为应用进程角色定义的完整性标记集合。*kernel_int* 是完整级别最高的完整性标记，*sys_int* 的完整级别低于 *kernel_int*，*pri_int* 是三个标记中完整性级别最低的。*User_I* 中的每个完整性标记的完整级别都低于 *pri_int*。*User_I* 集合中至少有一个元素，$usr\_int \in User\_I$。

角色 *kernel_r* 的完整性标记是 *kernel_int*，角色 *sys_r* 和 *sec_r* 的完整性标记都是 *sys_int*. *Pri_R* 中的角色的完整性标记都为 *pri_int*。*Sys_R* 和 *Pri_R* 中的角色的机密性标记为 *sys_con*。

微内核的所有客体对象的安全标记为(*sys_con*, *kernel_int*)，策略服务器的客体对象的安全标记为(*sys_con*, *ss_int*)，资源管理、进程管理和驱动进程的客体对象的安全标记

为(*sys_con*, *sys_int*)，特权进程所有的客体对象的安全标记为(*sys_con*, *pri_int*)。

Sys_D 中任意两个域之间都不能相互转换，即：$transfer \notin DDI(d_1, d_2), d_1, d_2 \in Sys\_D$。进程管理在域 *pm_d* 中运行，资源管理在域 *rs_d* 中运行，驱动进程在 *driver_d* 中运行，策略服务器在 *ss_d* 运行，微内核在 *kernel_d* 中运行。

进程管理的私有数据地址空间对象的类型为 *pm_t*，输入数据缓冲对象的类型为 *pmbuf_t*。资源管理的数据对象空间分别是 *rs_t* 和 *rsbuf_t*。策略服务器的数据对象空间分别是 *ss_t* 和 *ssbuf_t*，微内核的分别为 *kernel_t* 和 *kernelbuf_t*。*Sys_D* 中的域对 *Sys_T* 中的类型的访问模式见表 5-3。

$write \in DTM(d, t), d \in Sys\_D, t \in Userbuf\_T$. *Sys_D* 中的所有域对 *Userbuf_T* 中的所有类型的对象都有 *write* 权限。

对于每个角色 $r \in User\_R \cup Pri\_R$ 对 *rsbuf_t* 和 *pmbuf_t* 类型的客体都有 *write* 权限。即：$Rolecap(r: User\_R \cup Pri\_R) = \{(write, o) \mid OT(o) = rsbuf\_t \vee OT(o) = pmbuf\_t\}$.

表 5-3 DTM 矩阵

	rs_t	*rsbuf_t*	*pm_t*	*pmbuf_t*	*driver_t*
kernel_d		*w*		*w*	
ss_d		*w*		*w*	
pm_d			*r,w*	*r*	
rs_d	*r,w*	*r*			
driver_d		*w*			*r,w*

表 5-3 (续) DTM 矩阵

	driverbuf_t	*ss_t*	*ssbuf_t*	*kernel_t*	*kernelbuf_t*
kernel_d	*w*		*w*	*r,w*	*r*
ss_d		*r,w*	*r*		
pm_d			*w*		*w*
rs_d	*w*		*w*		*w*
driver_d	*r*				*w*

第六章 结论及进一步工作

本文给出了一个混合多策略模型，该模型有机综合了 MLS、DTE 和 RBAC 等安全模型的功能和属性，消除了 MLS 模型的缺陷，使其可以方便地表达各种信道控制策略，同时细化了权限的分配，提高了策略表达能力，保证了可信主体在不违背最小特权原则下解决安全降级的问题。并综合了多个安全模型的特点，为实现多安全策略视图提供了一个灵活的框架。文中给出了 MPVSM 模型的形式化描述，以及模型在我们的原型可信操作系统 Nutos 中的实现以及具体的策略配置实例。

