摘 要

信息安全是受到各方关注的焦点问题，但目前信息安全工作集中在应用层及传输通信层。事实上，现有的 Linux 操作系统有很多安全缺陷，如访问控制简单、Root 权限过大、系统审计不足等。操作系统是唯一紧靠硬件的基本软件，其安全职能是其他软件安全职能的根基，缺乏这个安全的根基，构筑在其上的应用系统安全性是得不到根本保障的。因此，增强操作系统本身安全性，研究操作系统安全增强技术，是解决国内计算机用户普遍存在的信息安全问题的关键。

本文主要论述了 Linux 操作系统的安全增强技术与实现。论文中首先介绍了 Linux 内核与网络系统的基本知识。然后从工作机制与应用两个角度具体论述了能力安全机制在 Linux 操作系统上的实现。第四章对 linux 系统配置从系统安全、网络安全与控制台安全三个方面进行了深入细致的阐述，在传统 Linux 虚拟控制台基础上作了一些有意义的改进，实现 Linux 服务器的串口控制台管理及虚拟控制台的自动登录。第五章详细讨论了 BLP 安全模型，在理论上对 Linux 系统安全作了抽象提高与总结；接着在第六章通过 BLP 模型实体对 Linux 系统的映射，借助可卸载内核模块机制，基于内核系统调用函数的安全改进与增删，实现 BLP 模型存取控制机制在 Linux 系统上的应用。最后对整个课题研究与开发进行了总结，并对 Linux 系统安全作了进一步的展望。

关键字：能力；控制台；BLP 安全模型；存取控制机制；系统调用；可卸载内核模块

Abstract

Information Security is a focal issue, which gains more and more attention these years. Now Information Security concentrates on application layer or transport layer. But in reality , mainstream Linux operating systems has many flaws on security.For example ,access control is too simple. Capability of root is no limit and there is lack of system audit. Operating system is the only basic software anear the hardware.Its security is the base of other software.If there is no the security base, the security of application system is not ensured. Thus,researching the technique to enhance the operating system security is critical to resolve the problem of information security.

In this paper we discuss how to build a security Linux operating system. At first, we introduce some basic knowledge of the Linux kernel and the network system.Then we describe the principles of capability security mechanism,and explain how to build a capability based on Linux system. In chapter 4 we discuss in detail the configure of Linux system at three aspects:system security, network security and console security.Then we give a systemic description of the BLP Security Model.At last ,based on the BLP model, we introduce the Linux loadable kernel module, and how to use LKMs to implement the enhanced security Linux operating system.

Key words: Capability; Console; BLP Security Model; Access Control Mechanism; System Call; Loadable Kernel Module

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：刘瑜 日期： 2004 年2 月 日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：刘瑜 导师签名：

日期： 2004 年 2月27日

第一章 绪 论

1.1 引言

随着科技和经济的高速发展，信息技术带给人类的影响日益扩大，尤其是网络的发展使计算机的应用日益普及。现代信息主要的表现形式是数字化方式。随着计算机及网络技术的高速发展、全球信息化时代的到来，在给人们工作、生活的方方面面带来方便和快捷的同时，也带来信息安全方面的威胁；非法的计算机系统入侵，窃取、篡改、毁坏重要的信息数据，给社会造成难以估量的损失，同时也为信息技术在更广泛领域的应用设置了障碍。Internet、电子商务、电子政务等领域的应用，均对系统的安全性提出了较高的要求，因此，信息本身的保密性、完整性、防抵赖性以及信息来源和去向的可靠性，在数字化时代尤为重要。

世界范围的信息革命激发了人类历史上最活跃的生产力，但同时也使得信息的安全问题日渐突出而且情况也越来越复杂。从大的方面来说，信息安全问题已威胁到国家的政治、经济、军事、文化、意识形态等领域。从小的方面来说，信息安全问题也是人们能否保护自己个人隐私的关键。

信息安全是为信息系统建立和采取的技术和管理的安全保护，防止信息资料不因偶然和恶意的原因而遭到非授权泄漏、更改、破坏，即确保信息的完整性、保密性、可用性。操作系统是连接计算机硬件与上层软件及用户的桥梁，操作系统安全性是计算机安全的重要基础，要妥善解决日益广泛的计算机安全问题，必须有坚固的安全操作系统作后盾。而操作系统的安全性涉及到身份认证、强制访问控制、敏感信息的多级安全保密、安全审计、基于角色的授权控制、抗否认性认证、以及重要信息的通信加密与解密等内容，是信息安全的核心。

事实上，国内的计算机用户，包括企业、政府机关、科研机构和个人在内，并不都具备有足够的专业化信息安全防护知识，即使是企业的系统安全管理员，也往往不清楚什么情况下系统已经被黑客入侵，甚至不知道应该如何正确配置软、硬件系统，也无从判定软件所存在的各种漏洞，只有当信息系统发生崩溃时，他们方才知道所有信息早已“灰飞烟灭”。因此，增强信息系统本身的安全性，研究信息系统安全增强技术，特别是操作系统的安全增强技术的研究，是解决国内计算机用户普遍存在的信息安

全问题的关键。

我国的政府、国防、金融等各行各业对操作系统的安全都有各自的要求，都迫切需要找到一个即满足功能、性能要求，又具备足够的安全可信度的操作系统。Linux 的发展及其应用在国际上的广泛兴起，在我国也产生了广泛的影响，只要其安全问题得到妥善解决，将会得到我国各行各业的普遍接受。Linux 操作系统在自由软件基金会（Free Software Foundation）推动下，使得任何人都可以免费获得其源代码，对它进行分析、修改、增添新功能。从而使之成为功能与 Unix 兼容的多用户操作系统。因为它符合 IEEE POSIX.1 标准，移植性好，目前已被广泛使用。但是，它的安全问题一直得不到第三方的验证和评估。这个问题在个人用户使用时，并不明显，但是在政府部门、金融部门准备大规模应用这种操作系统的时候，却成为最大的阻碍。

1.2 国内外操作系统安全研究现状

国际上对计算机系统安全性的研究已有相当的历史。自 20 世纪 70 年代以来，起初出于军事目的，美国政府由国防部牵头，投入了大量的人力、物力，从操作系统底层开始，对计算机系统的安全问题开展了广泛、深入的研究，取得了一系列有影响的重要成果。在系统安全模型方面至今仍被广泛引用的 Bell-Lapadula 模型就是 70 年代的这些研究项目的成果之一。在长期的研究中，美国提出了不少操作系统安全结构模型，如可证明的安全操作系统、军事安全操作系统、VAX 体系的 VMM 安全内核等，对操作系统安全性的研究起了重要的推动作用。

Multics 是安全操作系统的最早尝试，在 1965 年由 AT&T 和 MIT（麻省理工学院）联合开发，它试图在各方面都做得很强大，包括安全性方面。从商业角度看，Multics 并不成功，但是单从安全性角度来看，Multics 迈出了安全操作系统设计的第一步，为后来的安全操作系统研制积累了大量经验。Mitre 公司的 Bell 和 La Padula 合作的安全模型 BLP 模型在 Multics 中得到了成功应用。

1986 年，IBM 以 SCO Xenix 为基础开发了目标为 B2 到 A1 级的 Secure Xenix，它初步实现以 IBM PC/AT 为运行基础。

1987 年，美国 Trusted Information Systems 公司以 Mach 操作系统为基础开发了 B3 级的 Tmach（Trusted Mach）操作系统。除了进行用户标识和鉴别及命名客体的存取控制外，它将 BLP 模型加以改进，运用到对 MACH

核心的端口、存储对象等的管理当中。

1989 年，加拿大多伦多大学开发了与 UNIX 兼容的 TUNIX 操作系统。它用 Turing Plus 语言（而不是 C）重新实现了 Unix 内核，这种语言是强类型高级语言，其大部分语句都具有用于正确性证明的形式语义。TUNIX 的模块性相当好。它在实现中也改进了 BLP 模型。

到 1990 年年底，大约 20 来个系统获得了 TCSEC（美国国防部《可信计算机系统评价准则》）的某一级别的安全级（D 级除外），但是大多数系统处在较低的级别。

1997 年，美国安全计算公司（SCC）和国家安全局（NSA）完成了 DTOS（Distributed Trusted Operating System）安全操作系统。与传统的基于 TCSEC 标准的开发方法不同，DTOS 采用了基于安全威胁的开发方法。

2001 年，美国国家安全局（NSA）等单位以 Flask 安全体系结构为指导，在 Linux 基础上开发了 SELinux。SELinux 定义了一个类型实施（TE）策略，基于角色的访问控制（RBAC）策略和多级安全（MLS）策略组合的安全策略。

我国在操作系统安全方面，也开展了一些工作，影响面较广的最新成果之一是在一定程度上解决了国产操作系统 COSIX 的安全实现问题。该系统以美国的 TCSEC 为蓝本，目标是 B1 级。此外。还有几家科研院所和公司实现了基于 Linux 的安全操作系统，目前最高已经达到了 B2 级。

1999 年 10 月 19 日，中国国家技术监督局发布了国家标准 GB17859－1999《计算机信息系统安全保护等级划分准则》，为计算机信息系统安全保护能力划分了等级。该标准已于 2001 年起强制执行。

1.3 课题来源与本论文工作

由于 Linux 提倡的开放源码政策及自身安全方面所具有的一些优良特性，使得基于 Linux 的安全操作系统研究发展迅速，安全操作系统的应用已非常广泛。国内近年来在这方面也取得了大量成果，但较之国外仍存在差距，在研究成果的产品化上亦有诸多缺陷。本课题源于信息产业部电子发展基金项目——新型分布式防火墙系统研究，并最终形成产品。本课题的开展，是在跟踪国际上先进网络系统安全技术的基础上，开发实现一套分布式防火墙系统原型。它将可应用于各类公司企业内部网、ISP/ICP、政府机关网络，保护网络安全。而本文的工作是该课题下的一个子项目，拟对 Linux 系统进行分析裁减，加强安全性配置与开发，扩展控制台范围（如

串口控制台，安全网口控制台和基于网页的控制台等），由此达到为系统防火墙应用提供一个坚实可靠支撑环境的目的。

本论文以如何有效建立 linux 系统安全环境为出发点，对涉及的一系列操作系统安全关键技术点进行了较深入地研究与实现。

第一章概述目前信息安全状况及国内外操作系统安全研究现状，并简要介绍了本课题的来源。然后在第二章对 Linux 系统从内核体系结构与网络接口两个方面进行了深入剖析，在此基础上，从第三章开始，围绕 Linux 最小特权管理实现、系统安全配置方法、串口控制台扩展和 BLP 模型存取控制机制的实现四个重要方面，层层深入的系统论述了 Linux 操作系统安全增强的技术细节，并在结束语中对本课题的研究作了进一步的展望。

第二章 Linux 系统分析

2.1 Linux 内核体系结构

Linux 内核的主要功能是与计算机硬件进行交互，实现对硬件部件的编程控制和接口操作，调度对硬件资源的访问，并为计算机上的用户程序提供一个高级的执行环境和对硬件的虚拟接口。内核主要由 5 个模块构成，它们分别是：进程调度模块、内存管理模块、文件系统模块、进程间通信模块和网络接口模块[8]。

进程调度模块用来负责控制进程对 CPU 资源的使用。所采取的调度策略是各进程能够公平合理地访问 CPU，同时保证内核能及时地执行硬件操作。内存管理模块用于确保所有进程能够安全地共享机器主内存区，同时，内存管理模块还支持虚拟内存管理方式，使得 Linux 支持进程使用比实际内存空间更多大的内存容量。并可以利用文件系统把暂时不用的内存数据块会被交换到外部存储设备上去，当需要时再交换回来。文件系统模块用于支持对外部设备的驱动和存储。虚拟文件系统模块通过向所有的外部存储设备提供一个通用的文件接口，隐藏了各种硬件设备的不同细节。从而提供并支持与其它操作系统兼容的多种文件系统格式。进程间通信模块子系统用于支持多种进程间的信息交换方式。网络接口模块提供对多种网络通信标准的访问并支持许多网络硬件。

这几个模块之间的依赖关系见图 2.1 所示，其中的连线代表它们之间的依赖关系。

由图可以看出，所有的模块都与进程调度模块存在依赖关系。因为它们都需要依靠进程调度程序来挂起（暂停）或重新运行它们的进程。通常，一个模块会在等待硬件操作期间被挂起，而在操作完成后才可继续运行。例如，当一个进程试图将一数据块写到软盘上去时，软盘驱动程序就可能在启动软盘旋转期间将该进程置为挂起等待状态，而在软盘进入到正常转速后再使得该进程能继续运行。另外 3 个模块也是由于类似的原因而与进程调度模块存在依赖关系。

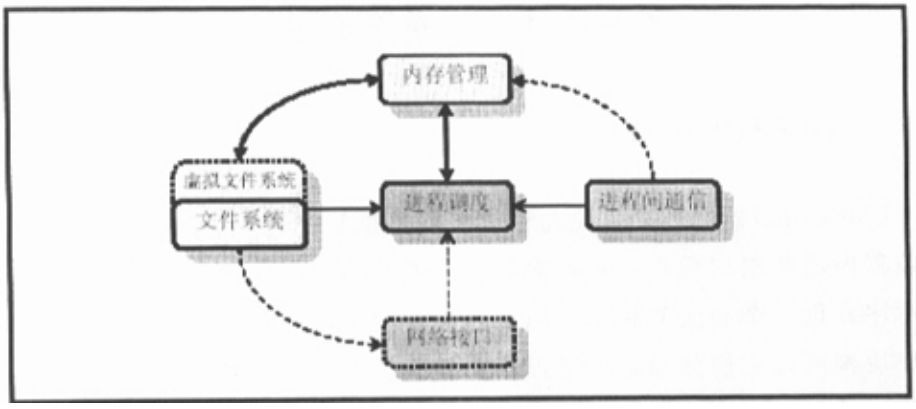


图2.1 Linux内核系统模块结构及相互依赖关系

其它几个依赖关系有些不太明显，但同样也很重要。进程调度子系统需要使用内存管理器来调整一特定进程所使用的物理内存空间。进程间通信子系统则需要依靠内存管理器来支持共享内存通信机制。这种通信机制允许两个进程访问内存的同一个区域以进行进程间信息的交换。虚拟文件系统也会使用网络接口来支持网络文件系统（NFS），同样也能使用内存管理子系统来提供内存虚拟盘（ramdisk）设备。而内存管理子系统也会使用文件系统来支持内存数据块的交换操作。

若从单内核模式结构模型出发，还可以根据 Linux 内核源代码的结构将内核主要模块绘制成图 2.2 所示的框图结构。

其中内核级中的几个方框，除了硬件控制方框以外，其它粗线方框分别对应内核源代码的目录组织结构。

除了这些图中已经给出的依赖关系以外，所有这些模块还会依赖于内核中的通用资源。这些资源包括内核所有子系统都会调用的内存分配和收回函数、打印警告或出错信息函数以及一些系统调试函数。

2.1.1 Linux 内核进程控制

程序是一个可执行的文件，而进程（process）是一个执行中的程序实例。在 Linux 操作系统上同时可以执行多个进程。每个进程都由一个 task_struct 数据结构来描述，这个数据结构包含了进程的所有信息，是系统控制进程的唯一有效手段。Linux 将所有 task_struct 结构的指针存放在一个 task 数组中，task 数组是操作系统内核空间中专门开辟的一块区域。task 数组的大小 NR_TASKS 缺省值为 512，是系统可同时存在的最大进程

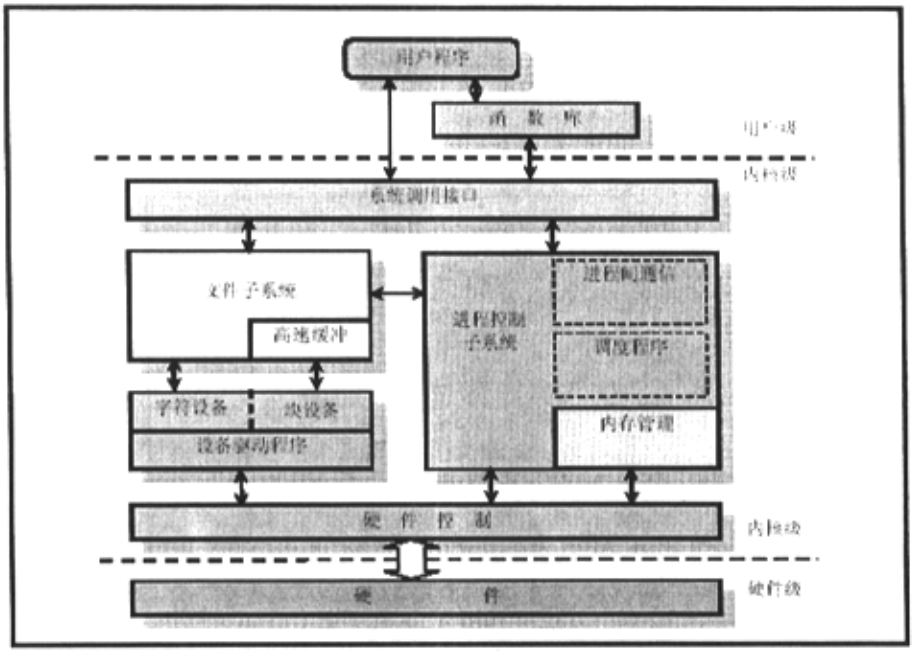


图 2.2 内核结构框图

数量。系统除了第一个进程是“手工”建立以外，其余的都是进程使用系统调用 fork 创建的新进程。内核程序使用进程标识号（process ID，pid）来标识每个进程。进程由可执行的指令代码、数据和堆栈区组成。进程中的代码和数据部分分别对应一个执行文件中的代码段、数据段。每个进程只能执行自己的代码和访问自己的数据及堆栈区。进程之间相互之间的通信需要通过系统调用来进行。对于只有一个 CPU 的系统，在某一时刻只能有一个进程正在运行。内核通过调度程序分时调度各个进程运行。

Linux 系统中，一个进程可以在内核态（kernel mode）或用户态（user mode）下执行，因此，linux 内核栈和用户栈是分开的。用户栈用于进程在用户态下临时保存调用函数的参数、局部变量等数据。内核栈则含有内核程序执行函数调用时的信息。

内核程序是通过进程表对进程进行管理的，每个进程在进程表中占有一项。在 linux 系统中，进程表项是一个 task_struct 结构。

当一个进程在执行时，CPU 的所有寄存器中的值、进程的状态以及堆栈中的内容被称为该进程的上下文。当内核需要切换（switch）至另一个进程时，它就需要保存当前进程的所有状态，也即保存当前进程的上下文，以便在再次执行该进程时，能够恢复到切换时的状态执行下去。在发生中断时，内核就处在被中断进程的上下文中，在内核态下执行中断服务例程。

但同时会保留所有需要用到的资源，以便中断服务结束时能恢复被中断进程的执行。

一个进程在其生存期内，可处于一组不同的状态下，称为进程状态。见图 2.3 所示。

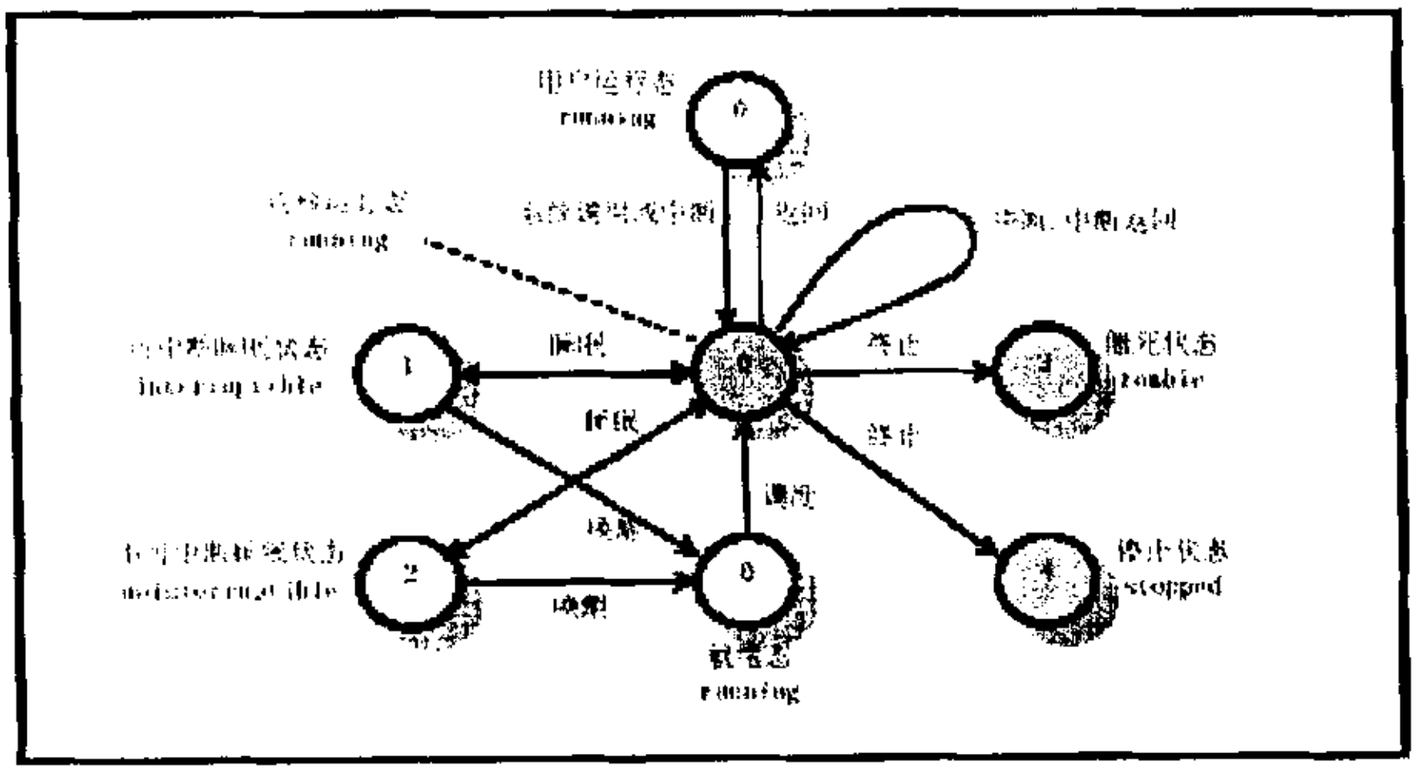


图 2.3 进程状态及转换关系

当进程正在被 CPU 执行时，称为处于执行状态（running）。当进程正在等待系统中的资源而处于等待状态时，则称其处于睡眠等待状态。在 linux 系统中，还分为可中断的和不可中断的等待状态。当系统资源已经可用时，进程就被唤醒而进入准备运行状态，该状态称为就绪态。当进程已停止运行，但其父进程还没有询问其状态时，则称该进程处于僵死状态。当进程被终止时，称其处于停止状态。

