多保护域进程模型及其实现

谢 钧[1,2], 黄 皓[1], 张 佳[1]
(1. 南京大学计算机软件新技术国家重点实验室, 江苏南京 210093;
2. 解放军理工大学指挥自动化学院, 江苏南京 210007)

摘 要: 在很多安全操作系统中都存在一些安全关键进程或可信进程, 一旦它们被黑客入侵则会破坏整个系统的安全性. 本文的多保护域进程模型在进程内部通过细粒度的内核级保护域隔离机制对进程数据和代码实施访问控制, 从而防止黑客利用程序局部漏洞劫持整个进程, 以达到增强安全关键进程自身安全的目的. 本文为该模型提供了两种设计方案并对其中一种设计做了原型实现.

关键词: 保护域; 访问控制; 计算机安全; 操作系统
中图分类号: TP393 文献标识码: A 文章编号: 0372-2112 (2005) 01-0038-05

A Multi-Protection Domains Process Model and Its Implementation

XIE Jun[1,2], HUANG Hao[1], ZHANG Jia[1]
(1. State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu 210093, China;
2. Institute of Command Automation, PLA University of Science and Technology, Nanjing, Jiangsu 210007, China)

Abstract: Many secure operating systems have some privileged processes or trusted processes which are always at risk of being hijacked by various attacks such as the buffer overflow attack. Once they are hijacked, the security of the whole system would be damaged. In this paper, a multi-protection domains process model is described which provides fine-grained kernel level protection for codes and data within process address space. The fine-grained internal protection of process can effectively prevent attackers from hijacking the whole process by damaging the process. s data or codes. This paper offers two designs for this model and a prototype implementation of one of them.

Key words: protection domain; access control; computer security; operating system

1 引言

在传统的多用户操作系统中, 利用访问控制机制可以限制用户的权限, 防止用户对信息的非法访问和破坏. 但 root 用户却拥有无限的特权, 因此 root 进程成为被攻击的主要目标. MLS(Multilevel Security) 系统要求信息只能从低密级实体流向高密级实体, 但在实际的系统中为实现某些功能, 总有一些进程不得不违反这样的安全策略. 由于我们必须信任这些进程, 因此它们被称为可信进程. 一旦这些可信进程的代码中存有漏洞就可能被黑客利用进而破坏整个系统的安全性. Capability(权能) 系统[1]和 DTE(Domain and Type Enforcement)[2]系统根据最小特权原则为每个进程分配尽可能少的特权或权限来降低不安全进程给整个系统所带来的危害. 但实际上一个进程在其执行过程中并不总需要同样的权限, 如: 进程在刚开始执行时可能需要访问某个文件进行初始化, 但以后就不需要再访问该文件; 或进程在与用户交互期间不允许创建进程, 等等. 传统操作系统的安全机制都是基于用户或进程的, 不能阻止黑客利用程序漏洞来控制整个进程, 也不能灵活地动态改变进程的特权. 因此有必要提供一种内核级机制来增强这些安全关键进程的自身安全.

本文的多保护域进程模型将进程的不同执行阶段与不同的保护域标记相关联, 并为不同的保护域授予不同的访问或操作权限. 当进程执行在不同的保护域中时, 对进程的数据和代码的访问受多保护域访问控制机制的监控, 不同的保护域对同一数据或代码可以具有不同的访问权限. 由于在进程内部实现了细粒度的保护机制, 大大增强了进程的自身安全性, 特别是能有效地抵御利用缓冲区溢出、格式串错误等程序漏洞的攻击, 而缓冲区溢出是近十年来被利用最为广泛的安全漏洞形式[3]. 将进程保护域机制和进程权能机制结合, 可以动态而安全地改变进程的权能, 使进程在不同的执行阶段具有更为精确的权限.

2 多保护域进程模型

在多保护域进程模型中, 一个进程的用户内存空间根据

收稿日期: 2004-01-05; 修回日期: 2004-09-04
基金项目: 江苏省自然科学基金(No. BK2002073)

功能和安全策略的要求可以被划分为多个区域, 同时根据安全策略的要求进程在不同的执行阶段可以与不同的保护域标记相关联, 即进程可以在不同的时刻在不同的保护域中执行. 进程对各个内存区的访问必须受多保护域内存访问控制机制的监控. 多保护域内存访问控制机制通过访问控制矩阵 M_1 来控制进程对内存区的访问. 访问控制矩阵 M_1 的行为保护域标识, 列为划分的内存区标识, $M_1[d, m] < \{r, w, x\}$表示当进程在保护域 d 中执行时对内存区 m 所允许的访问模式(其中 r, w, x 分别表示读、写、执行). 当进程在不同的保护域中执行时, 对同一内存区可以具有不同的访问权限, 如图 1 所示.