同时，本文还分析了基于微内核操作系统的分层完整性保障机制，并应用到我们实现的原型操作系统 Nutos 中。基于微内核的 Nutos 操作系统将内核划分为微内核层和功能服务模块层，为操作系统建立了有纵深的防御体系，在不同强度的攻击下尽可能多地保证系统各种安全底线不被破坏。由于 i386 的硬件保护机制保证了安全微内核的不被篡改性，有效地隔离了各种内核安全威胁，抵御内核模块攻击等针对系统内核的攻击，保证了操作系统的整体安全性。Nutos 系统在有效地保障自身各功能模块，包括安全决策模块和决策实施模块的完整性的基础上实现多安全策略，保障操作系统应用用户的各种安全性需求，解决了存在于主流操作系统中的各种安全缺陷和安全隐患。

多策略安全模型的设计和安全操作系统的初步构建，为我们下一步的形式化工作打下了基础。下一步的研究工作将主要包括：对操作系统微内核的相关操作和安全状态进行形式化建模，定义安全条件；研究各服务模块的最大权能，并对其进行形式化描述；研究 MPVSM 安全模型在各种应用中的具体配置实施方法，以及在各种实施情况下的模型正确性的形式化证明。

致 谢

首先，我要非常感谢我的导师黄皓教授，感谢他的知遇之恩和悉心指导，感谢他在我迷茫和困惑的时候给我指明了方向。在和黄老师相处的这三年中，我学到的不仅仅是做研究的方法，更多的是严谨求实的做研究的态度和诚恳正直的做人的态度。再多言语也无法表达心中对导师的那份敬意和感激，以后无论从事什么领域的工作，导师的教导都将会使我受益终身。

其次我要感谢课题组中的所有同学：王志强、崔俊、于淑英、刘国斌、薛永岭、张博、王彬、王宇、徐艳湘，感谢大家的努力、支持和帮助，在共同的研究和讨论中使我得到了启发、开拓了思路，特别是原型操作系统的实现是大家共同的工作成果。

感谢南京大学及计算机科学与技术系给我的宝贵的学习深造的机会和良好的学习科研条件。

感谢我的父母和家人在生活和学习上对我无微不至的照顾、理解和支持。特别感谢我的爱人，她一直给予我支持和鼓励，给我前进的动力。

感谢国家自然科学基金项目“操作系统的安全结构及其在单向网关上的应用（60473099）”和江苏省自然科学基金项目“计算机安全产品自身安全保障体系的研究（BK2002073）”的资助。

最后感谢各位专家评委的辛勤工作。

参考文献

[Ames 83] S. Ames, et al., Security Kernel Design and Implementation: An Introduction. *IEEE Computer*, v16 n7: p14-23, July 1983.

[Amoroso 91] E.Amoroso, T.Nguyen, J.Weiss, etc. Towards an Approach to Measuring Software Trust. In *proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pp:198-218, May 1991.

[Anderson 72] J. Anderson. Computer Security Technology Planning Study. U.S. Air Force Electronic Systems Division, TR-73-51, Bedford, MA, Oct. 1972.

[Badger 95] L. Badger, D. F. Sterne, D. L. Sherman, et al. A Domain and Type Enforcement UNIX Prototype. In *proceedings of the Fifth USENIX UNIX Security Symposium*, June 1995.

[Bell 75] D.Bell, L.LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report MTR-2547, Vol. I, MTR-2997 Rev.1, MITRE Corporation, Bedford, MA, Mar. 1975.

[Berger 90] J. L. Berger, J. Picciotto, J. P. L. Woodward, P. T. Cummings. Compartmented mode workstation: Prototype highlights. *IEEE Transactions on Software Engineering*, Special Section on Security and Privacy, 16(6):608–618, 1990..

[Biba 77] K.Biba. Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, MITRE Corporation, Bedford, MA, Apr. 1977.

[Denning 77] D.E. Denning, P.J. Denning. Certification of Programs for Secure Information Flow. *Communication of the ACM*, 20(7): 504-513, 1977.

[Denning 76] D. Denning. A Lattice Model of Secure Information Flow. *Communication of the ACM*, 19(5): 236-250, 1976.