只有当进程从“内核运行态”转移到“睡眠状态”时，内核才会进行进程切换操作。为了避免进程切换时造成内核数据错误，内核在执行临界区代码时会禁止一切中断。

2.1.2 Linux 文件系统

Linux 系统的一个重要特征就是支持多种不同的文件系统。这样，Linux 系统就十分的灵活，并且可以十分容易地和其他操作系统共存。目前，Linux 系统支持大约十几类文件系统：EXT、EXT2、EXT3、XIA、MINIX、MSDOS、VFAT、PROC、SMB、ISO9660 和 HPFS 等。

在 Linux 系统中，每一个单独的文件系统都是代表整个系统的树状结构的一部分。当挂接一个新的文件系统时，Linux 把它添加到这个树状的文件系统中。所有系统中的文件系统，不管是什么类型，都挂接到一个目录下，并隐藏掉目录中原有的内容。这个目录叫做挂接目录或者挂接点。

当文件系统卸载掉时，目录中的原有内容将再一次的显示出来。

初始化时磁盘将被划分成几个逻辑分区。每一个逻辑分区可以使用一种文件系统，例如 EXT2 文件系统。文件系统把存储在物理驱动器中的文件组织成一个树状的目录结构。可以存储文件的设备称为块设备。Linux 文件系统把这些块设备当作简单的线形块的集合，而不管物理磁盘的结构如何，将读写某一个设备块的请求转换成特定的磁道、扇区和柱面是通过设备的驱动程序实现的。因此，不同的设备控制器控制的不同设备中的不同文件系统在 Linux 中都可以同样地使用。文件系统甚至可以不在当地的系统中，也就是说，文件系统可以通过网络远程连接到本地磁盘上。如下面的例子，这是一个 Linux 在 SCSI 磁盘上的根文件系统：

```
A    E    boot    etc    lib    opt    tmp    usr
C    F    cdrom   fd     proc   root   var    sbin
D    bin  dev     home   mnt    lost + found
```

在这里，用户和使用文件的程序都不必知道/C 实际上是挂接的 VFAT 文件系统，而 VFAT 文件系统本身却存储在系统中的一个 IDE 硬盘上。同样，/E 是系统中第二个 IDE 控制器控制的主硬盘系统。也可以使用调制解调器将远程的文件系统挂接到/mnt/remote 目录下。

一个文件系统中不仅包括含有数据的文件，而且还存储着文件系统的结构。文件系统中的信息必须是安全和保密的。

Linux 系统中的第一个文件系统是 Minux，但它的文件名只能有 14 个字符，最大的文件长度是 64M 字节。所以，1992 年 4 月引进了第一个专门为 Linux 设计的文件系统 ext(extended file system)。但 ext 的功能还是有限。最后在 1993 年又推出了一个新的文件系统—ext2。当 Linux 引进 ext 文件系统时有了一个重大的改进：真正的文件系统从操作系统和系统服务中分离出来，在它们之间使用了一个接口层—虚拟文件系统 VFS（Virtual File system），VFS 允许 Linux 支持多种不同的文件系统，每个文件系统都要提供给 VFS 一个相同的接口。这样所有的文件系统对系统内核和系统中的程序来说看起来都是相同的。Linux 系统中的 VFS 层使得你可以同时在系统中透明地挂接很多不同的文件系统。

VFS 能高速度及高效率地存取系统中的文件，同时它还得确保文件和数据的正确性。这两个目标有时可能相互矛盾。VFS 在每个文件系统挂接和使用时把文件系统的有关信息暂时保存在内存中，所以当内存中的信息改变时，例如创建、写入和删除目录和文件时，系统要保证正确地升级文件系统中相关的内容。系统缓存中最重要的就是缓冲区缓存，它被集成到

每个单独的文件系统存取它们的块设备所使用的方法中。每当系统存取数据块时，数据块都被放入缓冲区缓存中，并且依照它们的状态保存到各种各样的队列中。缓冲区缓存中不仅保存了数据缓冲区，而且还可以帮助管理和块设备驱动程序之间的异步接口。

2.1.2.1 ext2 文件系统概述

ext2 的物理结构图参见图 2.4

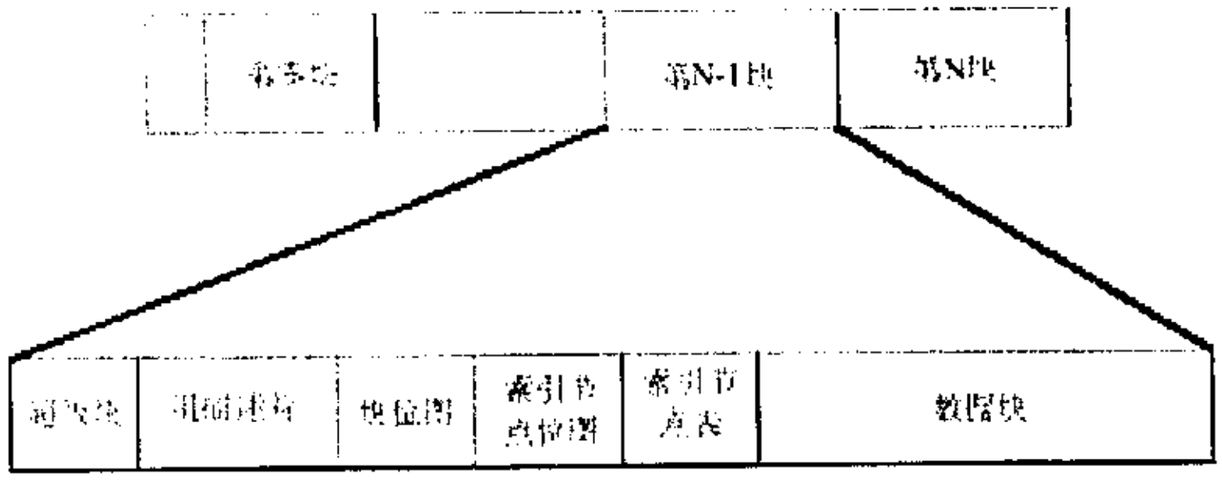


图2.4 ext2文件物理结构示意图

ext2 文件系统是 Linux 系统中最为成功的文件系统，各种 Linux 的系统发布都将 ext2 文件系统作为操作系统的基础。

ext2 文件系统中的数据是以数据块的方式存储在文件中的。这些数据块具有同样的大小，并且其大小可以在 ext2 创建时设定。每一个文件的长度都要补足到块的整数倍。例如，如果一个块的大小是 1024 字节，那么一个 1025 字节大小的文件则占用两个数据块。所以，平均来说一个文件将浪费半个数据块的空间。但这样可以减轻系统中 CPU 的负担。文件系统中不是所有的数据块都存储数据，一些数据块用来存储一些描述文件系统结构的信息。ext2 通过使用索引节点（i_node）数据结构来描述系统中的每一个文件。索引节点描述了文件中的数据占用了哪一个数据块以及文件的存取权限、文件的修改时间和文件类型等信息。ext2 文件系统中的每一个文件都只有一个索引节点，而每一个索引节点都有一个唯一的标识符。文件系统中的所有的索引节点都保存在索引节点表中。ext2 中的目录只是一些简单的特殊文件，这些文件中包含指向目录入口的索引节点的指针。

对于一个文件系统来说，某一个块设备只是一系列可以读写的数据块。文件系统无须关心数据块在设备中的具体位置，这是设备驱动程序的工作。每当文件系统需要从块设备中读取数据时，它就要求设备驱动程序读取整数数目的数据块。ext2 文件系统将它所占用的设备的逻辑分区分成了数据块组。每一个数据块组都包含一些有关整个文件系统的信息以及真

正的文件和目录的数据块。

2.1.2.2 VFS（虚拟文件系统）

图 2.5 显示了 Linux 系统内核中的 VFS 和实际文件系统之间的关系。VFS 必须管理同时挂接在系统上的不同的文件系统。它通过使用描述整个 VFS 的数据结构和描述实际挂接的文件系统的数据结构来管理这些不同的文件系统。

VFS 和 ext2 文件系统一样使用超级块和索引节点来描述系统中的文件。和 ext2 中的索引节点一样，VFS 的索引节点用来描述系统中的文件和目录。

当一个文件系统初始化时，它将在 VFS 中登记。这些过程在系统启动操作系统初始化时完成。实际的文件系统或者内建到系统内核中，或者作为可装入模块在需要时装入。当一个基于块设备的文件系统挂接时，当然也包括根文件系统， VFS 首先要读取它的超级块。每一个文件系统的超级块读取程序都必须先清楚文件系统的结构，然后把有关的信息添加到 VFS 的超级块数据结构中。VFS 中保存了系统中挂接的文件系统的链表以及这些文件系统对应的 VFS 超级块。每一个 VFS 超级块都包含一些信息和指向一些执行特别功能的子程序的指针。例如，代表挂接的 ext2 文件系统的 VFS 超级块包含了一个指向 ext2 索引节点读取程序的指针。这个 ext2 索引节点读取程序，和其他文件系统索引节点的读取程序一样，把信息添加到 VFS 索引节点中的字段中。每一个 VFS 超级块都包含了一个指向文件系统中第一个 VFS 索引节点的指针。

对于根文件系统来说，第一个 VFS 索引节点是代表/目录的索引节点。这种对应关系对于 ext2 文件系统来说十分有效，但对于其他的文件系统则效果一般。

当系统中的进程存取目录和文件时，需要调用系统中的子程序来遍历搜索系统中的 VFS 索引节点。

例如，键入 ls 或者 cat 命令将导致 VFS 搜索整个文件系统中的 VFS 索引节点。因为系统中的每一个文件和目录都由一个索引节点来表示，那么有些索引节点将经常会被重复地搜索。这些索引节点将保存在索引节点缓存中，这样将增快以后的存取速度。如果要找的索引节点不在索引节点缓存中，那么进程将调用一个特殊的系统程序来读取相应的索引节点。读取了索引节点以后，此索引节点将放在索引节点缓存中。最少用到的索引节点将被交换出索引节点缓存。

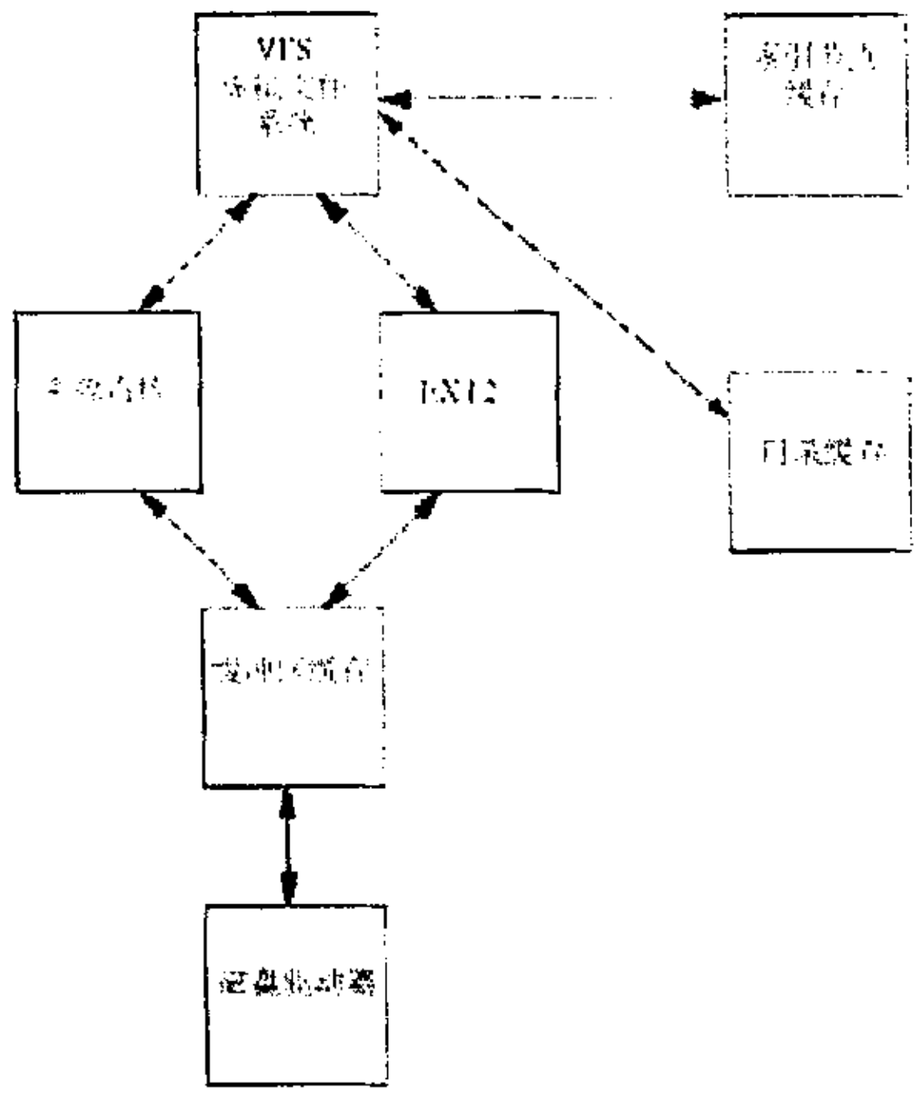


图2.5 VFS系统结构示意图

所有的 Linux 系统中的文件系统都使用一个相同的缓冲区缓存来保存相应的块设备中的数据缓冲区，这样可以加速所有的文件系统对其物理设备的存取。

这个缓冲区缓存和 Linux 系统中的各个不同的文件系统无关，并且缓冲区缓存集成到了 Linux 内核用来分配和读写数据缓冲区的机制中。把 Linux 中的各个文件系统和其相应的物理设备分开是有很大好处的。所有的块设备都在 Linux 系统内核中注册，并且提供一个相同的，以数据块为基础的，一般情况下是异步的接口。当实际文件系统从相应的物理设备中读取数据时，将导致文件系统控制的物理设备读取它的物理块。当文件系统读取数据块时，它们把数据块保存在所有文件系统和系统内核共同使用的公共的缓冲区缓存中。缓冲区缓存中的缓冲区以它们的数据块号和读取设备的标识符名来区分。所以，如果需要一些相同的数据，那么这些数据将从系统的缓冲区缓存而不是磁盘中读出，这就加快了读取的速度。VFS 中也保存了一个目录查找缓存，一些经常使用的目录的索引节点将会很快地找到。目录缓存中并不保存目录的索引节点本身，索引节点保存在索引节点缓存中，目录缓存只是简单地保存目录的名字和目录的索引节点号的对应关系。

2.1.2.3 /proc 文件系统

/proc 文件系统是最能显示 Linux 的 VFS 作用的文件系统。它实际上并不存在，并且/proc 目录和它下面的子目录以及其中的文件也不存在。但和其他实际存在的文件系统一样，/proc 文件系统也在 VFS 中注册。当 VFS 调用/proc 文件系统中打开的目录或者文件的索引节点时，/proc 文件系统才利用系统内核中的有关信息创建这些文件和目录。例如，/proc/device 文件是从内核中描述系统设备的数据结构中产生的。/proc 文件系统提供给用户一个可以了解内核内部工作过程的可读窗口。

2.1.3 可调内核参数

遵循 Unix 的 BSD4.4 版本所倡导的风格，Linux 提供 sysctl 系统调用以便在系统运行过程中对它所拥有的某些特性进行检查和重新配置，而且这并不需要编辑内核的源代码、重新编译，然后重启机器。Linux 把可以被检查和重新配置的系统特性有机地组织成了几个种类：常规内核参数、虚拟内存参数、网络参数等等。

同样的特性也可以从另一个不同的接口进行访问：/proc 文件系统。每种可调内核参数在/proc/sys 下都表现为一个子目录，而每个单独的可调系统参数由某个子目录下的一个文件来代表。

/proc/sys 绕过了通常的 sysctl 接口：一个可调内核参数的值可以简单的通过读取相应的文件来得到，通过写入该文件可以设置它的值。普通 Linux 文件系统的许可被应用于这些文件，以便对可能读写它们的用户进行控制。大多数文件对所有用户是可读的但是只对 root 可写，但/proc/sys/vm 下的文件（虚拟内存参数）只能被 root 读写。如果不使用/proc/sys，检查和调整系统将需要编写程序并使用必需的参数调用 sysctl，显然，这比不上使用/proc/sys 来得方便。

2.2 Linux 网络系统概述

网络和 Linux 系统几乎可以说是同义词，因为 Linux 系统就是 WWW 的产物，基于 Linux 的网络服务器是 Linux 应用方面极为成功的范例，并且广泛用于商业领域。

2.2.1 网络接口总体分析

Linux 的网络接口分为四部份：网络设备接口部份，网络接口核心部份，网络协议族部份，以及网络接口 socket 层[7]。

网络设备接口部份主要负责从物理介质接收和发送数据。实现的文件在 linux/driver/net 目录下面。

网络接口核心部份是整个网络接口的关键部位，它为网络协议提供统一的发送接口，屏蔽各种各样的物理介质，同时又负责把来自下层的包向适合的协议配送。它是网络接口的中枢部份。它的主要实现文件在 linux/net/core 目录下，其中 linux/net/core/dev.c 为主要管理文件。

网络协议族部份是各种具体协议实现的部份。Linux 支持 TCP/IP，IPX，X.25，AppleTalk 等多种协议，各种具体协议实现的源码在 linux/net/目录相应的名称下。本项目着重研究的 TCP/IP (IPv4)协议，实现的源码在 linux/net/ipv4 下,其中 linux/net/ipv4/af_inet.c 是主要的管理文件。

网络接口 Socket 层是为用户提供网络服务的编程接口。主要的源码在 linux /net/ socket.c 中。

下面着重讨论 Linux 系统是如何支持 TCP/IP 协议的。

TCP/IP 协议最初用来支持计算机和 ARPANET 网络之间通信。在 UNIX 系统中，首先带有网络功能的版本是 4.3BSD。Linux 系统的网络功能就是以 UNIX 4.3BSD 为模型发展起来的，它支持 BSD 套接口和全部的 TCP/IP 功能。这样 UNIX 系统中的软件就可以十分方便地移植到 Linux 系统中了。

2.2.1.1 TCP/IP 网络的分层

图 2-6 显示了 Linux 系统网络实现的分层结构。BSD 套接口是最早的网络通信实现，它由一个只处理 BSD 套接口的管理软件支持。其下面是 INET 套接口层，它管理 TCP 协议和 UDP 协议的通信末端。UDP(User Datagram Protocol)是无连接的协议，而 TCP 则是一个可靠的端到端协议。当网络中传送一个 UDP 数据包时，Linux 系统不知道也不关心这些 UDP 数据包是否安全地到达目的节点。TCP 数据包是编号的，同时 TCP 传输的两端都要确认数据包的正确性。IP 协议层是用来实现网间协议的，其中的代码要为上一层数据准备 IP 数据头，并且要决定如何把接收到的 IP 数据包传送到 TCP 协议层或者 UDP 协议层。在 IP 协议层的下方是支持整个 Linux 网络系统的网络设备，例如 PPP 和以太网。网络设备并不完全等同于物理设备，因为一些网络设备，例如回环设备是完全由软件实现的。和其他那些使用 mknod 命令创建的 Linux 系统的标准设备不同，网络设备只有在软件检测到和初始化这些设备时才在系统中出现。当构建系统内核时，即使系统中有相应的以太网设备驱动程序，也不能看到/dev/eth0。ARP

协议在 IP 协议层和支持 ARP 翻译地址的协议之间。

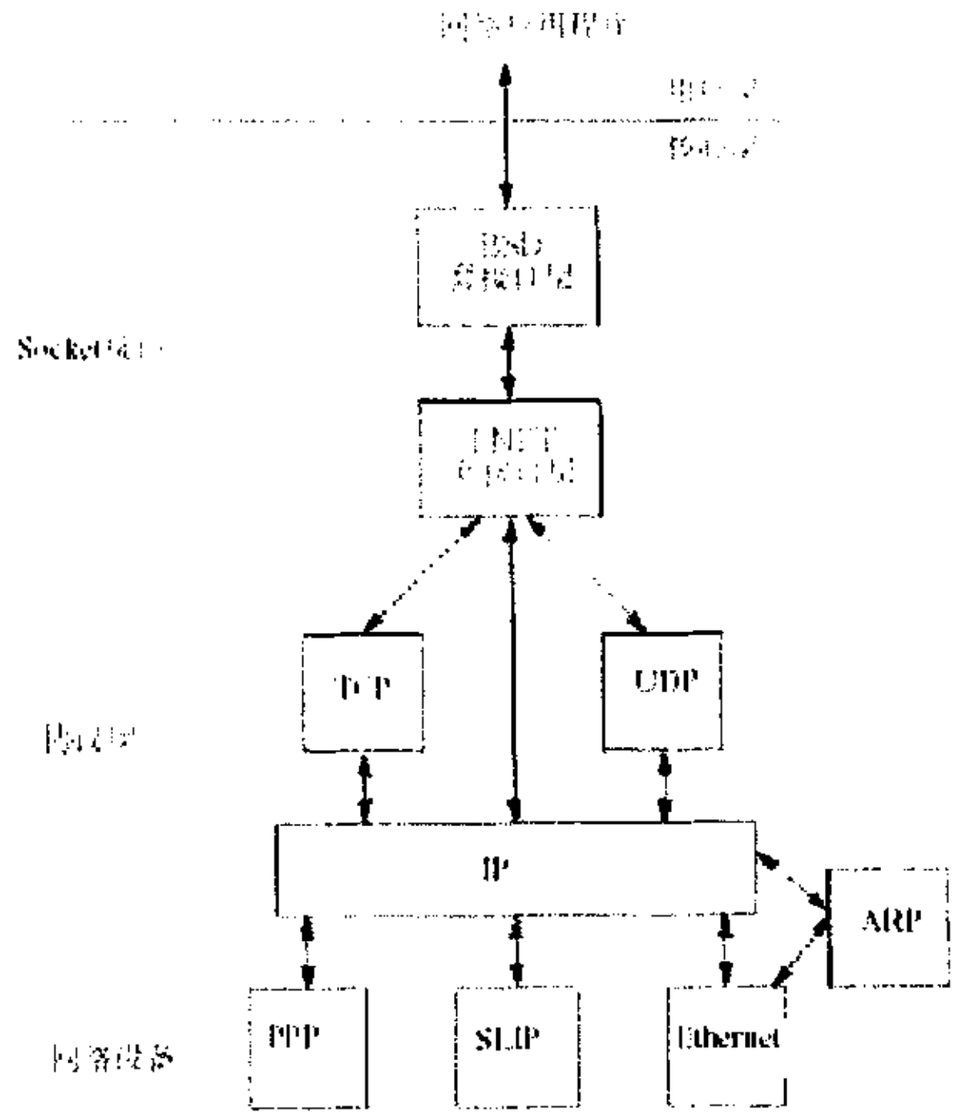


图2-6 Linux中TCP/IP结构分层示意图

2.2.1.2 BSD 套接口

BSD 是 UNIX 系统中通用的网络接口，它不仅支持各种不同的网络类型，而且也是一种内部进程之间的通信机制。两个通信进程都用一个套接口来描述通信链路的两端。套接口可以认为是一种特殊的管道，但和管道不同的是，套接口对于可以容纳的数据的大小没有限制。Linux 支持多种类型的套接口，也叫做套接口寻址族，这是因为每种类型的套接口都有自己的寻址方法。Linux 支持以下的套接口类型：