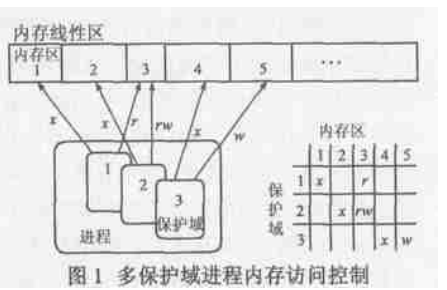

图 1　多保护域进程内存访问控制

进程一开始在初始保护域中执行, 在执行过程中通过调用一个特殊的执行点可以进入其它保护域执行, 这个特殊的执行点就是各保护域的入口点. 每个保护域可以有多个入口点, 进程只有通过调用这些预定义的入口点才能从一个保护域进入另一保护域. 多保护域访问控制机制通过保护域调用访问控制表 M_2 决定是否允许保护域的切换. $M_2[d_1, e] = y, eI$ (Entrys(d_2) , 表示 e 是保护域 d_2 的入口点, 允许 d_1 通过调用 e 进入 d_2. 如图 2 所示, 保护域 1 可以调用保护域 3 的入口点 1 进入保护域 3, 保护域 3 能调用保护域 4 的入口点 1 进入保护域 4, 但保护域 1 却不能直接进入保护域 4.

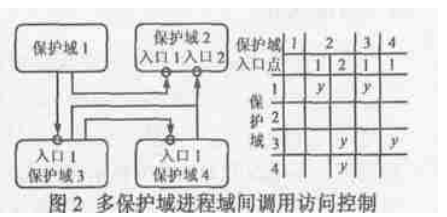

图 2　多保护域进程域间调用访问控制

多保护域进程模型利用内存区访问控制实现了对进程数据和代码的细粒度的保护, 使得各保护域在内存空间上能根据安全策略隔离开来, 限制了攻击者利用程序漏洞(如缓冲区溢出、格式串错误等) 来控制整个进程的能力, 有效提高了进程抗攻击的能力. 缓冲区溢出攻击是其中最典型且使用最为广泛的一种攻击方式, 大致可分为 3 大类: (1) 利用缓冲区插入恶意代码或利用缓冲区溢出篡改进程代码并利用缓冲区溢出修改跳转指令或函数指针执行该恶意代码; (2) 利用缓冲区溢出修改跳转指令或函数指针改变程序的正常执行次序; (3) 利用缓冲区溢出修改程序状态数据(如函数参数等) , 扰乱程序的正常执行并迫使程序进行非法活动. 第一类攻击最容易实现也最为普遍, 而第三类攻击较难实现, 往往要和前两类攻击共同使用才能达到攻击效果. 一个复杂的攻击可能包括所有这 3 类方法. 对于第一类攻击, 模型可对所有数据段(包括堆栈) 采用不可执行的访问模式来阻止恶意代码的插入, 并将所有进程的代码段设置为只读以防止非法篡改原有程序代码, 完全杜绝了第一类攻击. 对于第二类和第三类攻击, 利用进程内存访问控制机制将程序数据按不同访问要求划分到具有不同访问模式的区域中, 使入侵者不能随意修改程序的数据, 很难利用缓冲区溢出达到其攻击目的. 模型还能利用保护域调用访问控制机制强制进程的某些执行顺序, 实现 Assured Pipeline 完整性模型[4], 防止某些关键模块被非法绕过, 大大降低了第二类攻击成功的可能性. 以上特性对于所有利用程序漏洞非法篡改进程数据和代码的攻击都是有效的, 例如利用程序格式串错误漏洞的攻击以及特洛伊木马代码模块攻击等.

进程在其执行过程中并不总是需要同样的权限, 根据最小特权原则, 进程应该在确实需要某项特权时才赋予它该特权, 因此进程的权限应能动态改变. Unix 系统提供 seteuid 系统调用来允许进程在需要时暂时降低自己特权级而后再恢复原来的特权, 但这可能导致安全漏洞: 如果在进程降低特权时攻击者控制了该进程, 当恢复进程原特权时会使攻击者具有更大的破坏能力. Linux 系统提供权能机制来限制进程所拥有特权, 特别是限制 root 进程的无限特权, 并能通过系统调用来消减进程的权能. 多保护域进程模型的各保护域在内存空间上能根据安全策略互相隔离, 因此多保护域机制与 seteuid 以及权能机制结合可以用来实现进程特权或权能的动态降低与安全恢复, 使进程在不同的执行阶段具有更精确的特权.