[Efstathopoulos 05] P. Efstathopoulos, M. Krohn, S. VanDeBogart, et al. Labels and Event Processes in the Asbestos Operating System. In *proceedings of the 20th Symposium on Operating Systems Principles*, Oct. 2005.

[Eswaran 75] K.Eswaran, D.Chamberlin. Functional Specifications of Subsystem for Database Integrity. In *proceedings of the International Conference on Very Large Data Bases*, Sep.1975.

[Feiertag 79] R.J. Feiertag, P.G. Neumann. The Foundations of a Provably Secure Operating System (PSOS). In *proceedings of the National Computer Conference*, 48: 329-334, 1979.

[Feustel 98] E.A. Feustel, T. Mayfield. The DGSA: Unmet Information Security Challenges for Operating System Designers. *ACM Operating Systems Review*, 32(1): 3-22, Jan 1998.

[Fine 93] T. Fine, S.E. Minear, M.J. Nash. Assuring Distributed Trusted Mach. In *proceeding of the 1993 IEEE Computer Society Symposium on Security and Privacy*, 206-218.

[Ford 78] Ford Aerospace and Communications Corporation. Secure Minicomputer Operating System (KSOS) Executive Summary: Phase I: Design of the Department of Defense Kernelized Secure Operating System. WDL-781, Palo Alto, CA 94303, Mar 1978.

[Fraser 99] T. Fraser. LOMAC–low water-mark mandatory access control for Linux. In Proceedings of the USENIX Security Symposium, 1999.

[Goguen 82] J. Goguen, J. Meseguer. Secuirty Policies and Security Models. In *proceedings of the 1982 IEEE Symposium on Research in Security and Privacy*, 1982.

[Gold 84] B.D. Gold, et al. KVM/370 in Retrospect. In *proceedings of the 1984 Symposium on Security and Privacy*, 1984.

[Goldberg 73] R. P. Goldberg. Architecture of virtual machines. In *Proceedings of AFIPS National Computer Conference*, 42:309–318, June 1973.

[Karger 90] P. Karger, et al. A VMM Security Kernel for the VAX Architecture. In *proceedings of the 1990 Symposium on Reserarch in Security and Privacy*, 1990.

[Kelem 91] N.L. Kelem, R.J. Feiertag. A Separation Model for Virtual Machine Monitors. In *proceedings of IEEE Symposium Security and Privacy*, 1991.

[Halfmann 99] U. Halfmann, W.E. Kuhnhauser. Embedding Security Policies into a Distributed Computing Environment. *ACM Operating Systems Review*, 33(2): 51-64, Apr. 1999.

[Harrison 76] M. Harrison, W. Ruzzo, J. Ullman. Protection in Operating Systems. *Communications of the ACM*, 19(8):461-471, 1976.

[Herder 06-1] JN.Herder, H.Bos, B.Gras, et al. MINIX 3: A Highly Reliable, Self-Repairing Operating System. In *proceedings of ACM SIGOPS Operating Systems Review*, 2006.

[Herder 06-2] JN.Herder, H.Bos, B.Gras, et al. Construction of a Highly Dependable Operating System. In *proceedings of the Sixth European Dependable Computing Conference*, 2006.

[Lampson 74] B. Lampson. Protection. In *5th Princeton Symposium on Information Science and Systems*, 1971, Reprinted in *ACM Operating Systems Review*, 8(1):18-24, 1974.

[Lampson 76] B. Lampson, H. Sturgis. Reflections on an Operating System Design. *Comm of the ACM*, 19(5): 251-266, May 1976.

[Liedtke 93] J. Liedtke. Improving IPC by Kernel Design. In *proceedings of the ACM Symposium on Operating System Principles (SOSP)*, 1993.

[Liedtke 95] J. Liedtke, K. Elphinstone, et al. Achieved IPC Performance. In *proceedings of the 5th USENIX Security Symposium*, 1995.