UNIX	UNIX 域套接口
INET	Internet 地址族 TCP/IP 协议支持通信。
AX25	Amateur radio X25
IPX	Novell IPX
APPLETALK	Appletalk DDP
X25	X25

这些类型的套接口代表各种不同的连接服务。使用 BSD 套接口的确切含义就在于套接口所使用的地址族。设置一个 TCP/IP 连接就和设置一个业余无线电 X.25 连接有很大的不同。和 VFS 一样，Linux 从 BSD 套接

口协议层中抽象出了套接口界面，此界面负责和各种不同的应用程序之间进行通信。内核初始化时，内核中的各个不同的地址族将会在 BSD 套接口界面中登记。稍后当应用程序创建和使用 BSD 套接口时，就将会在 BSD 套接口和它支持的地址族之间建立一个连接。

利用套接口进行通信的进程使用的是客户机/服务器模式。服务器用来提供服务，而客户机可以使用服务器提供的服务，例如一个提供 web 页服务的 Web 服务器和一个读取并浏览 web 页的浏览器。服务器首先创建一个套接口，然后给它指定一个名字。名字的形式取决于套接口的地址族。系统使用数据结构 sockaddr 来指定套接口的名字和地址。一个 INET 套接口可以包括一个端口地址。Linux 中可以在/etc/services 中查看已经注册的端口号，例如，一个 web 页面服务器的端口号是 8 0。在服务器指定套接口的地址以后，它将监听和此地址有关的连接请求。请求的发起者，也就是客户机，将会创建一个套接口，然后再创建连接请求，并指定服务器的目的地址。对于一个 INET 套接口来说，服务器的地址就是它的 IP 地址和端口号。这些连接请求必须通过各种协议层，然后等待服务器的监听套接口。一旦服务器接收到了连接请求，它将接受或者拒绝这个请求。如果服务器接受了连接请求，它将创建一个新的套接口。一旦服务器使用一个套接口来监听连接请求，它就不能使用同样的套接口来支持连接。当连接建立起来以后，连接的两端都可以发送和接收数据。最后，当不再需要此连接时，可以关闭此连接。

2.2.1.3 INET 套接口层

INET 套接口层包括支持 TCP/IP 协议的 Internet 地址族。正如上面提到的，这些协议是分层的，每一个协议都使用另一个协议的服务。Linux 系统中的 TCP/IP 代码和数据结构也反映了这种分层的思想。它和 BSD 套接口层的接口是通过一系列与 Internet 地址族有关的套接口操作来实现的，而这些套接口操作是在网络初始化的过程中由 INET 套接口层在 BSD 套接口层中注册的。这些操作和其他地址族的操作一样保存在 pops 向量中。BSD 套接口层通过 INET 的 proto_ops 数据结构来调用与 INET 层有关的套接口子程序来实现有关 INET 层的服务。例如，当 BSD 套接口创建一个发送给 INET 地址族的请求时将会使用 INET 的套接口创建功能。BSD 套接口层将会把套接口数据结构传递给每一个操作中的 INET 层。INET 套接口层在它自己的数据结构 sock 中而不是在 BSD 套接口的数据结构中插入有关 TCP/IP 的信息，但 sock 数据结构是和 BSD 套接口的数据结构有关

的。它使用 BSD 套接口中的数据指针来连接 sock 数据结构和 BSD 套接口数据结构，这意味着以后的 INET 套接口调用可以十分方便地得到 sock 数据结构。数据结构 sock 中的协议操作指针也会在创建时设置好，并且此指针是和所需要的协议有关的。如果需要的是 TCP 协议，那么数据结构 sock 中的协议操作指针将会指向一系列的 TCP 协议操作。

2.2.2 netfilter 防火墙功能框架

Netfilter 是 linux2.4 内核中实现数据包过滤、数据包处理、NAT 等功能的防火墙框架，它比以前任何一版 Linux 内核的防火墙子系统都要完善强大[1]。Netfilter 提供了一个抽象、通用化的框架，该框架定义的一个子功能的实现就是包过滤子系统。Netfilter 框架包含以下三个特性：

（1.）为每种网络协议(IPv4、IPv6 等)定义一套钩子函数（IPv4 定义了 5 个钩子函数）,这些钩子函数在数据报流过协议栈的几个关键点被调用。在这几个点中，协议栈将把数据报及钩子函数标号作为参数调用 netfilter 框架。

（2.）内核的任何模块可以对每种协议的一个或多个钩子进行注册，实现挂接，这样当某个数据包被传递给 netfilter 框架时，内核能检测是否有任何模块对该协议和钩子函数进行了注册。若注册了，则调用该模块注册时使用的回调函数，这样这些模块就有机会检查(可能还会修改)该数据包、丢弃该数据包以及指示 netfilter 将该数据包传入用户空间的队列。

（3）那些排队的数据包可以被传递给用户空间异步地进行处理。一个用户进程能检查数据包，修改数据包。

所有的包过滤/NAT 等都基于该框架。内核网络代码中不再有到处都是的、混乱的修改数据包的代码。当前 netfilter 框架在 IPv4 和 IPv6 网络栈中都被实现。

2.2.2.1 netfilter 分析

Netfilter/iptable 的产生源于 Linux2.2 内核中 ipchains 的某些不合理性。下面是其中的几点原因：

（1）基于 2.2 内核的 ipchains 没有提供传递数据包到用户空间的框架，所以任何需要对数据包进行处理的代码都必须运行在内核空间，而内核编程却非常复杂，而且只能用 C 语言实现，并且容易出现错误并对内核稳定性造成威胁。

（2）透明代理实现非常复杂，必须查看每个数据包来判断是否有专

门处理该地址的 socket。

（3）创建一个不依赖于接口地址的数据报过滤规则是不可能实现的。我们必须利用本地接口地址来判断数据报是本地发出、还是发给本地或是转发的。转发链只有输出接口的信息，因此管理员必须考虑数据报的源。

（4）伪装和数据包过滤都在同一个模块内实现，导致防火墙代码过于复杂。

（5）IPchains 代码即不模块化又不易于扩展(例如对 mac 地址的过滤)。

下面着重分析 Netfilter 在 IPv4 中的结构。

一个数据包按照如图 2.7 所示的过程通过 Netfilter 框架：

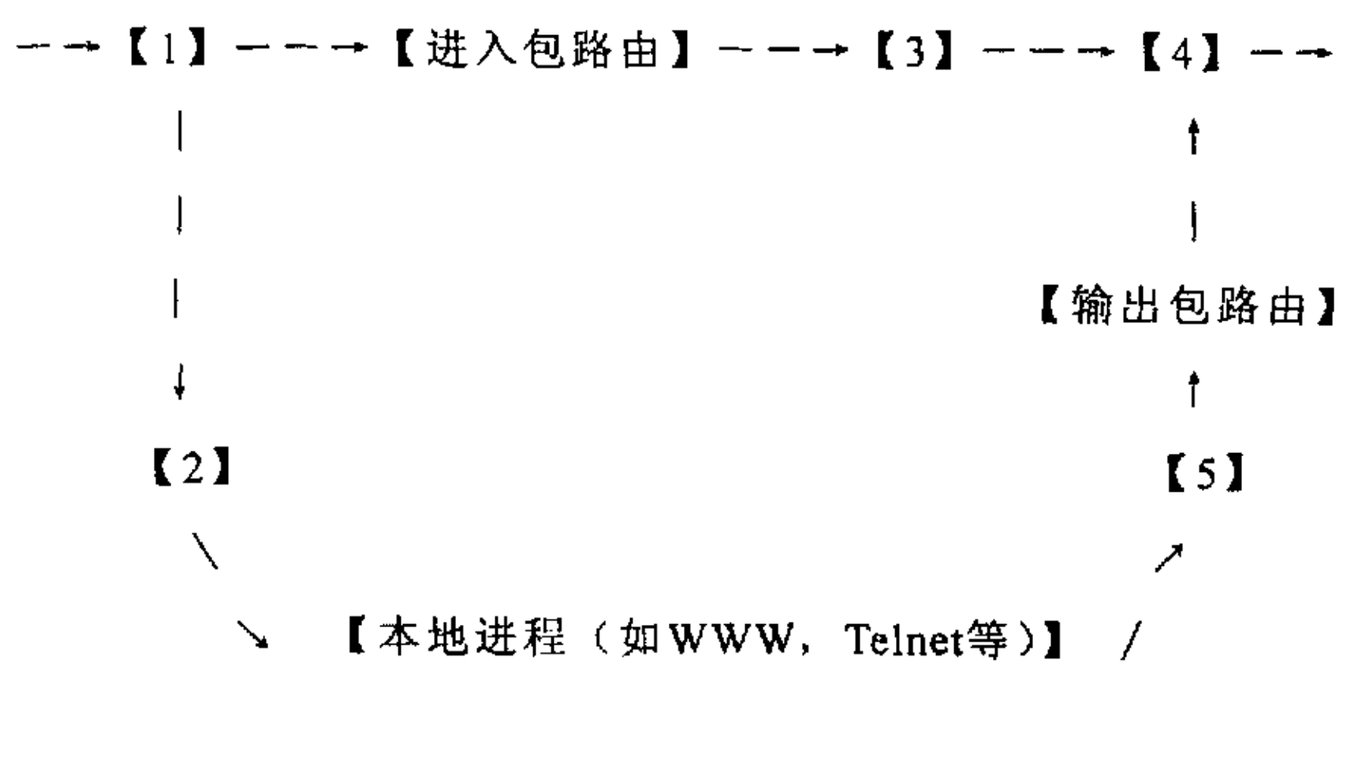


图2.7 Netfilter框架示意图

从图中可以看到 IPv4 一共有 5 个钩子函数，分别为：

1 NF_IP_PRE_ROUTING

2 NF_IP_LOCAL_IN

3 NF_IP_FORWARD

4 NF_IP_POST_ROUTING

5 NF_IP_LOCAL_OUT

数据报从左边进入系统，进行 IP 校验以后，数据报经过第一个钩子函数 NF_IP_PRE_ROUTING[1]进行处理；然后就进入路由代码，其决定该数据包是需要转发还是发给本机的；若该数据包是发给本机的，则该数据经过钩子函数 NF_IP_LOCAL_IN[2]处理以后然后传递给上层协议；若该数据包应该被转发则它被 NF_IP_FORWARD[3]处理；经过转发的数据报经过最后一个钩子函数 NF_IP_POST_ROUTING[4]处理以后，再传输到网络上。

本地产生的数据经过钩子函数 NF_IP_LOCAL_OUT [5]处理可以后，进行路由选择处理，然后经过 NF_IP_POST_ROUTING[4]处理以后发送到网络上。

从上面关于 IPv4 的 netfilter 结构讨论，可以看到钩子函数是如何被激活的。

内核模块可以对一个或多个这样的钩子函数进行注册挂接，并且在数据报经过这些钩子函数时被调用，从而模块可以修改这些数据报，并向 netfilter 返回如下值：

NF_ACCEPT 继续正常传输数据报

NF_DROP 丢弃该数据报，不再传输

NF_STOLEN 模块接管该数据报，不要继续传输该数据报

NF_QUEUE 对该数据报进行排队(通常用于将数据报给用户空间的进程进行处理)

NF_REPEAT 再次调用该钩子函数

2.2.2.2 netfilter 配置工具 iptable

除了用户想在 netfilter 框架基础上编写自己的防火墙处理程序外，在利用 netfilter 架构防火墙时，使用得最多的应该是 netfiter 的用户配置工具 iptables。其实它就是 ipchains 的后继工具，但却有更强的可扩展性。

内核模块可以注册一个新的规则表(table)，并要求数据报流经指定的规则表。这种数据报选择用于实现数据报过滤(filter 表)，网络地址转换(NAT 表)及数据报处理(mangle 表)。

Linux2.4 内核提供的这三种数据报处理功能都基于 netfilter 的钩子函数和 IP 表。它们是独立的模块，相互之间是独立的。它们都完美的集成到由 Netfileter 提供的框架中。

包过滤（filter）

filter 表格不会对数据报进行修改，而只对数据报进行过滤。iptables 优于 ipchains 的一个方面就是它更为小巧和快速。它是通过钩子函数 NF_IP_LOCAL_IN， NF_IP_FORWARD 及 NF_IP_LOCAL_OUT 接入 netfilter 框架的。因此对于任何一个数据报只有一个地方对其进行过滤。这相对 ipchains 来说是一个巨大的改进，因为在 ipchains 中一个被转发的数据报会遍历三条链。

NAT

NAT 表格监听三个 Netfilter 钩子函数：NF_IP_PRE_ROUTING，

NF_IP_POST_ROUTING 及 NF_IP_LOCAL_OUT。NF_IP_PRE_ROUTING 实现对需要转发的数据报的源地址进行地址转换，而 NF_IP_POST_ROUTING 则对需要转发的数据包的目的地址进行地址转换。对于本地数据报的目的地址的转换则由 NF_IP_LOCAL_OUT 来实现。

NAT 表格不同于 filter 表格，因为只有新连接的第一个数据报将遍历表格，而随后的数据报将根据第一个数据报的结果进行同样的转换处理。

NAT 表格被用在源 NAT,目的 NAT，伪装(其是源 NAT 的一个特例)及透明代理(是目的 NAT 的一个特例)。

数据报处理(Packet mangling)

mangle 表格在 NF_IP_PRE_ROUTING 和 NF_IP_LOCAL_OUT 钩子中进行注册。使用 mangle 表，可以实现对数据报的修改或给数据报附上一些带外数据。当前 mangle 表支持修改 TOS 位及设置 skb 的 nfmard 字段。

这三张表是防火墙处理程序的基础。由于每个功能在 NetFilter 中对应一个表，而每个检查点又有若干个匹配规则，这些规则组成一个链，所以就有这样的说法："NetFilter 是表的容器，表是链的容器，链是规则的容器"

一个链(chain)其实就是众多规则(rules)中的一个检查清单(checklist)。每一条链中可以有一条或数条规则，每一条规则都是这样定义的"如果数据包头符合这样的条件，就这样处理这个数据包"。当一个数据包到达一个链时，系统就会从第一条规则开始检查，看是否符合该规则所定义的条件：如果满足,系统将根据该条规则所定义的方法处理该数据包；如果不满足则继续检查下一条规则。最后，如果该数据包不符合该链中任一条规则的话，系统就会根据该链预先定义的策略(policy)来处理该数据包。

而一个 iptables 命令基本上包含如下五部分：希望工作在哪个表上、希望使用该表的哪个链、进行的操作(插入，添加，删除，修改)、对特定规则的目标动作和匹配数据包条件。

基本的语法为：iptables -t table -Operation chain -j target match(es)（系统缺省表为"filter"）

此外，在 2.4 内核 netfilter 框架中，连接跟踪已经作为一个单独的模块被实现。该功能是包过滤、NAT 的基础，用于对包过滤功能的一个扩展，使用连接跟踪来实现"基于状态"的防火墙。

第三章 Linux 安全机制及在特权管理中的应用

Linux 是一种多用户、多任务的操作系统。这类操作系统的一个基本功能就是防止使用同一台计算机的不同用户之间互相干扰，所以，Linux 的设计宗旨就考虑到安全问题。当然，Linux 中仍然存在很多安全问题，其新功能的不断加入及安全机制的错误配置或错误使用，都可能带来很多问题。

3.1 Linux 安全结构分析

Linux 的现有版本无需任何修改，就基本符合美国国防部《可信任计算机系统评价准则（TCSEC）》的 C1 级标准。它主要有以下安全特性：

（1）系统中每个文件和目录都具有不同于其他文件和目录的唯一标识符，系统中的每个用户也都有一个可相互区分的标识符。

（2）用户必须通过注册名和口令经系统识别无误后，才可登录系统，没有合法用户的口令，非法用户将无法进入系统。

（3）系统具有一定的自主型存取控制（DAC）安全机制，即“owner/group/other”存取控制机制，用户对系统中的文件和目录拥有相应的许可权限，系统能够控制用户访问哪些程序或信息以及如何访问。

（4）Linux 系统内核在一个物理上的安全域中运行，这个域受到硬件的保护，安全域保护它内部的核心和安全机制，安全机制本身是无法绕过的。

（5）Linux 系统运行态分用户态和核心态两种，系统调用是用户程序进入 Linux 内核的唯一入口，一旦用户程序通过系统调用进入内核，便完全与用户隔离，从而使内核中的程序可对用户的存取请求进行不受用户干扰的访问控制。因此，Linux 支持存取控制机制的开发，从而支持安全性的开发。

从安全结构来看，Linux 设置了多层防御体系，如图 3.1 所示：

在一个系统中，第一道防线是物理安全。若系统对公众开放且没有监督，则没有任何安全性可言。侵入者可以打开机箱，抽出硬盘，仅仅只需简单的插入另一台计算机中，就可以获得大量的信息。

假设系统通过某种方法保证了物理上的安全，如门锁或通行证，则下一层防御就是帐号安全。这里，用户名和口令的组合是关键。如果攻击者获得帐号和口令。甚至是 Root 帐号的口令，系统将面临巨大的风险。

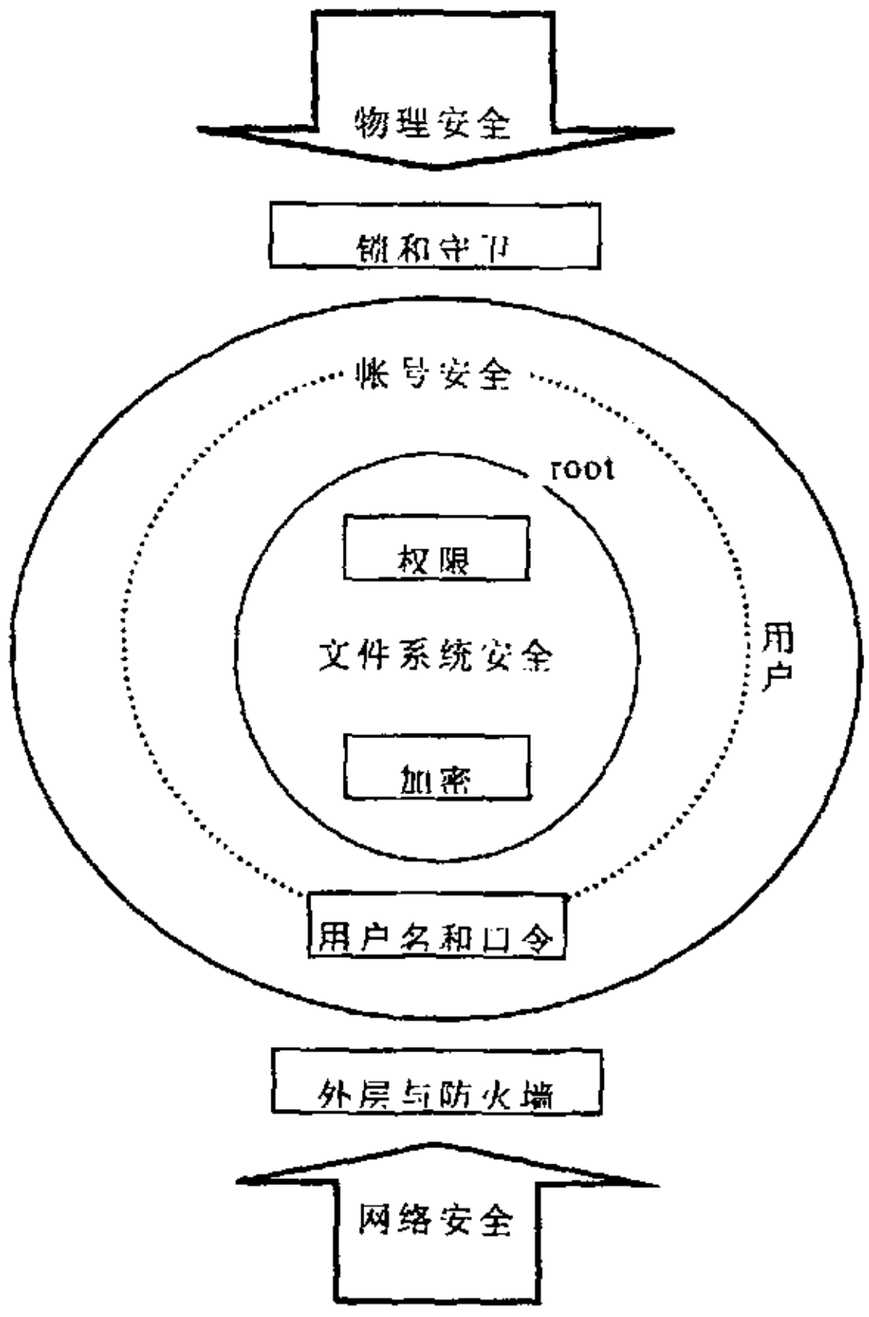


图3.1　Linux安全多层防御机制

最内层是文件系统安全。文件系统甚至在帐号被突破的情况下也能保护用户信息。Linux 的权限机制发挥着巨大的作用。如果攻击者进入了一个帐号，但关键信息仍然被权限保护着，该帐号只被赋予了有限的访问权限。使文件安全的另一个重要方法是加密。如果重要文件被加密，即使侵入者进入了一个帐号，甚至突破了权限机制，但仍然不能使用加密的文件。

最外层的防御边界是网络安全，它维护通信安全并保证只有被授权的用户才能访问系统。防火墙是公认的边界防御。象 FTP 和 Telnet 这些常规服务的安全可以由 TCP wrapper 这样的外层程序进行调用来得到支持。边界防御也可以用扫描程序来测试。