3　基于 i386 体系结构的设计

本文多保护域进程模型的设计与实现都是基于 Intel 386 (i386) 体系结构的, 但由于 i386 及其兼容体系结构在桌面计算机市场中占有大多数份额, 因此该设计与实现仍然具有广泛的应用价值和重要性.

311　基于段表的设计

由于 i386 中通过分段机制实现从逻辑地址空间到线性地址空间的映射, 因此我们把进程的线性地址空间按不同的模块和访问需求划分为多个不同长度的段, 同时为同一进程的不同保护域分配不同的局部描述符表 LDT(在一般的操作系统中每个进程只有一个 LDT) . 进程各保护域的 LDT 中各段的地址空间映射完全一样, 但各段的访问模式不同, 这样不同保护域对同一地址空间(段) 就可以具有不同的访问模式, 从而通过 i386 的硬件分段机制实现了多保护域进程模型中的内存访问控制. 如图 3 所示, 保护域 1 对内存区 3 的访问模式是/ 只读0, 而保护域 2 对该内存区的访问模式却是/ 读写0. 由于 i386 段硬件保护机制, 所有违反这些访问模式的内存访问指令都会引起一个段保护异常.

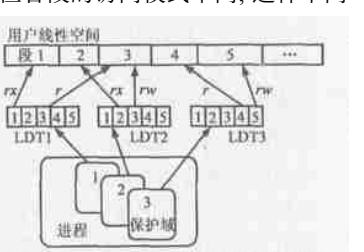

图 3　基于段表的多保护域进程设计

为了实现进程在保护域之间的迁移, 需要在内核中增加两个新的系统调用来提供保护域间的切换服务. 保护域切换的主要任务就是切换进程的当前保护域标识, 局部描述符表, 保护域的上下文, 保护域堆栈, 调用目的保护域的入口函数. 入口函数执行完后应返回原保护域, 通过调用保护域返回系

统调用返回原保护域. 由于不涉及进程调度, 线性地址空间不变即不更改页表, 是一种轻量级上下文切换. 为此各保护域的入口函数地址必须在内核中注册. 为了减轻内核的负担, 可以将保护域域间调用访问控制放在各入口函数中自行处理, 所有的入口函数都有一个参数来标识调用保护域, 入口函数可自行判断是否允许域变迁.

3.2 基于页表的设计

通过分段机制可以很方便地将进程的用户线性空间划分为多个不等长的内存空间, 但是由于虚拟地址中的段选择符保存在段寄存器中, 当进行跨段访问内存单元时需要修改和保存段寄存器中的内容, 因此段间指针的处理比段内指针的处理要复杂得多, 并且 gcc 等大多数高级语言编译器都不能直接支持多段编程, 这会给应用程序员增加编程的复杂性. 同时由于内核段与用户段的不同对所有系统调用中所使用的指针参数要进行修改使它们在不同的段中能对应线性空间的同一地址, 因此需要修改几乎所有的系统调用. 为了避免多段编程的这些困难, 可以利用 i386 体系结构的分页机制来实现内存地址空间的划分与隔离.

在 i386 体系结构中, 当启动分页机制时, 进程的线性地址空间被划分为等长的页, 通过页表(这里统指页目录表和页表) 线性空间中连续的若干页可以被映射到物理地址空间中不连续的任意页框. 页表项中除了包括该页在物理地址空间中的起始地址外还包括该页被允许的访问模式. 在传统操作系统中每个进程都对应着一组页表来实现该进程的线性空间到物理空间的映射. 在多保护域进程模型基于页表的设计中, 根据安全策略进程的用户线性空间被划分为多个逻辑的区域(不等长) , 同一区域所对应的所有页的访问模式是一样的, 在内核中通过专门的数据结构来管理和维护这些逻辑的内存区域. 同时为该进程的每个保护域分配一组页表, 不同保护域页表的地址空间映射完全一样, 但对相同区域的页表项中的访问模式可能不一样, 如图 4 所示同一页在不同保护域的页表中的访问模式可能不同. 在保护域切换时, 切换当前保护域的页目录而不是段表. 这样利用 i386 的分页机制同样实现了多保护域进程的内存区域访问控制机制, 但由于没有采用多段模式, 避免了处理段间访问的地址变换问题.