[Liedtke 96] J.Liedtke. Toward real microkernel. *Comm. of ACM,* 1996

[Lipner 82] S.Lipner. Non-Discretionary Controls for Commercial Applications. In *proceedings of the 1982 Symposium on Privacy and Security*, Apr.1982.

[Lipton 77] R. Lipton, L. Snyder. A Linear Time Algorithm for Deciding Subject Security. *Journal of ACM*, 24(3):455-464, 1977.

[Loepere 92] K. Loepere. Mach 3 Kernel Principles. Technical report, Open Software Foundation

and Carnegie Mellon University, NORMA-MK12, July1992.

[Logic 89] Key Logic. The KeyKOS/KeySAFE System Design. SEC009-01. http://www.agorics.com/Library/KeyKos/keysafe/Keysafe.html, Mar. 1989.

[Loscocco 98] P.A. Loscocco, S.D. Smalley, P.A. Muckelbauer, et al. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In *proceedings of the 21st National Information Systems Security Conference*, Oct. 1998.

[Loscocco 01] P.Loscocco, S.Smalley. Meeting critical security objectives with security-enhanced linux. In *proceedings of Ottawa Linux Symposium 2001*, June 2001.

[McCauley 79] E.J. McCauley, P.J. Drongowski. KSOS: The Design of a Secure Operating System. *AFIPS Conf. Proc.*, *1979 National Computer Conference*, 1979.

[McLean 94] J. McLean. Security Models. *Encyclopedia of Software Engineering,* Wiley Press, 1994.

[Neumann 75] P.G. Neumann. A Provably Secure Operating System: Final Report. DAAB03-73-C-1454, Stanford Research Institute, June 1975.

[Nibaldi 79-1] G.H. Nibaldi. Proposed Technical Evaluation Criteria for Trusted Computer Systems. M79-225, The MITRE Corporation, Bedford, MA, Oct. 1979.

[Nibaldi 79-2] G.H. Nibaldi. Specification of a Trusted Computing Base. M79-228, The MITRE Corporation, Bedford, MA, 1979.

[Organick 72] E.Organick. The MULTICS System: An Examination of Its Structure. The MIT Press, Cambridge, MA, 1972.

[Osborn 00] S. Osborn, R. Sandhu, Q. Munawer. Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and System Security*, 3(2): 85-106, May 2000.

[Popek 79] G. J. Popek, M. Kampe, C. S. Kline, E. J. Walton. UCLA Data Secure Unix. *AFIPS Conf. Proc.*, 48:355-364, *1979 National Computer Conference*, 1979.

[Radhakrishnan 06] M. Radhakrishnan, J. A. Solworth. Application Support in the Operating System Kernel, In *proceedings of ACM Symposium on InformAtion, Computer and Communications Security*, 2006.

[Rushby 92] J.Rushby. Noninterference, Transitivity, and Channel-Control Security Policies. Technical Report CSL-92-02, Computer Science Lab, SRI International, Dec. 1992.

[Saltzer 75] J. H. Saltzer, M. D. Schroeder. The protection of information in computer system. In *Proceedings of the IEEE*, 63(9):1278–1308, Sep. 1975.

[Sandhu 91] R. Sandhu. The Typed Access Matrix Model. In *proceedings of IEEE Symposium on Security and Privacy*, May 1991.

[Sandhu 96] R. Sandhu, E.Coyne, H.Feinstein, C.Youman. Role-Based Access Control. *IEEE Computer*. 29(2), Feb. 1996

[Sandhu 97] R. Sandhu. Rational for the RBAC96 Family of Access Control Models. In

Proceedings of 1st ACM Workshop on Role-based Access Control, 1997.

[Schiller 73] W. L. Schiller. The Design and Specification of a Security Kernel for the PDP-11/45. MTR-2709, the MITRE Corporation, Bedford, MA, June 1973.

[Schiller 75] W. L. Schiller. Design of a Security Kernel for the PDP-11/45. MTR-2934, the MITRE Corporation, Bedford, MA, Mar. 1975.