在 GNU/Linux 系统中不完善之处主要表现在以下几个方面[12]。

1．超级用户可能滥用权限。作为一个超级用户，它可以做任何事情，包括删除不该删除的系统文档、杀死系统进程以及改变权限等等。

2．系统文档可以被任意地修改。在 Linux 系统中，有许多的重要文件，比如/bin/login，如果入侵者修改该文件，就可以轻易地再次登录。

3．系统内核可以轻易插入模块。系统内核允许插入模块，使用户扩展 Linux 操作系统的功能，使 Linux 系统更加模块化，但这样做是十分危

险的。模块插入内核后，就成为内核的一部分，可以做原来内核所能做的任何事情。

4. 进程不受保护。某些重要的进程，保护不完善，使用命令 kill –9，可以很容易的删除。

5. 网络应用不安全。Linux 作为网络操作系统虽然提供了许多网络应用，例如 telnet、WWW、E_mail 等，这些无安全保护的应用给使用者带来了方便，同样也给攻击者带来了方便。

3.2 Linux 内核能力（capability）

Linux 是一种比较安全的操作系统，它给普通用户尽可能低的权限，而把全部的系统权限赋予一个单一的帐户--root。root 帐户用来管理系统、安装软件、管理帐户、运行某些服务、安装/卸载文件系统、管理用户、安装软件等。另外，普通用户的很多操作也需要 root 权限，这通过 setuid 实现。

这种依赖单一帐户执行特权操作的方式加大了系统面临的风险，而需要 root 权限的程序可能只是为了一个单一的操作，例如：绑定到特权端口、打开一个只有 root 权限可以访问的文件。这些程序可能有安全漏洞，而如果程序不是以 root 的权限运行，其存在的漏洞就不可能对系统造成较大威胁。

解决这个问题的方法是从 POSIX.le 标准中抽取出来的思想——能力（capability），又叫最小特权原则（principle of least privilege）。它的意思是指为完成特定任务，授予主体所需要的最小访问特权的过程、策略。从 2.1 版本开始，内核开发人员在 Linux 内核中加入了能力(capability)的概念。其目标是消除需要执行某些特殊操作的程序对 root 帐户的依赖。从 2.2 版本的内核开始，这些功能基本可以使用了，虽然还不太完善，但是却大大加强了操作系统自身的健壮性与安全性。

能力使你可以更精确的定义经授权的进程所允许处理的事情。例如，你可以给一个进程授予修改系统时间的权力，而没有授予它可以杀掉系统中的其他进程、毁坏你的文件并胡乱运行的权力。而且，为了防止意外的滥用其特权，长时间运行的进程可以暂时获得能力，只要时间足够处理特殊的零散工作就可以了。在处理完这个零散的工作以后再收回能力。

3.2.1 能力(capability)的概念与应用

传统 UNIX 的信任状模型非常简单，就是“超级用户对普通用户”模型。在这种模型中，一个进程要么什么都能做，要么几乎什么也不能做，这取决于进程的 UID。如果一个进程需要执行绑定到私有端口、加载/卸载内核模块以及管理文件系统等操作时，就需要完全的 root 权限。很显然这样做对系统安全存在很大的威胁。UNIX 系统中的 SUID 问题就是由这种信任状模型造成的。例如，一个普通用户需要使用 ping 命令。这是一个 SUID 命令，会以 root 的权限运行。而实际上这个程序只是需要 RAW 套接字建立必要 ICMP 数据包，除此之外的其它 root 权限对这个程序都是没有必要的。如果程序编写不完善，就可能被攻击者利用，获得系统的控制权。使用能力(capability)可以减小这种风险。系统管理员为了系统的安全可以剥夺 root 用户的能力，这样即使 root 用户也将无法进行某些操作。而这个过程又是不可逆的，也就是说如果一种能力被删除，除非重新启动系统，否则即使 root 用户也无法重新添加被删除的能力。

Linux 内核中使用的能力(capability)就是一个进程能够对某个对象进行的操作，它标志对象以及允许在这个对象上进行的操作[20]。POSIX 1003.1e 中也提出了一种能力定义，通常称为 POSIX 能力(POSIX capabilities)，与 Linux 中的定义在范围上稍有差别。内核使用这些能力分割 root 的权限，因为传统 UNIX 系统中 root 的权限过于强大了。

每个进程有三个和能力有关的位示：inheritable(I)、permitted(P)和 effective(E)，对应进程描述符 task_struct(include/linux/sched.h)里面的 cap_effective, cap_inheritable, cap_permitted。每种能力由一位表示，1 表示具有某种能力，0 表示没有。当一个进程要进行某个特权操作时，操作系统会检查 cap_effective 的对应位是否有效，而不再是检查进程的有效 UID 是否为 0。例如，如果一个进程要设置系统的时钟，Linux 的内核就会检查 cap_effective 的 CAP_SYS_TIME 位是否有效。cap_permitted 表示进程能够使用的能力。在 cap_permitted 中可以包含 cap_effective 中没有的能力，这些能力是被进程自己临时放弃的，也可以说 cap_effective 是 cap_permitted 的一个子集。进程放弃没有必要的能力对于提高安全性大有助益。例如，ping 只需要 CAP_NET_RAW，如果它放弃除这个能力之外的其它能力，即使存在安全缺陷，也不会对系统造成太大的损害。cap_inheritable 表示能够被当前进程执行程序继承的能力。

Linux 实现了 7 种 POSIX 1003.1e 规定的能力，还有 21 种(2.4 版本的

内核)Linux 所特有的，每一种能力用一个 32 位整数表示。这些能力在/usr/src/linux/include/linux/capability.h 文件中定义。一些较重要的能力如下：

能力名	能力值号	描述
CAP_CHOWN	0	允许改变文件的所有权
CAP_FSETID	4	允许设置setuid位
CAP_KILL	5	允许对不属于自己的进程发送信号
CAP_SETPCAP	8	允许向其它进程转移能力以及删除其它进程的任意能力
CAP_LINUX_IMMUTABLE	9	允许修改文件的不可修改(i)和只添加(a)属性
CAP_NET_BIND_SERVICE	10	允许绑定到小于1024的端口
CAP_NET_RAW	13	允许使用原始(raw)套接字
CAP_SYS_MODULE	16	插入和删除内核模块
CAP_SYS_PTRACE	19	允许跟踪任何进程
CAP_SYS_TIME	25	允许改变系统时钟
CAP_SYS_TTY_CONFIG	26	允许配置TTY设备

Linux2.2 内核提供了对能力的基本支持。但是在引入了能力之后面临一个问题，2.2 版本的内核能够支持能力，却缺乏一个系统和用户之间的接口。从 2.2.11 版本开始，这种情况发生了很大的改观，在这个版本中引入了能力边界集(capability bounding set)的概念，解决了系统和用户之间的接口问题。能力边界集(capability bounding set)是系统中所有进程允许保留的能力。如果在能力边界集中不存在某个能力，那么系统中的所有进程都没有这个能力，即使以超级用户权限执行的进程也一样。

能力边界集通过 sysctl 命令导出，可以在/proc/sys/kernel/cap-bound 中看到系统保留的能力。在默认情况下，能力边界集所有的位都是打开的。

root 用户可以向能力边界集中写入新的值来修改系统保留的能力。但是，root 用户能够从能力边界集中删除能力，却不能再恢复被删除的能力，只有 init 进程能够添加能力。通常，一个能力如果从能力边界集中被删除，只有系统重新启动才能恢复。

删除系统中多余的能力对提高系统的安全性是很有好处的。例如有一台重要的服务器，担心可加载内核模块威胁系统安全性，但又不想完全禁止在系统中使用可加载内核模块，或者一些设备驱动就是以内核模块形式加载。在这种情况下，最好使系统在启动时加载所有的模块，然后禁止加载/卸载任何内核模块。在 Linux 系统中，加载/卸载内核模块是由 CAP_SYS_MODULE 能力控制的。如果把 CAP_SYS_MODULE 从能力边

界集中删除，系统将不再允许加载/卸载任何的内核模块。CAP_SYS_MODULE 能力的值是 16，因此使用下面的命令就可以把它从能力边界集中删除：

```
echo 0xFFFEFFFF >/proc/sys/kernel/cap-bound
```

虽然可以直接修改/proc/sys/kernel/cap-bound 来删除系统的某种能力，但是这样却并不方便。lcap 程序提供了一种方便的从系统中删除指定能力的方式。如果不带参数，lcap 可以列出系统当前支持的各种能力：

```
[root@nixe0n.uestc]# ./lcap
Current capabilities: 0xFFFFFEFF
```

如果需要删除某个能力，直接把能力名作为参数，例如要删除加载/卸载内核模块的能力：

```
[root@nixe0n.uestc]# ./lcap CAP_SYS_MODULE
[root@nixe0n.uestc]# ./lcap
Current capabilities: 0xFFFBFEFF
```

3.2.2 基于内核能力的程序扩展

目前，Linux 内核能力仍然处于开发状态，有很多功能还没有实现。虽然利用能力已经可以有效地保护系统的安全，但是由于文件系统的制约，Linux 的能力控制还不是很完善，往往仅限于使用 lcap 从总体上放弃一些能力，而并不支持将程序的能力附加到文件本身中。这样 Linux 有时仍要检测进程是否作为 root 运行，而不是检测进程所需要的特殊能力。本文在这方面作了一个尝试，基于内核能力开发特权控制程序，希望对内核能力做一些扩展。

当一个用户 login 进入系统时，系统就会 fork()一个进程并让它执行/bin/bash，而这个 shell 进程就成为该用户启动的所有进程的祖先。如果把 login 程序修改，在用户进入系统时，就赋予 shell 进程相应的能力，则这个用户启动的所有进程都要受其影响。这样，就把能力和用户联系起来了。其特权控制模型如图 3.2 所示：

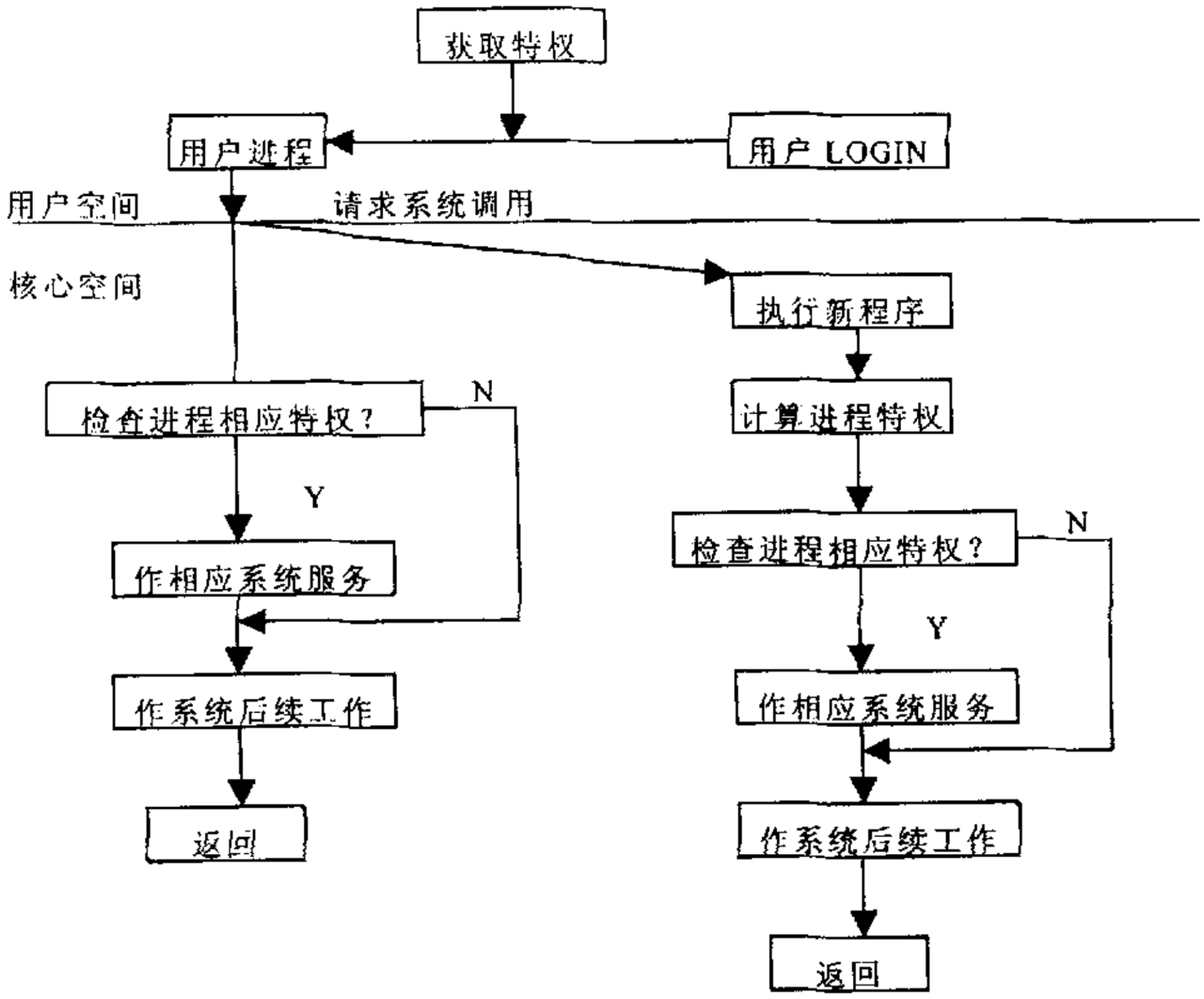


图3.2 特权控制模型

检查进程中权能的函数为 capable()，其代码如下：

```
/*
 *capable() checks for a particular capability.
 *New privilege checks should use this interface, rather than suser() or
 *fsuser().see include/linux/capability.h for defined capabilities.
 */

static inline int capable(int cap)
{
#if 1 /*ok now*/
    if(cap_raised(current->cap_effective,cap))
#else
    if(cap_is_fs_cap(cap)?current->fsuid==0:current->euid==0)
#endif
    {
        current->flags|=PF_SUPERPRIV;
```

```
        return 1;
    }
    return 0;
}
```

cap_raised 函数检查所要求的特权是否在进程的 cap_effective 中，如果在，则设置进程 task_struct 中的标志位为 PF_SUPERPRIV，这样进程就具有了相应的特权。

第四章 Linux 安全配置的实现与扩展

4.1 系统安全配置与优化

实践证明 Linux 是高性能、稳定可靠而又相当灵活的操作系统，目前，Linux 已可以与各种传统的商业操作系统分庭抗礼，在服务器市场，占据了相当大的份额。Linux 的服务器系统多种多样，可用作 web 服务器，邮件服务器，ftp 服务器，文件服务器以及网关服务器等等。针对不同的系统以及具体的应用环境，可以对 Linux 的性能从磁盘分区、文件系统以及编译优化等方面进行相应的调谐和安全加固。

根据项目需求，作为防火墙的支撑系统，需要将 Linux 小型化，对多余的服务和应用程序进行剥离。因此，隔离网络进行系统安装时，选择最小化安装方式，安装必须的软件包，然后再进一步裁减与优化系统。

4.1.1 优化的硬盘分区

在 Linux 系统中，可以自由地组织磁盘分区。一个优化的分区策略，能很好地改进 Linux 系统的性能，减少磁盘碎片，提高磁盘 I/O 能力。

根据磁盘的特点，在分区时，我们应该考虑将访问频率高的，对系统性能影响相对较大的分区置于磁盘的靠外部分。同时，为了减少磁盘碎片，应将内容经常改变的目录放在单独的分区。从方便备份数据的角度考虑，因为很多备份工具对整个分区进行备份的效率要高，所以我们应将 Linux 系统的几个主要的目录作为单独的文件系统，为它们各自分配一个区。另外，如果用 /分区记录数据，如 log 文件和 e-mail，可能因为拒绝服务而产生大量日志或垃圾邮件，导致系统崩溃。所以将 /var 作为单独的分区，用来存放日志和邮件，以避免 /分区被溢出。同时，也为 /home 单独分一个区，避免某些用户恶意操作而填满 /分区，使系统在重负载时的稳定性得到保证。推荐的分区策略如图 4.1 所示：

作为安全系统，对每个分区分别进行不同的配置和安装，将关键的分区设置为只读将大大提高安全性。/usr 可以安装成只读并且可以被认为是不可修改的。如果/usr 中有任何文件发生了改变，那么系统将立即发出安全报警。但是，这并不防碍用户自己需要时可以改变/usr 中的内容。/lib、/boot 和/sbin 的安装和设置也一样，在安装时应该尽量将它们设置为只读，这样对它们的文件、目录和属性进行的任何修改都会导致系统报警。当然

将所有主要的分区都设置为只读是不可能的,有的分区如/var 等，其自身的性质就决定了不能将它们设置为只读，但应该不允许它具有执行权限。

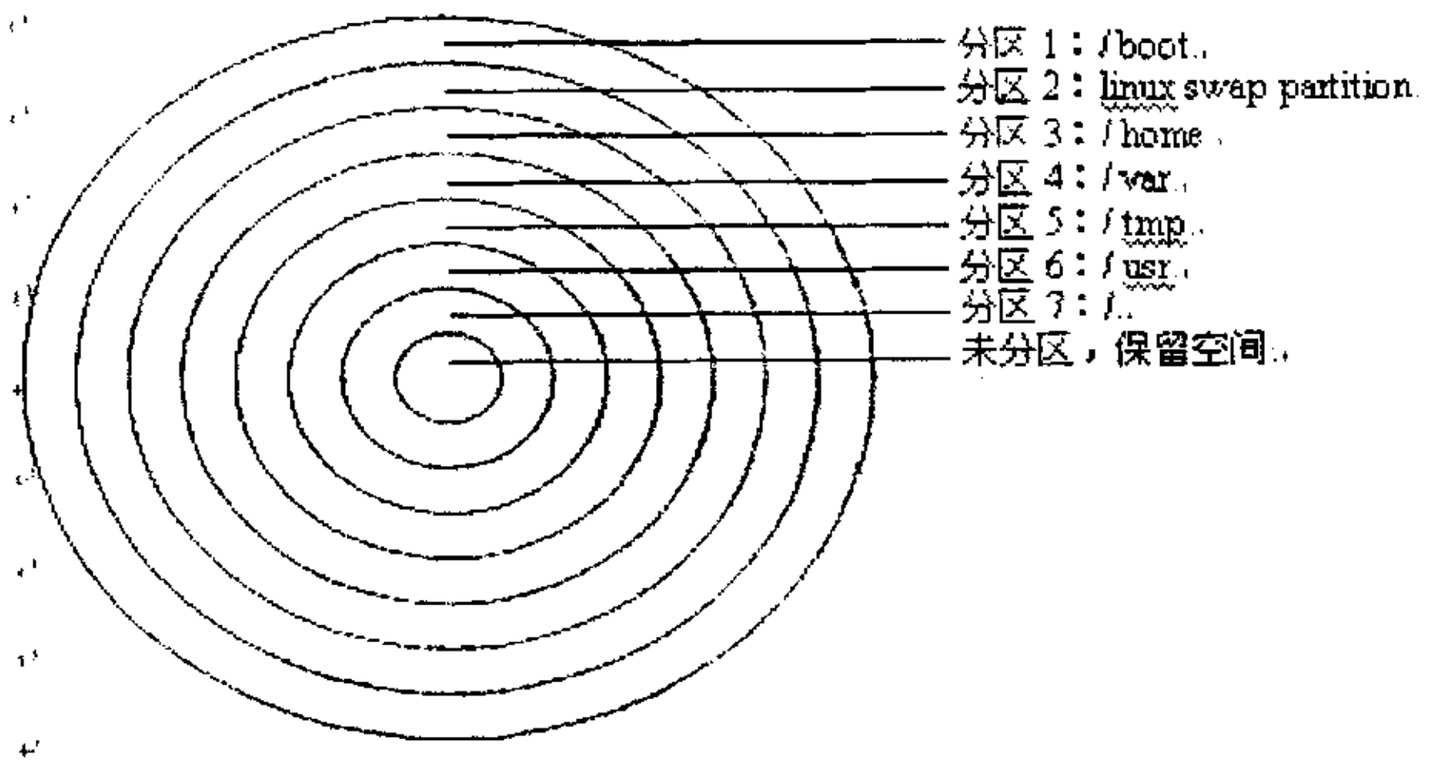


图 4.1 优化的分区

4.1.2 内核与配置文件优化

Linux 内核承担着 Linux 操作系统的最为核心的任务，它负责管理系统的进程、内存、设备驱动程序、文件和网络系统，决定着系统的性能和稳定性。一般各种 Linux 系统套件安装的内核，适合大部分的机器，但缺点是大而全，包含有许多根本不需要的模块。增加了系统被攻击者利用潜在漏洞的可能。只有根据项目的要求及机器硬件配置，来编译配置内核才能达到最优性能

由于作为防火墙系统，注重体系的精干与健壮，因此去掉内核中大部分的辅助功能，如声卡、业余无线广播、Old CD-ROM 设备等等的支持。而加强

- Networking options
- Network device support 这两大选项的配置。

为了对 netfilter 提供支持，主要对 netfilter 编译选项进行说明：

```
[*] kernel/user netlink socket
[*] netlink device emulation
[*] network packet filering (replaces ipchains)
```

其中，IP: netfilter configuration 子项中作如下选择：

[m] connection tracking (required for masq/NAT)

[m] FTP protocol support (NEW)

[m] IP tables support (required for filtering/masq/NAT)(NEW)

[m] MAC address match support

[m] mutiple port match support

[m] TOS match support

[m] packet filtering

[m] REJECT target support

[m] MIRROR target support

[m] full NAT

[m] masq target support

[m] REDIRECT target support

[m] packet mangling

[m] TOS target support

[m] MARK target support

[m] LOG target support

另外，本系统需要使用计算机串口作系统控制台，修改默认配置

Support for console on serial port (CONFIG_SERIAL_CONSOLE) [Y]

针对系统配置文件，做以下修改：

（1）编辑/etc/inittab 文件

大部分 Linux 的发行套件都建立六个虚拟控制台，实际上三个已经足够，并且可以节省可贵的内存空间。编辑/etc/inittab 文件，在下面三行的前面加上#。

#4:2345:respawn:/sbin/mingetty tty4

#5:2345:respawn:/sbin/mingetty tty5

#6:2345:respawn:/sbin/mingetty tty6

然后执行 init q，使系统重新读取/etc/inittab 文件，关闭 tty4、tty5、tty6 虚拟控制台。如果服务器不是处在封闭的地方，在下面一行的前面加上#。