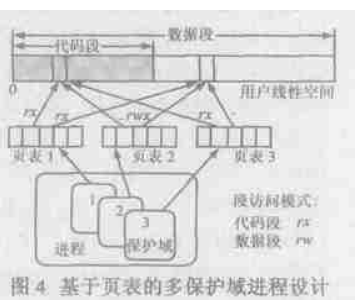

图 4 基于页表的多保护域进程设计

由于 i386 的页保护机制只支持/ rx0 和/ rwx0 两种模式, 我们利用 i386 的段保护机制来支持更多的访问模式. 为了避免段间地址的不同, 如图 4 所示为每个进程设置两个不同长度的用户段: 代码段和数据段, 但这两个段的起始地址都从 0 开始. 代码段的段访问模式是/ rx0, 而数据段的段访问模式为/ rw0, 其中数据段覆盖进程所有的线性空间, 最终进程对内存单元的访问模式是段/ 页访问模式的交集. 因此只要将所有可执行的内存区分配到该进程的代码段所在的线性空间, 而将所有不可执行的内存区分配到该进程代码段以外的线性空间中, 就能完全实现多保护域进程模型的所有访问模式. 例如: 将进程所有代码都装载到进程的代码段中同时代码所在的内存区的页访问模式设置为/ rx0, 而所有的数据都分配在该进程代码段以外的线性空间中, 这样就可以实现进程的所有数据都不能被执行而所有代码都不能被篡改的安全策略.

由于进程所有段的起始地址都是 0(包括内核段), 因此对于跨段的访问不存在变换段内偏移量的问题. 特别是在进行系统调用时, 由于内核段和用户段的起始地址相同无需进行地址转换(内核数据通过分页保护机制保护) . 因此这种分段的方法既利用了 i386 段保护机制来增强进程的安全性, 同时又避免了段间访问的复杂性. 但该方法也有不利的方面: 一是由于只有代码段才允许执行, 在装入程序时必须将所有代码都装入到该线性区域, 而所有数据都要装入到代码段以外的线性区域. 这使得进程的内存管理缺少灵活性, 特别是为了支持动态链接, 各共享库的代码段及其数据段不能连续分配, 在程序装入时要在进程代码段为这些动态链接库的代码预留空间. 二是由于进程的保护域切换要切换页目录, 需要刷新地址转换后援缓冲器 TLB, 这会影响一些性能. 而利用多段表来实现多保护域进程模型没有这两个问题. 利用段表来实现多保护域进程模型还有另一个好处是段机制具有越界保护功能, 缓冲区溢出不能跨段进行, 更有利于抵御缓冲区溢出攻击.

4 原型实现

本文基于 i386 体系结构的多保护域进程原型系统是在 Linux 2.4 基础上实现的, 并且遵循 Linux 尽可能少地使用分段机制的原则, 我们采用基于页表的设计来实现多保护域进程模型而避免段间地址的变换给应用程序员所带来的一系列复杂性. 在此只讨论几个实现中的关键问题.

4.1 线性空间的区域划分与访问控制

原型系统的进程描述符中有一个用户内存区域链表, 该链表中每个节点对应该进程的一个线性地址空间区域, 并包含一个保护域访问控制列表, 该列表中包含所有能访问该线性区的保护域标识和允许的访问模式(页访问模式) . 由于 Linux 将所有实际的物理内存分配都推迟到缺页异常处理中完成, 当发生缺页异常进行物理页面分配时, 根据缺页所在线性区的访问控制列表对所有能访问该线性区的保护域的页表进行修改, 将该页的访问模式设置为对应保护域对该线性区的访问模式. 由于进程在不同保护域中执行时使用该保护域对应的页目录与页表, 从而通过页保护机制实现了多保护域进程模型的内存访问控制.