[Secure 96] Secure Computing Corporation. DTOS FORMAL TOP-LEVEL SPECIFICATION. Technical report, 83-0902024A001 Rev A, 1996.

[Secure 97] Secure Computing Corporation. DTOS Lessons Learned Report. CDRL Sequence No.A008, Secure Computing Corporation, Roseville, Minnesota, June 1997.

[Shapiro 99] J. S. Shapiro, J. M. Smith, D. J. Farber. EROS: A Fast Capability System. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, 1999

[Shapiro 04] J. Shapiro, M. S. Doerrie, E. Northup, et al. Towards a Verified, General-Purpose Operating System Kernel. In *proceedings of 1st NICTA Workshop on Operating System Verification*, Oct. 2004.

[Smith 97] J.E. Smith, F.W. Weingarten. Research Challenges for the Next Generation Internet. In *Workshop on Research Direction for the Next Generation Internet*, May 1997.

[Spencer 99] R. Spencer, S. Smalley, M. Hibler, et al. The Flask Security Architecture: System Support for Diverse Security Policies. In *Proceedings of the 8th USENIX Security Symposium*, Aug. 1999.

[VMware 01] VMware. VMware and the National Security Agency team to build advanced secure computer systems. http://www.vmware.com/pdf/ TechTrendNotes.pdf , Jan. 2001.

[Walker 80] B.J. Walker, R.A. Kemmerer, G.J. Popek. Specification and Verification of the UCLA Unix Security Kernel. *Communicatin of the ACM*, 23(2):118-131, Feb. 1980.

[Walker 96] K. M. Walker, D. F.Sterne, M. L. Badger, et al. Confining Root Programs with Domain and Type Enforcement (DTE). In *Proceedings of the 6th USENIX UNIX Security Symposium*, July 1996.

[Watson 02] R.N.M.Watson. TrustedBSD: Adding trusted operating system features to FreeBSD. In *Proceedings of the 2002 Symposium on Operating System Design and Implementation (OSDI)*, Dec. 2002

[Weissman 69] C. Weissman. Security controls in the ADEPT-50 time-sharing system. In *Proceedings of the 1969 AFIPS Fall Joint Computer Conference*, 1969.

[Zanin 04] G. Zanin, L.V. Mancini. Towards a Formal Model for Security Policies Specification and Validation in the SELinux System. In *Preceedings of the 9th ACM Symposium on Access Control models and technologies*, June 2004.

中文参考文献

[1] 林志强，夏耐，茅兵，谢立. 缓冲区溢出研究综述. 计算机科学，31(9)，2004

[2] 谢钧. 计算机安全产品的自身安全保障技术的研究. 南京大学博士论文，2005.10
[3] 谢钧，许峰，黄皓. 基于可信级别的多级安全策略及其状态机模型. 软件学报，15(11)，2004.11
[4] 谢钧，黄皓，张佳. 多保护域进程模型及其实现. 电子学报，33(1): 38-42，2005
[5] 王志强，黄皓，夏磊. 进程内细粒度保护域模型及其实现. 计算机应用，2007.6
[6] 王志强. 同一地址空间下多保护域模型. 南京大学硕士论文，2007.5

附录 A 硕士期间发表的论文

[1] Lei Xia, Hao Huang, Shuying Yu. Towards a Multi-Model Views Security Framework. To appear in *International Conference on Security and Cryptography* (SECRYPT 2007), Barcelona, Spain, July 28, 2007.

[2] 夏磊，黄皓，王志强.一种使用 RBAC 模拟实施 BLP 的方法.*计算机应用研究*，2008.3

[3] 王志强，黄皓，夏磊. 进程内细粒度保护域模型及其实现. *计算机应用*，2007.6

附录 B 硕士期间参加的科研工作

[1] 江苏省自然科学基金项目(BK2002073)：“计算机安全产品自身安全保障体系的研究”，2004.9-2005.3

[2] 国家自然科学基金项目(60473099)：“操作系统的安全结构及其在单向网关上的应用”，2005.4-2007.5