#ca::ctrlaltdel:/sbin/shutdown -t3 -r now

禁止用Ctrl_Alt_del热键关闭计算机。

（2）为 LILO 或 Grub 增加开机口令

在/etc/lilo.conf 或/etc/grub.conf 文件中增加选项，从而使 LILO（Grub）启动时要求输入口令，以加强系统的安全性。由于在 LILO 中口令是以明

码方式存放的，所以还需要将 lilo.conf 的文件属性设置为只有 root 可以读写。

（3）增强历史记录安全性

在 linux 下，系统会自动记录用户输入过的命令，而 root 用户发出的命令往往具有敏感的信息，为了保证安全性，一般应该不记录或者少记录 root 的命令历史记录。为了设置系统不记录每个人执行过的命令，修改 /etc/profile 中的内容。

```
HISTFILESIZE=0
HISTSIZE=0
```

或者 ln -s /dev/null ~/.bash_history。

（4）限制分区权限

修改/etc/fstab，只给予分区必须的权限，如：

```
LABEL=/bakups /bakups ext3 nosuid,noexec 1 2
```

noexec 表示不能在这个分区运行程序，nosuid 表示不能使用 suid 程序，根据情况设置其他分区，一般来说/tmp,/usr 都需要 nosuid。

4.1.3 文件系统的安全性

Linux 内核中有大量安全特征。其中有很多的特征有着广泛的应用，从 Linux1.1 系列内核开始，ext2 文件系统就开始支持一些针对文件和目录的额外标记或者叫作属性(attribute)[10]。在 2.2 和 2.4 系列的内核中，ext2 文件系统支持的只添加（a—Append Only）和不可变（i—Immutable）这两种文件属性可以进一步提高安全级别。a 和 i 属性只是两种扩展 ext2 文件系统安全属性标志的方法。一个标记为不可变的文件不能被修改，甚至不能被根用户修改。一个标记为只添加的文件可以被修改，但只能在它的后面添加内容，即使根用户也只能如此。

在任何情况下，标准的 ls 命令都不会显示一个文件或者目录的扩展属性。ext2 文件系统工具包中有两个工具--chattr 和 lsattr，专门用来设置和查询文件属性。因为 ext2 是标准的 Linux 文件系统，因此几乎所有的发布都有 e2fsprogs 工具包。通过一个实例可以看出 ext2 扩展属性和文件权限的区别：

```
[root@uestc nixe0n]# ls -al test*
-rw-rw-r-- 1 nixe0n users 0 Nov 17 17:02 test.conf
-rw-rw-r-- 1 nixe0n users 0 Nov 17 17:02 test.log
-rw-rw-r-- 1 nixe0n users 0 Nov 16 19:41 test.txt
```

从 ls 的输出结果看，这些文件属于用户 nixe0n，而 nixe0n 所在的用户组是 users。用户 nixe0n 本人和 users 用户组的成员尉有具有对文件的修改权限，而其他的用户只有读取文件的权限。下面是 lsattr 命令的输出：

```
[root@uestc nixe0n]# lsattr -a test*
---i-------- test.conf
----a------- test.log
------------ test.txt
```

输出结果显示，test.log 只能被添加，而 test.conf 文件不允许修改。在 UNIX 系统中，如果一个用户以 root 的权限登录，文件系统的权限控制将无法对 root 用户和以 root 权限运行的进程进行任何的限制。这样对于 UNIX 类的操作系统，如果攻击者通过远程或者本地攻击获得 root 权限将可能对系统造成严重的破坏。而 ext2 文件系统可以作为最后一道防线，最大限度地减小系统被破坏的程度，并保存攻击者的行踪。ext2 属性是由 sys_open()和 sys_truncate()等系统调用检查和赋予的，不受用户识别号和其他因素的影响，在任何情况下，对具有不可修改(immutable)属性的文件的进行任何修改都会失败，不管是否是 root 用户。

但是，root 权限的用户可以通过删除 i 属性实现对文件的修改。这种防护只不过给获得 root 权限的攻击者增加了一点小麻烦罢了，系统的安全性并没有根本性的提高。

在 2.1 之前的内核版本中，存在一个安全层(securelevel)的特征。使用安全层可以解决上述问题，因为如果系统的安全层大于 0，内核将不允许对任何文件的 i 属性进行修改。这些版本的内核由 sysctl 命令的"kernel.securelevel"变量进行控制。如果在启动时，这个变量的值被设置为 1 或者更大的值，内核将不允许对具有 i 属性和 a 属性文件进行修改，除非切换到单用户状态。

但是，由于引入了更为灵活的内核能力特征(kernel capabilities)，以后的内核不再支持安全层。使用内核能力，也可以实现类似的限制。工具 lcap 用来查询和调整内核能力约束集(kernel capabilities bounding set)。在启动脚本中加入以下命令，就可以实现对具有 i 属性和 a 属性文件的保护：

```
lcap  CAP_LINUX_IMMUTABLE
lcap  CAP_SYS_RAWIO
```

第一个命令删除任何用户（包括超级用户）对 i 标志的修改能力。第二个命令删除任何用户（主要针对超级用户）对块设备的原始访问(raw access)能力，防止一些技术高超的攻击者直接修改文件系统索引节点的

immutable 域。在系统启动时，CAP_SYS_RAWIO 能力应该直接删除，这个能力是一个非常大的潜在威胁。高明的攻击者获得了超级用户权限之后，通过/dev/kmem 设备可以直接修改内核内存。通过这种方式，可以破坏系统的内核能力约束集(kernel capabilities bounding set)。如果没有任何参数，lcap 会列出内核支持的能力和目前生效的内核能力。

一旦一个内核能力被删除，就只有在系统重新启动，进入到单用户模式才能解除能力限制。

对于直接暴露在 Internet 或者位于其它危险的环境，有很多 shell 帐户或者提供 HTTP 和 FTP 等网络服务的主机，一般应该在安装配置完成后使用如下命令：chattr -R +i /bin /boot /etc /lib /sbin

```
chattr -R +i /usr/bin /usr/include /usr/lib /usr/sbin
chattr +a /var/log/messages /var/log/secure (...)
```

如果很少对帐户进行添加、变更或者删除，把/home 本身设置为 immutable 属性也不会造成什么问题。在很多情况下，整个/usr 目录树也应该具有不可改变属性。实际上，除了对/usr 目录使用 chattr -R +i /usr/命令外，还可以在/etc/fstab 文件中使用 ro 选项，使/usr 目录所在的分区以只读的方式加载。

另外，当系统日志 log 文件和 log 备份一起使用时不可变(i)和只添加(a)这两种文件属性特别有用。系统管理员将活动的 log 文件属性设置为只添加。当 log 被更新时，新产生的 log 备份文件属性设置成不可变的，而新的活动的 log 文件属性又变成了只添加。这通常只需要在 log 更新脚本中添加一些控制命令，从而使入侵者无法擦除自己的踪迹。

虽然 ext2 扩展属性能够提高系统的安全性，但是它并不适合所有的目录。如果在系统中滥用 chattr，可能造成很多问题，甚至使系统无法工作。

● /　　很显然，根分区不能有 immutable 属性。如果根分区具有 immutable 属性，系统将根本无法工作。

● /dev　在启动时，syslog 需要删除并重新建立/dev/log 套接字设备。如果对/dev/目录设置了 immutable 和 append-only 属性，就可能出现问题，除非在启动 syslogd 时使用-p 选项指定其它的套接字，例如：/var/run/syslog.sock。即使这样也还存在一些问题，syslog 客户程序需要/dev/log 套接字设备，因此需要建立一个指向真正套接字的符号链接。总而言之，为了减少麻烦，这个目录还是不要设置 immutable 和 append-only 属性。

● /tmp　有很多应用程序和系统程序需要在这个目录下建立临时文

件，因此这个目录也不能设置 immutable 和 append-only 属性。

● /var　这个目录不能设置 immutable 属性。对 append-only 属性的使用要根据实际情况。例如，为 var/log 目录下的日志文件设置了 append-only 属性，会使日志轮换(log rotate)无法进行，但不会造成太大问题，需要权衡是否使用日志轮换的利弊，以决定是否对日志文件设置 append-only 属性。

4.2 网络安全配置与连接

网络安全是一个非常重要的话题，而服务器是网络安全中最关键的环节。Linux 被认为是一个比较安全的 Internet 服务器操作系统，主要提供了两种网络系统服务存取控制机制：一种是以 ipchains 或 iptables 为具体实现的 firewall 机制，另一种是以 TCP Wrapper 为具体实现的 xinetd 机制。作为一种开放源代码操作系统，一旦 Linux 系统中发现有安全漏洞，Internet 上来自世界各地的志愿者会踊跃修补它。然而，相对于系统本身的安全漏洞，更多的安全问题是由不当的配置造成的，这可以通过适当的配置来防止。服务器上运行的服务越多，不当配置出现的机会也就越多，出现安全问题的可能性就越大。

4.2.1 一些重要的网络配置细节

（1）关闭不用的服务

linux 中控制服务的工具有 chkconfig，ntsysv 等等，其实这些工具控制的服务都是 linux 以 UNIX sysV 的风格保存的服务启动项目，即/etc/rc.d/下面的内容。以 rc3.d 为例，3 表示 init3 时要做的项目，里面的文件都是一些链接，S 开头表示启动，K 开头表示终止。

运行 ntsysv 关闭服务，只打开

crond　　可定义计划任务

network　网络

random　　生成随机数，用于 ssh 的会话对称密钥的生成

iptable　2.4 内核防火墙服务

sshd　　ssh 服务器端

syslog　　系统日志服务

xinetd　　Internet 超级进程（如果下面没有提供服务，可以关闭）

xinetd 是类似于 init 这个超级进程的一个进程，但可以完全关闭它，对于一般的服务器，基本上只需要上面 ntsysv 所列的服务就行了，其他

的都关闭。

（2）限制特定 IP 地址的访问

默认情况下，多数 Linux 系统允许所有的请求，使用 TCP_WRAPPERS 增强系统安全性可以使服务器更好地抵制外部侵入。通过修改 /etc/hosts.deny 和/etc/hosts.allow 来增加访问限制。例如：

将/etc/hosts.deny 设为“ALL：ALL@ALL”则默认拒绝所有访问，除非由 hosts.allow 文件指明允许。

然后在/etc/hosts.allow 文件中添加允许的访问：

“sshd：192.168.1.10/255.255.255.0　gate.openarch.com”表示允许 IP 地址 192.168.1.10 和主机名为 gate.openarch.com 的机器通过 SSH 连接。

配置完成后，可以使用 tcpdchk 检查：

```
# tcpdchk
```

tcpchk 是 TCP_Wrapper 配置检查工具，它检查你的 tcp wrapper 配置并报告所有发现的潜在/存在问题。

（3）阻止 ping

Linux 中，如果要忽略 icmp 包.，即使 ping 程序没有反应，安全性自然增加了。为此，可以在/etc/rc.d/rc.local 文件中增加如下一行：

```
echo 1 >/proc/sys/net/ipv4/icmp_echo_ignore_all
```

若想恢复就用

```
echo 0 >/proc/sys/net/ipv4/icmp_echo_ignore_all
```

（4）防止源路由

路由和路由协议会导致一些问题。IP 源路径路由（IP source routing），也就是 IP 包含有到达目的地址的详细路径信息，这是非常危险的，因为根据 RFC 1122 规定目的主机必须按原路径返回这样的 IP 包。如果黑客能够伪造源路由的信息包，那么就能截取返回的信息包，并且欺骗你的计算机，让它觉得正在和它交换信息的是可以信任的主机。

用下面的脚本在服务器上禁止 IP 源路径路由：

```
for f in /proc/sys/net/ipv4/conf/*/accept_source_route;  do
echo 0 > $f
done
```

把上面的脚本加到"/etc/rc.d/rc.local"文件中，保证在系统重新启动之后仍然有效。注：上面的命令将禁止所有的网络界面（lo、ethN、pppN，等等）的源路径路由包。

4.2.2 网络防火墙设置

在基本的安全设置做完后，接下来一个比较主要的安全就是防火墙，这非常重要。如果设置没有做得很好，或软件还存在一些漏洞，但是防火墙设置得好的话，基本上可以弥补这些问题。由于项目计划中，基于 2.4 内核 netfilter 框架，另行开发了分布式防火墙套件，因此这里仅仅给出一个基本的单服务器防火墙脚本，里面不包括端口转发，伪装等网关功能内容。

```
#!/bin/bash
#启用转发功能
echo 1 > /proc/sys/net/ipv4/ip_forward
#显示开始信息
echo "Start FireWall for this server..."
#清空目前的规则
iptables -F
#编辑默认策略为不能通过数据接入链
iptables -P INPUT DROP
#转发链默认 drop
iptables -P FORWARD DROP
#数据输出链默认 drop
iptables -P OUTPUT DROP
#输出链允许源地址是 xxx.xxx.xxx.xxx 的数据输出，
# 也可以指定网卡，如： –i eth0
iptables -A OUTPUT -s xxx.xxx.xxx.xxx -j ACCEPT
#接入链允许端口为 15818 的源地址为 xxx.xxx.xxx.xxx 的数据通过
iptables -A INPUT -p tcp -d xxx.xxx.xxx.xxx --dport 15818 -j ACCEPT
#如果需要添加端口就在下面修改，xxx 处添加端口号
#iptables -A INPUT -p tcp -d xxx.xxx.xxx.xxx --dport xxx -j ACCEPT
#允许所有 udp 包通过
iptables -A INPUT -d xxx.xxx.xxx.xxx -p udp -j ACCEPT
#限制 ping 包每一秒钟一个，10 个后开始
iptables -A INPUT -p icmp -d xxx.xxx.xxx.xxx -m limit --limit 1/s
--limit-burst 10 -j ACCEPT
#限制 IP 碎片，每秒钟只允许 100 个碎片，防止 DoS 攻击
```

iptables -A INPUT -f -m limit --limit 100/s --limit-burst 100 -j ACCEPT

4.2.3 远程连接的安全性实现

在 Linux 系统中，有一系列 r 字头的公用程序（rsh，rlogin 等），可以使用户在不需要提供密码的情况下执行远程操作，因此，在提供方便的同时，也带来了潜在的安全问题，它们常被黑客用做入侵的武器，非常危险。还有如 telnet 等网络程序，用明文（plain text）传送口令和秘密的信息，所以可利用任何连接到网络上的计算机监听这些程序和服务器之间的通信并获取口令和秘密信息。现在，telnet 程序对于日常的管理工作是必不可少的，但是它又是不安全的。因此使用 OpenSSH 来替代 telnet、rlogin 或 rcp 等这些过时的、不安全的远程登录程序。

OpenSSH 是 SSH（Secure SHell）协议的免费开源实现，所有使用 OpenSSH 工具的通讯，包括口令，都会被加密。RedHat Linux 自含的 openssh，主要有三个 rpm 包：

openssh-2.1.p4-1.rpm：包含 openssh 的核心文件

openssh-server-2.1.1p4-1.rpm：包含 openssh 的服务器程序

openssh-clients-2.1.1p4-1.rpm：包含 openssh 的客户端程序

（1）使用基于传统口令认证的 openssh

缺省情况下，ssh 仍然使用传统的口令验证，在使用这种认证方式时，不需要进行任何配置。就可以使用帐号和口令登录到远程主机。所有传输的数据都会被加密，但是不能保证正在连接的服务器就是你想连接的服务器。可能会有别的服务器在冒充真正的服务器，也就是受到“中间人”这种方式的攻击。

使用以下语法登录服务器：

ssh 用户名@主机地址。

（2）配置并使用基于密匙认证的 openssh

openssh 采用 RSA/DSA 密钥认证系统，基于密匙的安全验证（ssh2）需要依靠密匙，即用户必须为自己创建一对密匙，并把公用密匙放在需要访问的服务器上。如果要连接到 SSH 服务器上，客户端软件就会向服务器发出请求，请求用密匙进行安全验证。服务器收到请求之后，先在该服务器的 home 目录下寻找公用密匙，然后把它和发送过来的公用密匙进行比较。如果两个密匙一致，服务器就用公用密匙加密“质询”（challenge），并把它发送给客户端软件。客户端软件收到“质询”后，就可以用私人密匙解密再把它发送给服务器。使用这种方式，用户必须知道自己密匙的口令。

与基于传统口令认证相比，这种方式不需要在网络上传送口令，不仅加密所有传送的数据，而且阻止了“中间人”攻击方式。

下面简要说明基于密匙认证的 openssh（基于 ssh2）配置脚本：（假定在远程服务器和本地工作站上均有一个管理员帐号：admin）

● 配置远程服务器：

为获得最大化的安全性，修改服务器上/etc/ssh/sshd_conf 脚本文件：

PasswordAuthentication yes 改为 PasswordAuthentication no

即只能使用密匙认证的 openssh，禁止使用口令认证。

● 在客户端生成密匙：

用 admin 登录本地工作站，然后执行：

$ssh-keygen -d

系统将显示：

Generating DSA parameter and key.

Enter file in which to save the key (/home/admin/.ssh/id_dsa):

请输入密匙文件的保存路径，回车使用缺省路径。

如果还没有/home/admin/.ssh 目录，系统将显示：

Created directory '/home/admin/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase:

Your identification has been saved in /home/admin/.ssh/id_dsa.

Your public key has been saved in /home/admin/.ssh/id_dsa.pub.

The key fingerprint is:

D8:20:0f:01:5d:8f:6f:de:b9:d5:ce:28:d8:ca:45:59 admin@admin.example.com

这里的密码是对生成的私匙文件（/home/admin/.ssh/id_dsa）的保护口令。

公匙文件是/home/admin/.ssh/id_dsa.pub

● 发布公匙：

通过 ftp 将公匙文件/home/admin/.ssh/id_dsa.pub 复制到远程服务器上的以下目录：

/home/admin/.ssh

然后，将 id_dsa.pub 重命名为 authorized_keys2:

mv id_dsa.pub authorized_keys2

再用 chmod 修改 authorized_keys2 的属性：

chmod 644 authorized_keys2

（注：如果 authorized_keys2 文件的权限不正确，会导致 ssh 连接失败）

● 启动 sshd 服务：

在远程服务器上使用 root 权限执行：

/etc/rc.d/init.d/sshd start

● 连接远程服务器：

用 admin 帐号登录本地工作站，并使用 ssh 连接远程服务器：

ssh -l admin www.example.com

Enter passphrase for DSA key '/home/admin/.ssh/id_dsa'：

输入原先设置的密码后，就可以登录到 www.example.com 了。

4.3 控制台扩展与安全配置

通过控制台管理 Linux 服务器，有很多种不同的办法：键盘+显示器、通过网络登录、基于 WEB 的工具、X 系统下的图形工具等等，这些方法用在普通的服务器上是没有任何问题的，但是要管理一台 24×7 模式的服务器，确保在任何情况下都能直接快捷地登录到服务器，就不是那么容易了。

对比以上说到的几种方式：

- X 系统下的图形工具。界面友好，而且 X 系统也可以通过网络扩展到远程，但网络毕竟不是这么可靠，如果服务器的网络瘫痪了，而又正好需要远程解决网络的问题，就无能为力。

- 基于 Web 的工具。这种方式的界面也是相当友好，如 webmin、VNC 等远程管理软件包，但是由于安全方面的问题，这些工具是不会安装在真正重要的服务器上的，另外这种方式也受限于网络；

- 通过网络登录。这是常用的办法，一般服务器会对信任的远程站点开放 SSH 服务，以便维护人员可以远程登录进行日常维护，这是很安全的，但它还是基于网络的，当然也受限于网络；

- 键盘+显示器。这是最保险的办法，但是也有缺陷，很多服务器都不配备显示器和键盘，就算有，也只是安装系统时使用，一旦投入生产运行就不再使用显示器和键盘了。

事实上，对于 UNIX/LINUX 系统的服务器，还可以通过 COM 口即串行口来扩展系统的控制台，作为另外的途径管理服务器。

4.3.1 串口终端控制台的实现

控制台是系统输出管理信息的字符输出设备，这些管理信息是由内核产生，比如系统日志、告警信息等。Linux 下默认的控制台是键盘和显示器，而一些老式的 SUN、HP、IBM 服务器则使用串行口接 Text Terminal（字符终端,如著名的 DEC 的 VT100）作为控制台。

字符终端在大型机时代用得比较多，一个大型机通常联接许多终端，每个终端没有处理能力，只是简单地用键盘输入和从屏幕输出结果，处理过程都是由主机完成。字符终端屏幕多为黑白字符型的，不具备图形显示功能。这里所说的 Text Terminal 和大型机的终端是一样的，只不过是通过应用程序模拟而已，所以称为虚拟终端，如 WINDOWS 的超级终端就是常用的一个虚拟终端程序，在管理工作站上打开超级终端（使用正确的速率），就相当于打开一个实际的硬件终端设备。

这种通过串行口实现控制台功能有着广泛的应用，如 CISCO 路由器，都是通过这种方式进行设备的管理，小小的一条串口线就相当于一台 PC 的显示器加键盘，这无疑为路由器省下了许多不必要的配件，而事实上部分 CISCO 的设备就是一台 PC，如思科 PIX515 防火墙，只要插上一块 PCI 显卡，接上显示器，按下电源开关，就可以看到一台 BX 主板、Pentium II350MHZ 处理器、128M 内存电脑的启动信息，和一台普通电脑启动是没有任何区别的。当 PIX515 完成自检后，控制台就转到串口上了。

在本项目软件开发中，边界防火墙实现的一个目标就是：利用管理工作站的虚拟终端程序模拟一个字符终端设备，通过串口对串口的形式将 Linux 控制台重定向到该虚拟终端，以实现从虚拟终端控制 Linux 服务器。

4.3.1.1 连线定义与测试

两台电脑通过 COM 口互联起来需要一种“NULL MODEM CABLE”，也就是"交叉 MODEM 电缆线"，和网线的交叉线意思相近，这是用来“背靠背”联接两个 DTE 设备的，两头都是 9 针 D 型母插头，分别插入管理工作站和服务器的 COM 口中，为了方便标识都接入第一个 COM 口。这种电缆两端 9 针插头的线序定义如下：

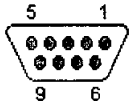


图 4.2 9 针母插头俯视图

9针母头管脚定义：

管脚号	管脚中英文名	
1	Data Carrier Detect	数据载波检测
2	Receive Data	接收数据
3	Transmit Data	发送数据
4	Data Terminal Ready	数据终端就绪
5	System Ground	信号地
6	Data Set Ready	数据设备就绪
7	Request to Send	请求发送
8	Clear to Send	清除发送