4.2 保护域的切换

进程通过显式地执行 ent. pdomain 系统调用来请求从一个保护域进入另一个保护域并执行被调用保护域中的入口函数, 当该入口函数执行完所请求的服务后返回原调用保护域. 内核在切换保护域时相应地要切换当前保护域标识和当前页目录, 然后以低特权级调用目的保护域的入口函数. 内核负责将调用域标识作为参数传入被调用域, 域间调用访问控制由入口函数自行处理. 但一个关键问题是在 i386 体系结构中只

允许低特权代码通过 int 指令调用高特权代码并通过 iret 指令从高特权代码返回低特权代码, 而要实现保护域切换内核必需调用低特权的目的保护域入口函数. 在原型系统中内核调用保护域入口函数前在内核堆栈中构建一个返回用户态的虚拟上下文, 其中的用户堆栈指针和指令指针分别被设置为目的保护域的堆栈栈顶和入口函数地址. 然后执行 iret 跳转到目的保护域的入口函数开始执行, 而不是返回原调用保护域. 当该入口函数执行完所请求的服务后要返回原调用保护域时, 调用 ret_pdomain 系统调用恢复原调用保护域的执行.

4.1.3　对 Linux 信号处理的修改

原 Linux 内核在调用用户态的信号处理函数前先要在用户态堆栈中插入执行 sigreturn 系统调用的代码, 以便在信号处理函数执行完后能自动返回内核. 但由于在我们的原型系统中进程的用户堆栈(在保护域代码段外) 利用段保护机制不允许执行, 因此必须对原 Linux 的信号处理进行修改. 信号处理函数的执行是从内核态调用用户态中的函数, 与内核调用用户态的保护域入口函数所要作的工作类似, 因此我们采用与内核调用保护域入口函数相同的方法来处理对信号处理函数的调用.

4.1.4　基于保护域的进程动态权能管理

当用户进程调用 register_pdomain 注册保护域时, 可同时为各保护域设置不同的权能(只能从原进程权能中删减), 在进程的保护域描述符中有保护域权能列表保存各保护域拥有的权能. 当进程调用 ent_pdomain 切换保护域时, 内核将进程的权能设置为该保护域的权能, 当 ret_pdomain 返回时恢复原进程权能, 从而实现进程权能的动态改变.

4.1.5　用户程序接口及系统兼容性

为了简化原型系统的实现, 支持多保护域的程序使用显式地利用系统调用来完成保护域的注册及内存保护空间的申请. 这样只修改内核代码而无需修改编译链接程序就能实现多保护域进程模型. 所有多保护域进程开始执行在初始保护域中, 在初始保护域中进程可以调用 register_pdomain 来注册保护域、其入口函数及设置域权能. 进程在初始保护域中可通过调用 create_parea 来申请被保护的内存空间并设置其访问控制列表. 在各保护域中进程可以通过 ent_pdomain 来调用其它保护域的入口函数并切换进程的当前保护域, 通过调用 ret_pdomain 来返回原调用保护域. 各保护域对进程内存单元的访问方式不变, 由硬件分段分页机制完成保护域的内存访问控制.

由于采用的是显式系统调用的方式, 所有原有用户程序都能以原有方式继续在系统中运行, 系统对原有应用具有完全的兼容性.

5　性能测试与分析

由于在原型系统中进程保护域对内存保护区的访问就是一般的进程内地址空间访问, 因此多保护域进程的性能开销主要增加在域间调用, 为了参照比较, 我们在 PC 机上(英特奔腾 Ⅳ1.7GHz 处理器, 256M 内存) 对几种调用方式的空调用进行了测试, 测试结果如表 1 所示.

表 1　空调用性能测试

调用方式	域内函数调用	系统调用	保护域间调用	进程间通信
时间	0.015Ls	0.92Ls	2.3Ls	8.1Ls
特权切换次数	0	2	4	8

由表 1 可以看出保护域间调用大约是系统调用的 2.5 倍, 主要原因是因为保护域调用需要穿越内核边界 4 次而系统调用只需要两次, 以及人为构建硬件上下文和地址解析后援缓冲器的刷新对性能的影响. 由于进出内核边界的次数少于进程间通信, 并且保护域间的调用无需涉及进程调度, 因此保护域间调用的性能要优于进程间通信.