表 4.1　9针母头管脚定义

两头均为9针母插头时，线路连接为：

	D-Sub 9	D-Sub 9	
Carrier Detect	1	7+8	Request to Send+Clear to Send
Receive Data	2	3	Transmit Data
Transmit Data	3	2	Receive Data
Data Terminal Ready	4	6	Data Set Ready
System Ground	5	5	System Ground
Data Set Ready	6	4	Data Terminal Ready
Request to Send+Clear to Send	7+8	1	Carrier Detect
RI	9	9	RI

表 4.2 9—9针线路连接

管理工作站和服务器接上串口线后，可以用一种简单的方法来测试线的连通性。在管理工作站上用超级终端新建一个联接，速率9600，8数据位，1停止位，无奇偶校验位，无硬件"Flow control"；在服务器上执行命

令：

echo '1'>/dev/ttyS0(需要 ROOT 权限)

然后在管理工作站超级终端里观察是否能收到数字 1，如果能收到就表明联接无问题。如果不能收到则要在服务器上检查一下各信号引脚是否正确，使用命令"statserial /dev/ttyS0"可以查看当前串行口的状态（在 CABLE 对端不接管理工作站时或者线序错误时 DSR 状态为 0），联接状态如下：

Signal Name	Pin（9）	Direction (computer)	Status	Full Name
TxD	3	out	-	Transmit Data
RxD	2	in	-	Receive Data
RTS	7	out	1	Request To Send
CTS	8	in	0	Clear To Send
DSR	6	in	1	Data Set Ready
GND	5	-	-	Signal Ground
DCD	1	in	0	Data Carrier Detect
DTR	4	out	1	Data Terminal Ready
RI	9	in	0	Ring Indicator

表 4.3 串口联接状态

4.3.1.2 串口终端系统设置

在连接好串口线后，还必须在服务器端进行有效设置与脚本程序的编写并执行，串口控制台才最终可以使用。

（1）重定向 GRUB

如果服务器上有几个系统（如 FreeBSD），或者有新的内核需要进行测试，可能希望通过 COM 口来选择系统，只需要将 GRUB 重定向到 COM 口。在 GRUB 的 manual 页[4]可以看到其支持 seriral 终端，在 GRUB 配置

脚本第一条 Title 的前面加入两行如下：

```
serial --unit=0 --speed=38400
terminal serial console
```

这样便能从接在 COM 口上的终端中看到 GRUB 的提示信息，通过上、下键选择光带就可以正确地进入各个系统了。

（2）重定向控制台（CONSOLE）

为了能够控制 LINUX 服务器启动过程，我需要传递一些参数到 LINUX 内核中，从 serial-console[2] 中可以看到只需要将 console=ttyS0,9600 传递到内核中就可以实现 CONSOLE 重定向到串口。当然 9600 的速度太慢，事实上可以使用 38400 的速率，这样显示的速度就和启动时显示器上显示的速度相近了。在服务器的/boot/grub/menu.lst 文件中 kernel 语句作如下修改：

```
kernel /vmlinuz-2.4.20-8 ro root=LABEL=/ console=tty0 console=ttyS0,38400
```

重新启动 LINUX 服务器，打开超级终端，就能看到平时在显示器上才看得到的启动信息，当然也可以按"i"键进入交互式的启动模式。但启动完后系统无法登录，这需要控制台登录软件的协助。

（3）开启 ttyS0 登录

LINUX 启动时的信息都是由 KERNEL 显示的，缺省情况下启动完毕后 init 再运行 mingetty 提供 6 个虚拟终端来登录系统，这可以在/etc/inittab 文件中清楚地看到。其实 login 和内核是没有关系的，所以如果只是在内核中加入参数使其能从 COM 口交互式地操作并不代表能从超级终端登录系统，需要另外运行一个提供从串口登录能力的程序，这个程序就是 agetty，它是属于 util-linux 软件包中的一员，在服务器的/etc/inittab 文件中加入一行如下：

```
S0:2345:respawn:/sbin/agetty -L 38400 ttyS0
```

这样就能从超级终端登录系统了。

（4）需注意的两点

- COM 口的传输距离有限制，普通的线缆只有 15 米的有效距离。
- 默认不能使用 root 从 ttyS0 登录，但是，可以把该终端加入到文件/etc/securetty 中，即终端 ttyS0 是安全的终端，就能使用 root 登录系统了。

4.3.2 控制台安全配置与性能优化

控制台是系统与用户之间交互的中介，稳定、安全和方便的控制台对系统的作用及其重要。下面对优化与配置控制台的一些细节进行详细讨论：

（1）使控制台以颜色区分不同文件

在/etc/bashrc 脚本中添加下面的三行命令：

```
LS_COLORS=’采用系统默认颜色配置’
export LS_COLORS
alias ls=’ls --color=auto -NF’
```

（2）禁止普通用户使用控制台程序

应该禁止服务器上的普通用户对关闭（shutdown）、重启（reboot）、挂起（halt）等控制台级别程序的访问，可以编辑如下脚本来实现：

```
[root@uestc/]# rm -f /etc/security/console.apps/servicename
```

这里 servicename 是要禁止的控制台程序名。例如：

```
rm -f /etc/security/console.apps/halt
rm -f /etc/security/console.apps/poweroff
rm -f /etc/security/console.apps/reboot
rm -f /etc/security/console.apps/shutdown
rm -f /etc/security/console.apps/xserver (如果删除，只有 root 可以启动 X).
```

（3）用户超时注销并删除命令记录

如果用户离开时忘记注销账户，则可能给系统安全带来隐患。可修改/etc/profile 文件，保证账户在一段时间没有操作后，自动从系统注销。编辑文件/etc/profile，在“HISTFILESIZE=”行的下一行增加如下一行：

```
TMOUT=600
```

则所有用户将在10分钟无操作后自动注销。

注销时删除命令记录

编辑/etc/skel/.bash_logout文件，增加如下行：

```
rm -f $HOME/.bash_history
```

这样，系统中的所有用户在注销时都会删除其命令记录。

如果只需要针对某个特定用户，如root用户进行设置，则可只在该用户的主目录下修改/$HOME/.bash_history文件，增加相同的一行。

（4）不接受从不同控制台的根用户登录

“/etc/securetty”文件指定“root”用户被允许从哪个TTY设备登

录。编辑“/etc/securetty”文件，在不需要的tty前面加“#”，禁用这些设备。由于本项目增加了串口控制台，因此在“/etc/securetty”文件中增加如下一行：

ttyS0

4.3.3 控制台自动登录

启动级别为 3 时自动登录的实现涉及两个软件包：mingetty-1.00-3.src.rpm 软件包及 util-linux-2.11r-10.src.rpm 软件包。

（1）mingetty-1.00-3.src.rpm软件包

对于启动级别为3的自动登录的实现，仍然需要考察/etc/inittab脚本，3:12345:respawn:/sbin/mingetty tty3

因此，如果想在启动级别 3 的情况下实现自动登录，必须要了解 mingetty 的功能，并修改 mingetty 的代码。缺省情况下，代码会安装在/usr/src/redhat/下， mingetty.c 源文件约有五百行代码，主要实现如下功能：

打开指定的 tty（由参数指定）；

提示用户登录（login:）；

获得登录用户名；

把用户登录名作为参数，调用/bin/login。

我们所关心的部分实质上只有以下三行：

```
... ...
 while ((logname = get_logname ()) == 0); //mingetty.c 文件 438
行
 execl (_PATH_LOGIN, _PATH_LOGIN, "--", logname, NULL);
 error ("%s: can't exec " _PATH_LOGIN ": %s", tty,
sys_errlist[errno]);
... ...
```

第一行的功能是输出 login 提示，并获得用户输入的登录用户名，登录用户名由 logname 返回。因此，可作如下修改

```
... ...
// while ((logname = get_logname ()) == 0); //注释掉本行，不再提
示 login:
logname = "root"; //添加本行代码
execl (_PATH_LOGIN, _PATH_LOGIN, "--", logname, NULL);
```

```
error ("%s: can't exec " _PATH_LOGIN ": %s", tty,
sys_errlist[errno]);
... ...
```

（注：这里假定用户以超级用户身份登录。）

第二行以用户登录名为参数，调用/bin/login 程序，进一步实现登录。因此，要想实现自动登录，还应该了解/bin/login 的功能，必要时还应修改其源代码。

第三行为出错处理。

（2）util-linux-2.11r-10.src.rpm 软件包

/bin/login 包含在 util-linux-2.11r-10.src.rpm 软件包中，login 可执行文件由几个源文件编译而成，我们最关心的是 login.c 源文件（大约 1500 行的代码）。下面简要分析 login 要实现的功能，并对相应部分作必要的修改。

Login 程序主要可以分为以下几个部分：

●Login 首先检查登录者是否为超级用户，如果不是超级用户，并且存在/etc/nologin 文件，则输出该文件内容，并中止登录过程；主要由 checknologin（）实现；

●如果登录用户是超级用户，那么 login 必须在/etc/securetty/中指定的 tty 列表中实现登录，否则将导致登录失败。同样可以不指定/etc/securetty 文件，此时，超级用户可以在任何 tty 上登录。

●经过前两步测试后，login 接下来将提示输入登录密码（由 getpass（）调用完成，有兴趣的读者可参考其手册页面），并进行验证，如果密码不对，则提示重新登录。

●顺利经过密码验证后，login 还将检查是否存在.hushlogin 文件，如果该文件存在，则执行一次"quiet"登录（所谓的 quiet 登录指的是，登录时不再提示邮件 mail，不再显示最后一次登录时间，不输出任何消息。启动级别为 3 时，正常情况下输出这些信息）

●login 接下来设置登录 tty 的用户 ID 和组 ID，并设置相应的环境变量，包括 HOME、PATH、SHELL、TERM、LOGNAME 等。对于普通用户来说，PATH 缺省被设置成/usr/local/bin: /bin/usr/bin:；对于超级用户来说，PATH 被设置成/sbin: /bin: /usr/sbin: /usr/bin:

●login 的最后一步是为用户启动 shell。如果在/etc/passwd 中没有为用户指定 shell，那么将使用/bin/sh，如果在/etc/passwd 中没有给出当前工作目录，则使用"/"。

至此，一个完整的登录过程就结束了。

从以上对 login 源程序分析过程中可发现，如果要实现自动登录，应该在第三步做文章，设法绕过提示输入密码以及对密码进行验证这一过程。实际上很简单，login 源程序对是否要求输入密码设置了一个开关控制 passwd_req，缺省情况下，其值为 1（passwd_req = 1），即要求输入密码进行身份验证。把该行代码改为（passwd_req = 0）后，问题就解决了。即对源文件作如下修改即可：

```
... ...
fflag = hflag = pflag = 0; //login.c 文件 402 行
//passwd_req = 1 //缺省时，要求进行密码验证，注释掉本行
passwd_req = 0 //添加本行
... ...
```

修改后，可以直接使用 util-linux-2.11r-10.src.rpm 提供的 Makefile 进行重新编译，也可以自行对其编译：

```
gcc -o login login.c setproctitle.c checktty.c xstrncpy.c -Wall
-lcrypt
```

在得到经修改的 mingetty 及 login 后，拷贝 mingetty 到/sbin/目录，拷贝 login 到/bin 目录，并将/etc/inittab 中的启动级别设置为 3，再重新引导系统即可实现控制台自动登录。

另外，在实现了启动级别 3 时的自动登录后，自动运行应用程序也非常简单，把应用程序添加在/etc/rc.d/rc.local 脚本中既可。事实上，在系统初始化到 mingetty 及 login 这一阶段，内核实际上已经完成了引导过程，到了系统初始化的最高阶段，与内核已没太大关系。此时，主要是/sbin/init 根据/etc/inittab 的内容而执行不同的启动级别。

第五章 BLP 安全系统模型的研究

5.1 安全模型的作用和特点

一个操作系统是安全的，系它满足某一给定的安全策略。同样，进行安全操作系统的设计和开发时，也要围绕一个给定的安全策略进行，安全策略是指有关管理、保护和发布敏感法律信息的法律、规定和实施细则，例如，可以将安全策略定为：系统中的用户和信息被划分为不同的层次，一些级别比另一些级别高，主体能读访问客体，当且仅当主体的级别高于或等于客体的级别；主体能写访问客体，当且仅当主体的级别低于或等于客体的级别。

安全模型就是对安全策略所表达的安全需求的简单，抽象和无歧义的描述[4]，它为安全策略与其实现机制的关联提供了一种框架，安全模型描述了某个安全策略需要用哪种机制来满足；而模型的实现则描述了如何把特定的机制应用于系统中，从而实现某一特别的安全策略所需的安全保护。

开发安全系统，一般首先应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。

安全模型有以下特点：

- 精确的、无歧义的；
- 简单的、抽象的、容易理解；
- 一般性的，只涉及安全性质，不过多牵扯系统的功能或其实现细节；
- 安全策略的明显表现。

安全模型一般分为两种：形式化的安全模型和非形式化的安全模型。非形式化安全模型仅模拟系统的安全功能；形式化安全模型则使用数学模型，精确地描述系统的安全功能。

5.2 重要的相关概念[6]

- 主体：系统中能够发起行为的实体，如进程。

• 客体：系统中被动的主体行为承担者。对一个客体的访问隐含着对其包含信息的访问。客体的实体有：记录、程序块、页面、段、文件、目录、目录树和程序，还有位、字节、字、字段、处理器、视频显示器、键盘、时钟、打印机和网络节点等。

• 安全策略：有关管理、保护和发布敏感信息的法律、规定和实施细节。简单的说，就是用户对安全的要求的描述。

• 安全模型：用形式化的方法来描述如何实现系统的安全要求（机密性，完整性和可用性）。

• 安全内核：控制对系统资源的访问而实现基本安全规则的计算机系统的中心部分。

• 客体重用：对曾经包含一个或几个客体的存储介质（如内存、盘扇面）重新分配和重用。为了安全的重分配、重用，介质不得包含重分配前的残留数据。

• 标识与鉴别：用于保证只有合法用户才能进入系统，进而访问系统中的资源。

• 访问控制：限制已授权的用户、程序、进程或计算机网络中其他系统访问本系统资源的过程。

• 自主访问控制：用来决定一个用户是否有权限访问此客体的一种访问约束机制，该客体的拥有者可以按照自己的意愿指定系统中的其他用户对此客体的访问权。

• 强制访问控制：用于将系统中的信息分密级和分类进行管理，保证每个用户只能够访问那些被标明能够由他访问的信息的一种访问约束机制。

• 最小特权原理：系统中每一个主体只拥有和其操作相符的所要求的必需的最小的特权级。

• 审计：一个系统的审计就是对系统中有关安全的活动进行记录、检查及审核。

5.3 BLP 安全模型

5.3.1 简介

Bell-LaPadula 模型（简称 BLP 模型）是 Bell 和 LaPadula 于 1973 年创立的一种模拟军事安全策略的计算机操作模型[6]，它是最早、也是最常用的一种计算机多级安全模型. 提出模型的最初目的是在操作系统环境

下提供对于信息的安全保护，一定程度上，该模型可以看作是存取矩阵模型的扩展，保密要求是用一组系统工作时必须满足的规则集合来表述的，对于数据的安全保护是基于强制存取控制策略的。

在 BLP 模型中将主体定义为能够发起行为的实体，如进程；将客体定义为被动的主体行为承担者，如数据、文件等；将主体对客体的访问分为 r (只读)，w (读写)，a (添加)，e (执行)和 c (控制)等几种访问模式，其中 c (控制)是指该主体用来授予或撤销另一主体对某一客体的访问权限的能力．BLP 模型的安全策略包括两部分：自主安全策略和强制安全策略．自主安全策略使用一个访问矩阵表示：访问矩阵第 i 行第 j 列的元素 M ij 表示主体 S i 对客体 O j 的所有允许的访问模式，主体只能按照在访问矩阵中被授予的对客体的访问权限对客体进行相应的访问．强制安全策略包括简单安全性和附加的一些特性，系统对所有的主体和客体都分配一个访问类属性，包括主体和客体的密级和范畴，系统通过比较主体与客体的访问类属性控制主体对客体的访问。

BLP 模型是一个状态机模型，它形式化地定义了系统、系统状态以及状态间的转换规则；定义了安全概念；制定了一组安全特性，以此对系统状态和状态转换规则进行限制和约束，使得对于一个系统，如果它的初始状态是安全的，并且所经过的一系列规则转换都保持安全，那么可以证明该系统是安全的。

5.3.2 模型实体概念

BLP 模型将系统中元素按其特性区分为主体（Subject）与客体（Object)。主体是系统的主动元素，能执行一系列的动作，主要是指进程。客体是系统中包含信息的被动元素，在操作系统环境下，客体可以是文件、内存空间、程序等。

模型是基于系统元素密级的，密级用安全级别（Security Level）表示。每一安全级别有两个组成部分:“密级(Classification)”与“范围(categories)的集合”，记为 L=（C，S)。密级是如下四元素集合中的任一元素：{绝密(Top Secret)，机密(Secret)，秘密(Confidential)，公开(Unclassification)。此集合是全序的，即：

绝密 > 机密 > 秘密 > 公开

范围的集合是系统中非分层元素集合的一个子集。这一集合的元素依赖于所考虑的环境和应用领域。例如，范围的集合可以是军队中的潜艇部队，导弹部队，航空部队等；也可以是某企业中的人事部门，生产部门，

销售部门等。

如上所述，可以给出以下安全级别的实例：（绝密，潜艇部队），（秘密，导弹部队），（机密，航空部队），（公开，人事部门），（机密，生产部门），（绝密，销售部门）。

安全级别的集合形成一个满足偏序关系的格，此偏序关系称为支配（dominate）（≥）关系。对支配关系可以给出如下形式化的定义：

设安全级别 L1＝（C1，S1），L2＝（C2，S2）。

L1 支配 L2 成立，当且仅当：1）C1≥C2，2）S1⊇S2。记为 L1≥L2。

类似的,还可以给出下面三个定义：

- L1 > L2，当且仅当：1）C1 > C2，2）S1⊃S2。
- L1 < L2，当且仅当：1）C1 < C2，2）S1⊂S2。
- L1≤L2，当且仅当：1）C1≤C2，2）S1⊆S2。

如果对于给定的安全级别 L1 和 L2，L1≥L2 与 L1≤L2 均不成立，则称 L1 与 L2 是不可比的（incomparable）。

模型对系统中的每个用户分配一个安全级别，称为允许安全级（Clearance）。分配给用户的允许安全级反映了对用户不将敏感信息泄漏给不持有相应允许安全级用户的置信度。用户能以受允许安全级支配的任意安全级别向系统注册，用户激活的进程将被授予此注册安全级别。

模型对系统中每个客体也分配一个安全级别，客体的安全级别反映了存储在客体内信息的敏感度，也反映来哦未经授权向不允许存取该信息的用户泄露这些信息造成的潜在损害度。

模型考虑如下主体对客体可执行的存取方式：

- 只读（Read-Only）：读包含在客体中的信息，通常又称为“读”。
- 添加（Append）：向客体中添加信息，且不读客体中信息。
- 执行（Execute）：执行一个客体（程序）。
- 读写（Read-Write）：向客体中写信息，且允许读客体中信息，通常又称为“写”。

模型支持对于客体的分散授权管理，这种授权管理是基于隶属关系的，即客体的建立者是客体的属主，属主拥有对此客体的所有存取权限，并可以将这些权限授予其他用户，或从其他用户处将这些权限收回。只有隶属关系是不能改变的，也不能转授他人。

5.3.3 系统状态与安全系统定义

BLP 模型中系统状态是系统中元素的表示形式，它由主体、客体、访

问属性、访问矩阵以及标识主体和客体的访问类属性的函数组成。状态 $v \in V$ 可由一个有序的四元组（b，M，f，H）描述，其中：

b $\subseteq$（S×O×M） 是当前存取集，由（主体，客体，存取方式）的三元组组成。表示在某个特定的状态下，哪些主体以何种访问属性访问哪些客体，其中 S 是主体集，O 为客体集，M＝{r，w，a，e}是访问属性（存取方式）集；

M 是存取矩阵。存取矩阵描述了每个主体可以何种存取方式存取每个客体。矩阵中任意一项 M（s,o）表明主体 s 被授予的对客体 o 的存取方式。

f 是一个与系统中每个主体和客体以及他们安全级别相联系的级别函数，可以形式化的记为

f：O∩S→L （L 是安全级别的集合）

对于每个客体只有一个安全级别，是在客体建立时赋予的。记为 f_o

每个主体有两个安全级别：一个是允许安全级，记为 f_s；另一个是当前安全级，记为 f_c。

当前安全级是主体实际操作时的安全级别，在运行中是可以改变的，但它必须受主体允许安全级支配。即：

对于每个主体 $s \in S$，$f_c(s) \leqslant f_s(s)$永真。

这意味着用户能以受其允许安全级支配的任意安全级别向系统注册，此安全级别即为用户进程的当前安全级，并决定了此用户所能执行的存取方式。

H 是当前客体的层次结构，是一棵带根的有向树。树中的每个结点均与系统中的客体相对应。未激活的客体或不可存取的客体不包括在此层次结构中。模型要求此层次结构满足相容规则，即树中任一结点的安全级别必须支配其父结点安全级别。

安全系统：定义规则 p 为函数 p：R×V→D×V，对规则的解释为，给定一个请求和一个状态，规则 p 决定系统产生的一个响应和下一状态。其中 R 为请求集，V 为状态集，D 为判定集{yes， no ，error }，"yes"表示请求被执行，"no"表示请求被拒绝，"error"表示错误或其它不能处理的情况。则对于一个系统，如果它的初始状态是安全的，并且所经过的一系

列规则转换 w={p1，p2，… ps}都保持安全，则可以证明该系统是安全的。

5.3.4 操作

以下给出模型提供的 8 种操作和每个操作的执行后系统状态是如何转变的[2]。

(1) Get access

此操作是按所要求的方式初始化对一个客体的存取。此操作的执行修改了当前存取集 b，向 b 中添加一个相应于此存取方式的三元组。

(2) Release access

Release 操作是“get”操作的逆操作，其功能是终止由以前的“get”开始的存取方式。此操作的执行修改了当前存取集 b,从 b 中删除相应的三元组。

(3) Give access

Give 操作是授予一个主体对一个客体的某种存取方式。此操作使主体之间传递对客体的存取权限。执行此操作修改存取矩阵 M，向 M 中相应的主体与客体的位置插入一项，表示授予相应主体对于给定客体的存取方式。仅当所插入的存取权限符合强制存取策略时，此授权操作才能执行。

(4) Rescind access

Rescind 操作是回收以前由“give”操作授予的存取方式，是“give”操作的逆操作。为执行此操作，要求回收授权的主体对操作所涉及的客体的在客体层次结构 H 中的父结点具有写权。此操作的执行修改了存取矩阵 M，删除其中被回收的授权相应的项。此操作还修改了当前存取集 b，删除了其中被回收的授权所相应的三元组，因为 rescind 操作具有强制 release 操作的副作用。

(5) Create object

BLP 模型中客体存在两种状态：一种是未激活状态，即客体存在于系统之中，但是不能存取的；另一种是激活状态，即是可存取的。这些客体被组织在客体层次结构中。Create 操作的功能是激活一个客体，使其变成可存取的，即将其加入到客体层次结构树的相应位置。这也就修改了客体层次结构树 H。

(6) Delete object

Delete 操作是 Create 操作的逆操作，是将客体从激活状态转为未激活状态。此操作的执行修改了客体层次结构树，在树 H 中删除了相应于要 delete 客体的结点以及其所有后续结点。此操作也修改了当前存取集 b，

因为删除客体的操作具有终止所有对此客体存取的副作用。

(7) Change subject security level

此操作改变主体的当前安全级别，执行此操作修改级别函数 f，将操作涉及的主体与新的当前安全级别相联系。新的安全级别必须受主体的允许安全级别支配。

(8) Change object security level

此操作修改客体的安全级别，此操作仅对未激活客体执行。执行此操作修改级别函数 f，将操作涉及客体与新的安全级别相联系。对一个客体来说，其安全级别只能升级，即：客体新的安全级别必须支配原有的安全级别，并且新的安全级别必须受改变客体安全级别的主体允许安全级别的支配，即： $f_s \geqslant f_o$。

5.3.5 规则

为了保证系统的安全，模型定义了一组必须满足的规则集合[4]。一个系统状态是安全的，当且仅当满足这些规则。系统中每个要求的操作由系统的访问监视器控制。当且仅当操作使系统状态是安全的，即操作满足模型的所有规则时，此操作才能允许执行。模型提出可信主体的概念，可信主体是指可以依靠，不会损害系统安全的主体。某些限制只是施加在不可信主体提出的存取要求上的。

模型定义如下安全规则：

（1）简单安全规则（ss 规则）

仅当一个主体的安全级别支配另一个客体的安全级别时，主体才能具有对此客体“读”或“写”的存取权限。

一个系统状态 $v=$（b，M，f，H）满足 ss 规则，当且仅当对 M 中的每个含有“读”或“写”存取方式的元素 M[s，o]，均有： $f_s(s) >= f_o(o)$

此规则的目的是防止主体读取安全级别比它的允许安全级别高的客体中的信息。它防止主体直接从不允许它存取的级别的客体中存取信息。

（2）星（ * ）规则

一个不可信主体可以对一个客体具有“添加”（append）权限，仅当此客体的安全级别支配主体的当前安全级别。

一个不可信主体可以对一个客体具有“写”（write）权限，仅当此客体的安全级别等于主体的当前安全级别。

一个不可信主体可以对一个客体具有“读”（read）权限，仅当此客体的安全级别受主体的当前安全级别的支配。

一个系统的状态 $v=$（b，M，f，H）满足星规则，当且仅当对每一主体 $s\in S'$，这里 S' 是不可信主体的集合，$S'\subseteq S$，且对于所有的客体 $o\in O$，有：

$$append\in M[s,\ o]\Rightarrow\ f_c(s)\leqslant f_o(o)$$

$$write\in M[s,\ o]\ \Rightarrow f_c(s)=f_o(o)$$

$$read\in M[s,\ o]\ \Rightarrow f_c(s)\geqslant f_o(o)$$

对于不可信主体，星规则包括了 ss 规则，即：若满足星规则，则一定满足 ss 规则。可以证明如下：

证：设 s' 是不可信主体，o 是一个客体，$m\in M[s',\ o]$ 是满足星规则的存取方式授权。

若 m=append 或 m=execute，则 m 一定满足 ss 规则，因为 ss 规则没有对其施加任何限制。

若 m=write，因 m 满足星规则，所以 $f_c(s')=f_o(o)$。又因 $f_s(s')\geqslant f_c(s')$，所以有 $f_s(s')\geqslant f_o(o)$，因此 ss 规则成立。

若 m=read，因 m 满足星规则，所以 $f_c(s')\geqslant f_o(o)$。又因 $f_s(s')\geqslant f_c(s')$，所以有 $f_s(s')\geqslant f_o(o)$，因此 ss 规则成立。

尽管这样，两个规则都是需要的，因为 ss 规则是要求对所有主体均成立，而星规则仅要求对不可信主体成立。

上述两个规则可以总结为下面两个基本规则：

1）No read-up secrecy　一个主体仅能读取其安全级别受此主体安全级别支配的客体的信息。

2）No write-down secrecy　一个主体仅能向其安全级别支配此主体安全级别的客体写信息。

这两个规则被所有采取强制存取控制策略的安全模型广泛采用。满足这两个规则即控制了系统中信息的流动，保证主体在没有所需的允许安全级别时，不能存取它不应存取的信息。

（3）稳定规则（Tranquility Principle）

没有主体能够修改激活客体的密级。

注：这个在模型最初版本中定义的规则，已经在后来的版本中被去掉。模型的当前版本允许修改激活客体的密级。控制密级改变的规则依赖于特定的应用，并依系统而异。

（4）自主安全规则（Discretionary Security Property）(ds 规则)

每个当前存取必须出现在存取矩阵中，即一个主体只能在获取了所需的授权后才能执行相应的存取。

当且仅当对所有的主体 s，客体 o，存取方式 m，有：

$$(s, o, m) \in b \Rightarrow m \in M[s, o]$$

系统状态满足自主安全规则。

以上是 BLP 模型的基本规则。仅当一个系统的状态满足上述规则时，此系统状态是安全的。仅当一个系统状态的转换是安全的，这个转换才是可允许的。

1977 年，Feiertag 等人对 BLP 模型作了新的扩充，增加了对未激活客体的可存取性和新激活客体状态限制的规则。即：

（5）未激活客体的不可存取性规则（Non-accessibility of Inactive Objects）

一个主体不能读取一个未激活客体的内容。

此规则说明没有出现在客体层次树中的未激活客体是不能通过授予“读”与“更新”权成为可存取的。此规则可形式化地写为：

系统状态 $v=(b, M, f, H)$ 满足未激活客体的不可存取性规则，当且仅当对任何主体 s，任何未激活客体 o 有：

$$(s, o, r) \in b \Rightarrow r \neq \text{“read”} \wedge r \neq \text{“write”}$$

（6）未激活客体的重写规则（Rewriting of Inactive Objects）

每个新激活客体均被赋予一个与其以前任何活动状态无关的初始状态。

已删除客体的不可存取性，即对所有已被删除的客体 $o \in O$，有：$(s, o, *) \notin b$。

5.3.6 BLP 模型分析与其局限性

BLP 模型是最典型的信息保密多级安全模型。通常用作处理多级数据和多用户的系统设计的基础。比如在一台计算机同时处理绝密级数据和秘密级数据时，要防止处理绝密级数据的程序把信息泄露给处理秘密级数据的程序。模型的出发点是维护系统的保密性，有效地防止信息泄露，并由此描述了不同访问级别的主体和客体之间的关系。如图 5.1 所示。

BLP 模型的安全策略包括强制存取控制和自主存取控制两部分。强制存取控制部分由简单安全规则和*规则组成，通过安全级强制性约束主体对客体的存取；自主存取控制通过存取控制矩阵按用户的意愿进行存取控制。

BLP 也使用了可信主体的概念，用于表示在实际系统中不受*规则制约的主体，以保证系统的正常运行和管理。

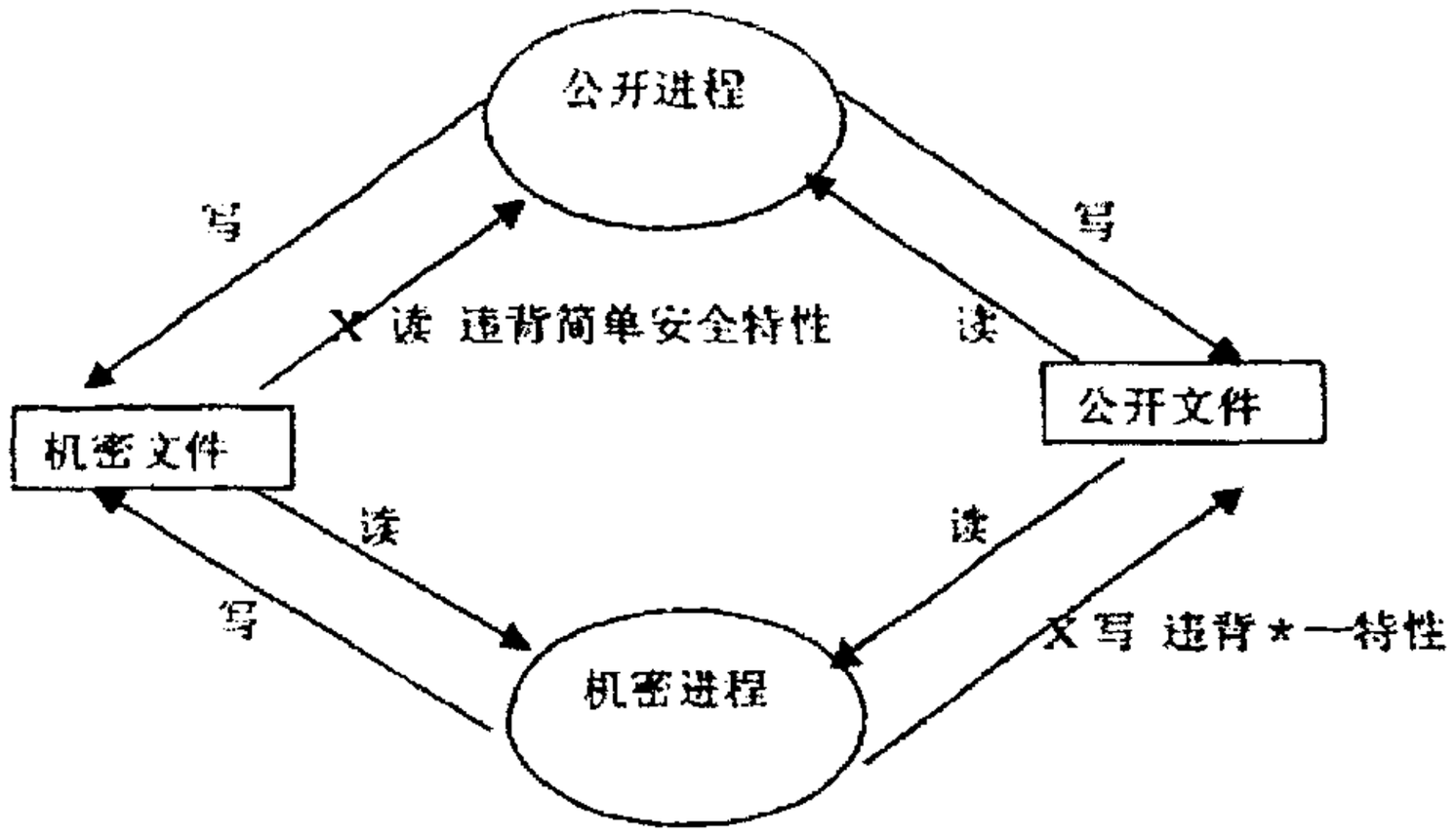


图 5.1　BLP 模型的安全策略

随着计算机技术的发展，BLP 模型已不足以描述各种各样的安全需求。应用 BLP 模型还应考虑以下几个方面的问题：

1）模型中，可信主体不受*规则约束，访问权限太大，不符合最小特权原则，应对可信主体的操作权限和应用范围进一步细化。操作系统的特权操作必须拥有相应特权的用户或进程才能完成，系统内核将对特权操作进行特权检查。

2）BLP 模型主要注重保密性控制，控制信息从低安全级传向高安全级，而缺少完整性控制，只能“从下读（read down）”，不能控制“向上写（write up）”操作，而“向上写”操作存在潜在的问题，使非法、越权篡改成为可能。但该模型的优点仍是明显有效的，因此有很多安全操作系统都以这种模型作为基本的安全模型。

第六章 BLP 模型下系统增强设计与实现

如何在 Linux 操作系统现有安全功能基础上，改进与增强系统，有多种方案可以选择。一是直接修改内核代码，安全性控制好，但 Linux 升级后程序维护工作量大；二是外包封装方式，安全性差；三是软连接技术，把安全模块软连接进入 Linux 核心，不修改 Linux 源代码，基于核心系统调用，借助模块机制扩展功能，从而提高系统安全性。本文对第三种方案作了一些尝试，希望有所借鉴。

6.1 相关技术基础

6.1.1 系统调用

系统调用是应用程序和操作系统内核之间的功能接口[8]，代表了一个从用户级别到内核级别的转换。例如，在用户级别中打开一个文件在内核级别中是通过 sys_open 这个系统调用实现的。系统调用的主要目的是使用户可以更方便的使用操作系统提供的有关设备管理、输入/输入系统、文件系统和进程控制、通信以及存储管理等方面的功能，而不必了解系统程序的内部结构和有关硬件细节，从而起到减轻用户负担、保护系统以及提高资源利用率的作用。

在 Linux 中，大部分的系统调用包含在 Linux 的 libc 库中，通过标准的 C 函数调用方法可以调用这些系统调用。目录/usr/include/sys/syscall.h 中有一个完整的系统调用列表，部分内容如下表所示：

```
#ifndef _SYS_SYSCALL_H
#define _SYS_SYSCALL_H
#define SYS_setup 0
/* 只被 init 使用,用来启动系统*/
#define SYS_exit 1
#define SYS_fork 2
#define SYS_read 3
#define SYS_write 4
#define SYS_open 5
#define SYS_close 6
……
```

```
.....
#define SYS_vm86 166
#define SYS_query_module 167
#define SYS_poll 168
#define SYS_syscall_poll SYS_poll
#endif /* <sys/syscall.h> */
```

每个系统调用都有一个预定义的编号(见上表),实际上正是使用编号来进行这些调用的。内核通过中断 0x80 控制每一个系统调用。这些系统调用的编号以及任何参数都将被放入某些寄存器中(eax 用来存放这些代表系统调用的编号)。

在内核中，系统调用的编号是一个被称为 sys_call_table[]的数组结构的索引值，这个结构把系统调用的编号映射到实际使用的函数。

6.1.1.1 Linux 系统调用机制

在 Linux 系统中，系统调用是作为一种异常类型实现的。它将执行相应的机器代码指令来产生异常信号。产生中断或异常的重要后果是系统自动将用户态切换为核心态来对它进行处理。这就是说，执行系统调用异常指令时，自动地将系统切换为核心态，并安排异常处理程序的执行。

Linux 用来实现系统调用异常的实际指令是：

Int $0x80

这一指令使用中断/异常向量号 128（即 16 进制的 80）将控制权转移给内核。为达到在使用系统调用时不必用机器指令编程，在标准的 C 语言库中为每一系统调用提供了一段短的子程序，完成机器代码的编程工作。事实上，机器代码段非常简短。它所要做的工作只是将送给系统调用的参数加载到 CPU 寄存器中，接着执行 int $0x80 指令。然后运行系统调用，系统调用的返回值将送入 CPU 的一个寄存器中，标准的库子程序取得这一返回值，并将它送回用户程序。

为使系统调用的执行成为一项简单的任务，Linux 提供了一组预处理宏指令。它们可以用在程序中。宏指令通过一定的参数，然后扩展为调用指定的系统调用的函数。

这些宏指令具有类似下面的名称格式：

_syscallN（parameters）

其中 N 是系统调用所需的参数数目，而 parameters 则用一组参数代替。这些参数使宏指令完成适合于特定的系统调用的扩展。例如，为了建立调

用 setuid（）系统调用的函数，应该使用：

_syscall1（ int， setuid， uid_t， uid ）

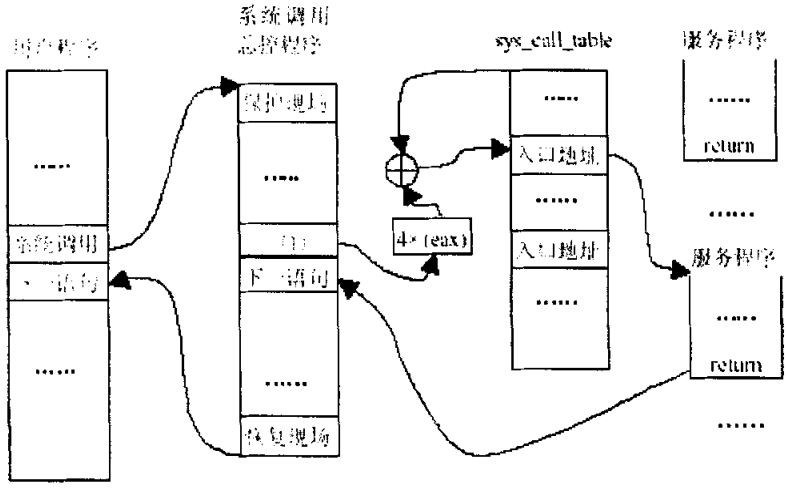
syscallN（ ）宏指令的第 1 个参数 int 说明产生的函数的返回值的类型是整型，第 2 个参数 setuid 说明产生的函数的名称。后面是系统调用所需要的每个参数。这一宏指令后面还有两个参数 uid_t 和 uid 分别用来指定参数的类型和名称。

另外，用作系统调用的参数的数据类型有一个限制，它们的容量不能超过四个字节。这是因为执行 int $0x80 指令进行系统调用时，所有的参数值都存在 32 位的 CPU 寄存器中。使用 CPU 寄存器传递参数带来的另一个限制是可以传送给系统调用的参数的数目。这个限制是最多可以传递 5 个参数。所以 Linux 一共定义了 6 个不同的_syscallN( )宏指令，从_syscall0（）、_syscall1（）直到_syscall5（）。

一旦_syscallN（）宏指令用特定系统调用的相应参数进行了扩展，得到的结果是一个与系统调用同名的函数，它可以在用户程序中执行这一系统调用。

6.1.1.2 Linux 系统调用过程

系统调用是用户程序和操作系统核心所提供功能的接口，所以系统调用的过程就是从用户程序到系统内核，然后又回到用户程序的过程。在 Linux 中，此过程大体可描述如下：



注①：此处语句为：call * SYMBOL_NAME(sys_call_table)(,%eax,4)；eax 中为系统调用号

图6.1 系统调用过程示意图

整个系统调用进入过程可表示如下：用户程序→系统调用总控程序(system_call)→各个服务程序。可见，系统调用的进入可分为“用户程序 → 系统调用总控程序”和“系统调用总控程序 → 各个服务程序”两部分。

6.1.1.3 增加系统调用

由上几节的分析可知，增加系统调用可采用以下方法：

（1）编写一个新的服务例程；该函数的名称应该是新的系统调用名称前面加上 sys_标志。假设新增加的系统调用为 mycall(int number)，在/usr/src/linux/kernel/sys.c 文件中添加源代码，如下所示：

```
asmlinkage int sys_mycall(int number)
{
return number;
}
```

作为一个最简单的例子，新加的系统调用仅仅返回一个整型值。

（2）连接新的系统调用；添加新的系统调用后，下一个任务是使 Linux 内核的其余部分知道该程序的存在。为了从已有的内核程序中增加新的函数的连接，需要编辑两个文件。第一个是入口表文件：

/usr/src/linux/arch/i386/kernel/entry.S

```
# cd /usr/src/linux/arch/i386/kernel
# vi entry.S
```

把函数的入口地址加到 sys_call_table 表中：

arch/i386/kernel/entry.S 中的最后几行源代码修改前为：

```
......
.long SYMBOL_NAME(sys_sendfile)
.long SYMBOL_NAME(sys_ni_syscall) /* streams1 */
.long SYMBOL_NAME(sys_ni_syscall) /* streams2 */
.long SYMBOL_NAME(sys_vfork) /* 190 */
rept NR_syscalls-190
.long SYMBOL_NAME(sys_ni_syscall)
.endr
```

修改后为：

```
.long SYMBOL_NAME(sys_sendfile)
.long SYMBOL_NAME(sys_ni_syscall) /* streams1 */
.long SYMBOL_NAME(sys_ni_syscall) /* streams2 */
.long SYMBOL_NAME(sys_vfork) /* 190 */
.long SYMBOL_NAME(sys_mycall) /* added by I */
.rept NR_syscalls-191
.endr
```

第二个是修改相应的头文件：

/usr/src/linux/include/asm/unistd.h

```
# cd /usr/src/linux/include/asm
# vi unistd.h
```

把增加的 sys_call_table 表项所对应的向量，在 include/asm/unistd.h 中进行必要申明，以供用户进程和其他系统进程查询或调用。

```
#define __NR_putpmsg 189
#define __NR_vfork 190
#define __NR_mycall 191 /* added by I */
// 引用结束
```

（3）由于在标准的 c 语言库中没有新系统调用的承接段，所以，在程序中，除了要#include<linux/unistd.h> ，还要申明如下：

_syscall1(int,additionSysCall,int, num)。

（4）重建新的 Linux 内核：为使新的系统调用生效，需要重建 Linux 内核。

6.1.2 可卸载内核模块机制

可卸载内核模块(Loadable Kernel Modules),即 LKMs，本来是 Linux 系统用于扩展功能的一种有效方式。使用 LKMs 的优点有:可以被系统动态加载,提高内核的灵活性，而且不需要重新编译内核。由于这些优点,他们常常被特殊的设备（如声卡）或文件系统等使用。

一旦 linux 内核模块被装入，就成为内核的一部分。和任何内核代码一样具有相同的权限和职责。

6.1.2.1 Linux 内核模块结构

在 Linux 中，每个 LKMs 至少由两个基本的函数组成:

```
int init_module(void) /*用于初始化所有的数据*/
{
...
}
void cleanup_module(void) /*用于清除数据,从而安全退出*/
{
...
}
```

加载一个模块(通常只限于 root 使用)的命令是:

```
# insmod module o
```

这个命令让系统进行如下工作:确定加载模块(在这儿是 module.o),然后调用 create_module，为 module.o 模块重新分配内存；接着再调用 get_kernel_syms，通过读取“内核符号表”找到一个没有分配的内核符号，为 module o 分配一个内核符号;最后调用 init_module 初始化 LKMs。LKMs 初始化之后，就开始执行模块函数中的内容。