6　相关工作

进程内部地址空间隔离保护机制可分为两大类: 基于软件和基于硬件的保护机制. 基于软件的保护机制如软件故障隔离技术 SFI[5] 及 type2safe 语言[6] 等, 该方法在程序编译或装载时在程序代码中插入内存访问检查指令, 并在运行时检查相关函数调用和内存访问的合法性, 由此带来的开销分别在 1%～220% 和 10%～150%[7]. 基于硬件的保护机制除了本文提出的方法外, 还有 Chiueh 基于特权级的机制[7] 以及 Takahashi 基于分段的机制[8]. 基于硬件的保护机制在检查内存访问合法性时不增加性能开销, 但在进行域间调用时需要跨越内核边界 4 次, 而基于软件的保护无需切换特权级. 因而一般来说硬件方式具有性能优势, 但当保护域之间频繁切换时采用基于软件的方法可能会获得更好的性能. Chiueh 和 Takahashi 的保护机制主要用于多组件应用中保护进程不被恶意代码组件模块破坏, 它们都不能支持数据空间不可执行的安全策略, 因而不能有效地抵御缓冲区溢出等插入恶意代码的攻击. Chiueh 的用户进程只能有两个保护域, 虽然 Takahashi 的保护进程可以拥有多个保护域, 但由于其进程(而不是保护域) 的实际线性空间大小随进程保护域个数 n 的增加而线性递减(是原线性空间的 1/n), 因此不能适用于较多保护域的领域. Process Cleaning 技术[9] 在进程降低特权之前复制进程的内存空间和状态, 当进程恢复特权时恢复进程原来的内存空间和状态, 从而实现进程特权的安全恢复, 但不能支持不同进程特权对同一进程地址空间的不同访问要求和共享.

7　结论

在各种系统中特权进程特别是各种网络服务进程是被攻击的的首要目标, 而抵御这类攻击的传统方法主要有两个: 限制进程的权限和入侵检测. 而多保护域进程模型一方面在进程内部通过细粒度的内核级保护域隔离机制对进程数据和代码实施强制的访问控制, 从而防止攻击者通过缓冲区溢出等手段利用程序局部漏洞破坏程序的数据和代码或劫持整个进程, 以达到增强安全关键进程自身安全的目的; 二是实现基于保护域的权能机制动态限制进程的特权更细致地实行最小特权原则. 同时多保护域进程模型可以与模块化的程序设计方法结合, 在设计程序时就关注程序的安全性, 将操作系统的保护机制与程序设计方法有机地结合起来, 充分利用系统安全特性增强应用程序自身安全.

参考文献:

[1] J S Shapiro, J M Smith, D J Farber. EROS: A fast capability system [A]. In Proc. 17th ACM Symposium on Operating Systems Principles [C]. New York, 1999. 170- 185.

[2] K M Walker, D F Sterne, M L Badger, et al. Confining root programs with domain and type enforcement(DTE) [A]. In Proc 6th USENIX Security Symposium[C]. Washington DC, 1996. 21- 36.

[3] C Cowan, P Wagle, C Pu, et al. Buffer overflows: Attacks and defenses for the vulnerability of the decade[A]. In Proc DARPA Information Survivability Conference and Expo[C]. Hilton Head SC, 2000. 1119- 1130.

[4] W D Young, P A Telega, W E Boebert. A verified labeler for the secure ada target [A]. In Proc 9th National Computer Security Conference [C]. Gaithersburg MD, 1986. 55- 61.

[5] R Wahbe, S Lucco, T E Anderson, S L Graham. Efficient software2based fault isolation [J]. ACM Operating Systems Review, 1993, 27(5): 203- 16.

[6] B N Bershad, T E Anderson, E D Lazowska, et al. Extensibility, safety and performance in the SPIN operating system[J]. ACM Operating Systems Review, 1995, 29(5): 267- 284.

[7] T C Chiueh, G Venkitachalam, P Pradhan. Integrating segmentation and paging protection for safe, efficient and transparent software extensions [A]. In Proc 17th ACM Symposium on Operating Systems Principles [C]. New York, 1999. 140- 153.

[8] M Takahashi, K Kono, T Masuda. Efficient kerenel support of fine2grained protection domains for mobile code[A]. In Proc 19th IEEE International Conference on Distributed Computing Systems[C]. Austin TX, 1999. 64- 73.

[9] K Kourai, S Chiba. A Secure Access Control Mechanism Against Internet Crackers[R]. ISE2TR201176, Institute of Information Sciences and Electronics, Univ. of Tsukuba, 2001.

作者简介:

谢 钧 男, 1973 年 1 月出生于四川广元, 解放军理工大学指挥自动化学院讲师, 南京大学计算机科学与技术系博士生, 主要研究领域为信息安全和计算机网络. E2mail: yulumail@263. net.

黄 皓 男, 1957 年 6 月出生于江苏南京, 南京大学计算机科学与技术系教授, 博士生导师, 主要研究领域为计算机网络信息安全.