确定 LKMs 是否真正加载到内存中，可以使用 lsmod 命令查看。lsmod 命令从/proc/modules 文件中读取信息，以显示当前哪些模块已经加载到内存中。Pages 域显示了内存中的信息状态（表示这个模块占据了多少页），而 Used by 域告诉我们这个模块在系统中被引用的频度。

Linux 内核是一个整体结构，而模块是插入到内核中的插件。尽管内核不是一个可安装模块，但为了方便起见，Linux 把内核也看作一个模块。模块与模块之间交互的方法是共享变量和函数，内核把各个模块中主要的变量和函数放在一个特定的区段，这些变量和函数就通称为符号。在 /proc/ksyms 文件中，显示出各种内核符号，这个文件中的每一行表示一个分配好了的内核符号。

6.1.2.2 截获 Linux 系统调用

Linux 系统的 LKMs 机制本意只是用于扩展内核功能（特别是针对一些硬件驱动程序），但是，LKMs 也可以用来捕获系统调用，甚至修改它，从而完成一些特别的任务。如下面的 LKMs 程序，当加载到内存以后，整个操作系统就会禁止任何用户在磁盘上创建目录。

```
#define MODULE
```

```
#define __KERNEL__
#include <linux/module.h>
#include <linux/kernel.h>
#include <asm/unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <asm/fcntl.h>
#include <asm/errno.h>
#include <linux/types.h>
#include <linux/dirent.h>
#include <sys/mman.h>
#include <linux/string.h>
#include <linux/fs.h>
#include <linux/malloc.h>
extern void* sys_call_table[];
/*声明使用系统本身的系统调用表*/
int (*orig_mkdir)(const char *path);
/*用于保存原始系统调用*/
int hacked_mkdir(const char *path)
{
return 0;
/*替换后的操作什么也不做*/
}
int init_module(void)
/*初始化模块*/
{
orig_mkdir=sys_call_table[SYS_mkdir];
sys_call_table[SYS_mkdir]=hacked_mkdir;
return 0;
}
void cleanup_module(void)
/*卸载模块*/
{
sys_call_table[SYS_mkdir]=orig_mkdir;
```

```
/*恢复原始 mkdir 系统调用*/
}
```

编译并启动这个模块，然后尝试新建一个目录，就会发现不能成功。由于返回值是 0（代表一切正常）因此得不到任何出错信息。在移去模块之后，才又可以新建目录。正如所看到的，只需要改变 sys_call_table 中相对应的入口就可以截获系统调用。

出此，可以得到截获系统调用的通常步骤如下：

（1）找到需要的系统调用在 sys_call_table[] 中的入口（include/sys/syscall.h）

（2）保存 sys_call_table[x]的旧入口指针。（x 代表想要截获的系统调用索引）

（3）将新定义的函数指针存入 sys_call_table[x]

保存旧的系统调用指针是十分有用的，因为在替换的新调用中有可能会用到旧调用完成的功能。事实上，通过 LKMs 截获并改进系统调用函数，是 Linux 系统增强的重要方法。

6.2 BLP 模型存取控制在 Linux 系统的实现

针对 Linux 系统的特点，对其内核和应用程序进行面向安全策略的分析，利用已有的安全机制，结合 BLP 安全模型，采用改进/增强法来实现模型，即在系统调用中实施安全强制存取控制机制（MAC）和自主存取控制机制（DAC），再辅以审计机制、最小特权管理机制等，从基础上保证操作系统的安全。

6.2.1 BLP 模型实体在 Linux 的映射

Linux 系统中，进程是唯一的主体，它可以在用户登录时创建，被系统初启时创建或被其他进程创建。一个进程被赋予一个唯一的进程标识符（pid）、用户标识符（uid）和用户组标识符（gid）。每个进程还被赋予相应的安全级标识，用于强制存取控制检查。

客体包括文件、目录、共享内存、消息、信号量、流、管道、进程。

存取控制矩阵 M 是由 Linux 的 9bit 位保护模式（owner/group/other）和 ACL（Acess Control List）共同组成

在建立了明确定义的 BLP 安全模型后，通过修改存取控制规则，使得只有安全级匹配时，主体才可以修改客体，并在信息加密机制的共同控制下，实现阻止非授权用户修改或破坏敏感信息的功能。

● 自主存取控制机制（DAC），通过 ACL 表和 9bit 位相结合的方式，实现按用户意愿说明其资源允许系统中哪些用户以何种权限进行共享。

● 强制存取控制机制（MAC），对系统中的每个进程、每个文件、每个 IPC 客体赋予相应的安全级，当进程访问客体时，根据进程的安全标识和访问方式，比较进程的安全级和文件的安全级，确定是否允许进程对文件的访问。

● 最小特权管理机制，将原 Linux 系统的超级用户特权划分为一组细粒度的特权，分别授予不同的系统管理员，使其只具有完成其任务所需的特权，任何一个用户都不能独自操纵整个系统。

● 审计，分别在系统级、应用级、网络级实施了审计机制，包括身份鉴别机制的使用，将客体引入用户地址空间（打开文件、初始化程序），删除客体，系统特权用户的操作等。

由此可得到基于 BLP 模型安全增强后 Linux 系统总体结构如 图 6.2 所示：

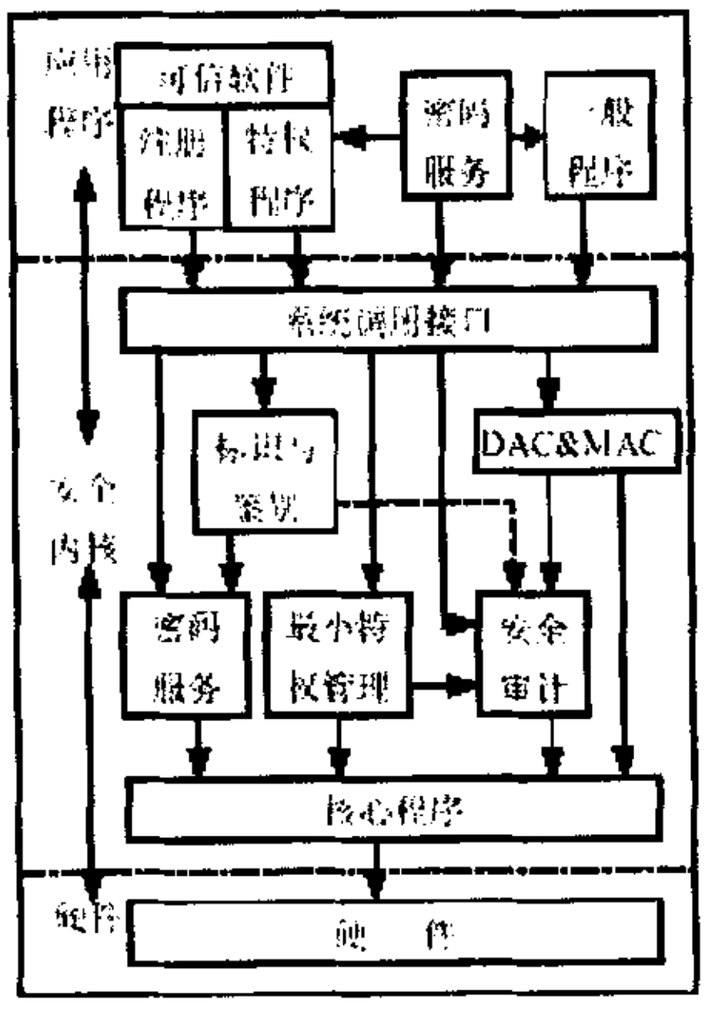


图 6.2 安全增强后 Linux 系统总体结构

从图 6.2 可以看出，安全增强的实质就是增强内核级安全，其关键安全机制，是由强制存取控制（MAC），自主存取控制（DAC），特权存取控制（PAC），安全审计等组成的，这些安全机制都遵从 BLP 模型的安全特性与主要结论。而系统调用是应用程序和系统内核之间的功能接口，代表了一个从用户级别到内核级别的转换。因此，在 Linux 系统中实现 BLP 模

型的关键安全机制，可从截获、修改与新增系统调用入手。

6.2.2 借助 LKMs 实现 BLP 模型存取控制机制

对照 BLP 安全模型，结合本项目的具体工作，由于分布式防火墙系统中，专门开发了日志与审计服务器模块，而且，针对最小特权管理机制，本文第三章基于 Linux 内核能力(capability)信任状模型进行了论述与解决。因此，这里着重阐述借助可卸载内核模块（LKMs）方法实现 BLP 模型存取控制机制，以达到系统增强的目的。从安全增强的具体实现来看，是以系统调用为基元，面向安全策略逐个分析系统调用以及新增系统调用的安全性。与安全无关的，不作改动；与安全有关的，加入相应的存取控制策略。这些存取控制策略就是 BLP 模型的安全规则。

Linux 内核级安全增强的具体做法是用加载模块的方法修改和安全有关的系统调用。对于文件保护，当调用被修改的系统调用时，先检查文件的保护类型及相关进程的安全级，若没有保护或属于非保护类型，则返回原来的系统调用。反之，根据文件的存取控制机制来选择打开模式，或返回错误类型。例如，对于被列为只读保护的文件，如果合法用户以只读模式打开文件，则返回原来的系统调用；若对只读保护的文件试图以写的模式打开，则返回错误。对于进程保护，为了保护重要的进程，使之不能被删除，可以在进程的标记位上设置一些未被操作系统使用的标志位（如安全级标志，保护位标志等）来保护重要的进程，并修改 Kill 调用，在真正执行 Kill 调用前先检查标志位，系统将拒绝用户删除设置保护位的进程。另外，在对内核进行加固后，应禁止插入或删除模块，从而保护系统的安全，否则入侵者将有可能再次对系统调用进行修改。我们可以通过替换 create_module()和 delete_module()来达到上述目的。当然，也可以通过内核能力机制来删除加载/卸载内核模块的能力。

事实上，修改系统调用可以通过两种方法来实现。第一种是直接修改系统的核心代码，然后重新编译生成新的核心。该方法的缺点是：每做一次修改都需要对系统进行重新编译，这给新核心代码的调试带来了相当大的困难。若系统管理员需要针对不同用户进行配置，重新编译的工作量相当巨大；第二种方法是将对系统的修改内容做成一个模块，通过静态或动态地加载和卸载，该模块会修改系统调用入口。应用模块技术，可以减小系统核心代码的规模，而且在需要时才装入模块可以减小系统所占用的硬件资源，从而提高系统的性能。模块的代码在装入核心后与核心中其他代码的地位是相同的，但代码的调试就方便得多了。若管理员针对不同用户

进行配置，只需修改模块配置文件或在装载时传递参数。

例如，为了有效的进行存取控制，可以考虑用一个 LKMs 记录每一个模块的加载，并且拒绝任何一个不是从指定安全目录加载的模块的企图。记录功能可以通过拦截 create_module(...)来很轻易的实现。用同样的方法也可以检查模块加载的目录。当然拒绝任何模块的加载也是有可能的。但这并不是一个好方法，因为大量系统功能需要模块的支持。因此可以考虑改变模块的加载方式，比如说要一个密码。密码可以在修改后的系统调用 create-module（...）里面检查。如果密码正确，模块就会被加载，否则，模块被丢弃。

（1）模块加载检测器原型：

```
#include
.....
extern void* sys_call_table[];
int (*orig_create_module)(char*, unsigned long);
int hacked_create_module(char *name, unsigned long size)
{
char *kernel_name;
char hide[]="ourtool";
int ret;
kernel_name = (char*) kmalloc(256, GFP_KERNEL);
memcpy_fromfs(kernel_name, name, 255);
/*这里向 syslog 记录，但是也可以记录到任何想要的地方*/
printk("<1> SYS_CREATE_MODULE : %s\n", kernel_name);
ret=orig_create_module(name, size);
return ret;
}
int init_module(void)
/*初始化模块*/
{
orig_create_module=sys_call_table[SYS_create_module];
sys_call_table[SYS_create_module]=hacked_create_module;
return 0;
}
void cleanup_module(void)
```

```
/*卸载模块*/
{
sys_call_table[SYS_create_module]=orig_create_module;
}
```

原理很简单，只不过是拦截了 sys_create_module(...)，并且记录下加载模块的名字。

（2）密码保护的模块加载原型：

给一个模块的加载加入密码校验。需要两件事情来完成这项任务：一件是检查模块加载的方法，另一件是密码校验的方法。第一点是十分容易实现的，只需要拦截 sys_create_module(...),然后检查一些变量，内核就会知道这次加载是否合法。而密码校验的方法很多，这里仅仅给出一个简单的例子，拦截 stat(...)系统调用，敲命令的同时敲一个密码,LKMs 会在拦截下的 stat 系统调用中进行检查。

```
#include ……
extern void* sys_call_table[];
/*如果 lock_mod=1 就是允许加载一个模块*/
int lock_mod=0;
……
/*拦截 create_module(...)和 stat(...)系统调用*/
int (*orig_create_module)(char*, unsigned long);
int (*orig_stat) (const char *, struct old_stat*);
……
int hacked_stat(const char *filename, struct old_stat *buf)
{
char *name;
int ret;
char *password = "password";
/*一个很好的密码*/
name = (char *) kmalloc(255, GFP_KERNEL);
(void) strncpy_fromfs(name, filename, 255);
/*有密码吗?*/
if (strstr(name, password)!=NULL)
……
int hacked_create_module(char *name, unsigned long size)
```

```
{
……
if (lock_mod==1)
{
lock_mod=0;
ret=orig_create_module(name, size);
return ret;
}
else
{
printk("<1>MOD-POL : Permission denied !\n");
return 0;
}
……
```

事实上，可以新增一个单独的系统调用来完成密码校验，并且能够得到更好的安全性。在借助可卸载内核模块（LKMs）实现 BLP 模型存取控制机制，具体实现安全系统调用时，还必须结合 BLP 安全模型，进行存取控制策略的安全性分析，如每一步的权限，安全级比较，客体不存在的处理方法等，使控制检查机制本身是不可绕过的，从而保证整个系统的健壮与安全。

目前，Linux 系统只有不完备的自主访问控制，也存在一些因系统安全机制和内核调用规则不完善而造成的安全漏洞。通过借助可卸载内核模块截获并改进系统调用函数，强化存取控制机制和调用点的审计。对于原系统有所欠缺的环节，如安全级比较，模块加载时的密码校验等，则采取新增系统调用函数的形式来实现这些功能。结合 BLP 模型的安全规则，对 Linux 系统内核函数进行全面的安全性改进与增删后，可以有效实现 Linux 下的标识与鉴别、自主访问控制、强制访问控制、审计、最小特权管理、客体重用等安全特性。

第七章 结束语

7.1 联调与测试

对 Linux 系统裁减优化及扩展开发完成之后，经单独调试、排错，在本人和课题组成员的共同努力下，将 Linux 系统装入工控机，作为边界防火墙服务器。并同分布式防火墙系统中的日志模块，入侵检测模块等进行了整合与互连。为了实现一套功能完备的分布式防火墙系统原型，达到相关的国家标准，课题组对防火墙系统进行了整体联调与测试。

（1）操作系统功能与稳定性的测试结果。

对照本项目软件需求说明书，作为防火墙运行支撑环境的操作系统，提供了通用函数库（Glibc2.3.2）支持，SSL,SSH，MD5 加密协议及 PAM,Kerberos 认证协议支持，启用路由转发，但禁止源路由功能。为了保证操作系统的稳定性，对系统进行了长时间的运行实验，并在内网开启日志服务器，入侵检测服务器，保持较大网络流量，经过 3 天连续运行，系统工作正常。

（2）串口控制台测试结果。

本项目的目标之一就是使用串口控制台来管理防火墙服务器。因此，串口必须支持 Root 权限登录，在使用串口虚拟终端对服务器连续进行了两次 7×24 管理模式运行，控制台对 Vi，ls，more 等 Linux 应用程序保持了兼容，成功地在串口虚拟终端对 Linux 系统进行如脚本编写，文件配置等一系列管理工作。

7.2 总结与展望

本课题的主要工作是针对 Linux 操作系统所具有的安全特性和安全缺陷，完成了 Linux 系统裁减，安全增强的设计、配置和一些安全扩展功能的实现。论文围绕 Linux 安全增强和访问控制，对系统从初始化安装加载、脚本执行、最小特权管理到控制台扩展等具体功能的实现中涉及到的主要技术和关键部分进行了总结和说明。

常规 Linux 服务器系统安装后占用空间一般在 1GB 以上，但作为防火墙的支撑系统，其中许多服务功能与守护进程（如 httpd，lpd 等），我们并不需要，而且，过多的系统服务也导致出现安全漏洞的可能性增大。因此，我们基于软件包模式裁减了 Linux 系统，只保留必需的功能，使整个操作系统尽可能精简。考虑到系统中必须实现一些通用操作系统的功能，不能象嵌入式系统一样针对特定目的进行裁减。最终，在防火墙服务器上实现了一个占用空间 100 余 MB，但提供了防火墙所有必需功能的精简 Linux 系统。

论文详细介绍了 Linux 安全增强系统中主要功能构件的配置、设计和实现，包括 LKMs 可加载内核模块，Linux 系统调用。同时详细介绍了内部各功能构件之间的函数接口和例程。

本系统主要在细化 Root 权限、规范化配置脚本和访问控制方面，结合 BLP 安全模型，对 Linux 操作系统的安全功能进行增强，实现了 Linux 下的标识与鉴别、最小特权管理、自主访问控制、强制访问控制、审计与客体重用等安全特性。

Linux 操作系统安全增强的设计考虑到安全性和易于实现性，目前在 BLP 模型下，利用可卸载内核模块，基于系统调用改进与增删实现了包括访问控制在内的多项安全特性。但还有很多方面有待完善，例如在身份鉴别上沿用了 Linux 操作系统的登录时口令鉴别身份的方式，未能实现更加严格的身份鉴别，如采用智能 IC 卡等特殊信息进行身份鉴别。

操作系统是一个复杂庞大的系统，支持各种应用和服务，从而具有多种安全隐患。信息时代的网络化，使得网络传递的安全性也逐渐成为操作系统安全的重要内容。因此，系统安全增强不可能是一劳永逸的，必须根据需求的变化，不断地更新和完善，才能真正适应实际应用的需要。

参考文献

[1] 博嘉科技 主编 Linux防火墙技术探秘，国防工业出版社，2002

[2] 刘启原，刘怡编著 数据库与信息系统的安全，科学出版社，2000

[3] [美] Aron Hsiao 著，史兴华译，Linux系统安全基础，人民邮电出版社，2002

[4] Bell，La Padula，L.，“Secure Computer System: Mathematical Foundations and Model”，Technical Report M74-244，The MITRE Corporation，1973

[5] Losococco P A，Smalley S D. Integrating Flexible Support for Security policies into the Linux Operating System. Technical Report，NSA and NAI labs，2001

[6] 卿斯汉，刘文清，刘海峰著 操作系统安全导论，科学出版社，2003

[7] 陈莉君编著 深入分析Linux内核源代码，人民邮电出版社，2002

[8] 代玲莉，欧阳劲编著 Linux内核分析与实例应用，国防工业出版社，2002

[9] [美] Evi Nemeth，Garth Snyder著，张辉译，Linux系统管理技术手册，人民邮电出版社，2003

[10] Andreas Gruenbacher，Extended Filesystem Attributes，Linux kernel maillist，7 May 2000

[11] 秦超，段云所，陈钟 访问控制原理与实现 网络安全技术与应用，2001，5：54－58.

[12] 姚顾波，刘焕金等编著 黑客终结——网络安全完全解决方案，电子工业出版社，2003

[13] 姬东耀，张福泰，王育民 多级安全系统中访问控制新方案 计算机研究与发展，2001，38（6）：715－720.

[14] 刘文清，刘海峰，卿斯汉 基于Linux开发安全操作系统的研究 计算机科学，2001，28（2）：52－54.

[15] Sandhu，R.S.Samarati Access control: principle and practice IEEE Communications Magazine:40－48.

[16] Trusted Computer System Evaluation Criteria DOD 5200 28-STD US Department of Defense 1985

[17] GB17859-1999 中华人民共和国国家标准：计算机信息系统安全保护等级划分准则 1999

[18] 毛德操，胡希明编著 Linux内核源代码情景分析 ，浙江大学出版社 ，2001

[19] 张斌，高波编著 Linux网络编程，清华大学出版社，2000

[20] Official Linux Capability FAQ

http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt

[21] Linux kernel source code (experimental version 2.4.18)

http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.18.tar.gz

[22] Walker.P.J.,Harris.M.A regulatory view of access control. IEEE Colloquium: 4/1-4/6.

致　　谢

首先深切感谢导师袁宏春教授领我进入这一充满无限生机的研究领域。袁老师严谨求实的科研精神、真诚平和待人的品格将使我受益终身，是我学习的榜样。他的因材施教、循循善诱的教育风格对我毕业后的工作将产生深刻的影响和启迪。在我的整个研究生学习期间，袁老师为我创造了良好的学习和宽松的研究环境，不仅在学习上给予我悉心指导，还在生活上，思想上给了我极大的帮助，教我为人的道理，使我受益不少。在此，表示我深深的谢意！

王蔚然教授在我整个研究生学习期间给了我很大的关怀、支持和帮助，为我提供了充分的学习和实践机会，使我得到了多方面的锻炼，在此表示衷心的感谢！

感谢方里冰老师、钱伟中老师对我学习所给予的帮助。

感谢在课题组工作过的唐寅博士、李志军博士，以及学友王红雷硕士、冯爱娟硕士等，与他们的良好协作与探讨开阔了我的思路。

感谢孙永奎硕士、孙金源硕士等在本论文的完成过程中所给予的支持和帮助。

诚挚地感谢我的父母对我的支持、理解和关心。感谢妻子对我的支持、理解和关心，她在承担繁重的科研任务的同时，总是尽可能抽出时间陪伴我走过一个个攻克技术难题的不眠之夜，在心灵上给予我极大的鼓励与安慰！

最后，衷心感谢为评阅本论文而付出辛勤劳动的各位专家。

个人简历、攻读硕士期间研究成果及发表的论文

● 个人简历

刘瑜，男，1975 年 1 月生，四川成都人，1997 年 7 月毕业于重庆大学材料科学与工程学院，获工学学士学位。2001 年 9 月至今在电子科技大学计算机学院攻读计算机应用技术硕士学位。

● 攻读硕士期间研究成果

[1] 参研信息产业部电子发展基金项目《新型分布式防火墙系统》，负责其中一个子课题：Linux 操作系统裁减、功能扩展与安全增强。

● 发表学术论文

[1] 刘瑜，袁宏春. BLP 安全模型在 Linux 系统增强中的应用与实现.计算机应用