KCapISO：一种基于 HybridHP 的宏内核操作系统载入模块权能隔离方案

钱振江[1),2),3)] 刘永俊[2)] 汤 力[2)] 姚宇峰[2)] 黄 皓[1)]

[1)](南京大学计算机科学与技术系 南京 210023)

[2)](常熟理工学院计算机科学与工程学院 江苏 苏州 215500)

[3)](伦敦大学国王学院 伦敦 WC2R 2LS 英国)

摘 要 宏内核操作系统提供对第 3 方模块和驱动程序等载入模块的支持，允许载入模块运行在内核态特权级. 由于运行在最高特权级，载入模块对内核的核心服务的关键对象的访问难以得到系统的有效控制. 考虑对被监控系统的性能影响控制在很小的范围，基于内嵌式的监控机制 HybridHP，提出了一种宏内核架构下的载入模块权能隔离方案 KCapISO，为内核和载入模块维护各自的页表，从权能上将两者隔离，确保载入模块无法修改内核的数据，并且无法以任何方式直接调用或跳转到内核中执行，这些动作都需经过 KCapISO 的监控和检查. 实验结果表明，KCapISO 能有效地将内核与载入模块在权能上相互隔离，同时获得较好的系统性能.

关键词 宏内核操作系统；载入模块；硬件虚拟化；权能隔离；安全监控

中图法分类号 TP316 **DOI 号** 10.11897/SP.J.1016.2016.00552

KCapISO: A HybridHP-Based Capability Isolation Method of Loaded Modules on Monolithic Kernel Operating System

QIAN Zhen-Jiang[1),2),3)] LIU Yong-Jun[2)] TANG Li[2)] YAO Yu-Feng[2)] HUANG Hao[1)]

[1)](*Department of Computer Science and Technology, Nanjing University, Nanjing* 210023)

[2)](*School of Computer Science and Engineering, Changshu Institute of Technology, Suzhou, Jiangsu* 215500)

[3)](*King's College London, London WC2R 2LS, UK*)

Abstract The monolithic kernel operating systems provide support for the loaded modules, such as third-party modules and drivers. Due to that the loaded modules run on the privilege level, the access to the key objects of core services within the kernel is difficult to be controlled effectively. Considering little influence on the performance of the monitored system, based on the embedded-style monitoring system HybridHP, we propose a capability isolation method of loaded modules on the monolithic kernel operating system, named KCapISO. KCapISO maintains the respective page tables for the kernel and loaded modules, which are isolated from the aspect of the capability. KCapISO ensures that the loaded modules cannot modify the kernel data, and cannot directly call or jump into the kernel in any way. And these behaviors are required to pass through monitoring and inspection by KCapISO. The experiment result shows that KCapISO effectively isolates the kernel and loaded modules from the aspect of the capability, and achieves good system performance.

Keywords monolithic kernel operating system; loaded module; hardware virtualization; capability isolation; security monitoring

---

收稿日期：2014-09-03；最终修改稿收到日期：2015-06-18. 本课题得到国家自然科学基金(61402057)、江苏省科技计划自然科学基金(BK20140418)、中国博士后科学基金(2015M571737)、江苏省"六大人才高峰"高层次人才基金(2011-DZXX-035)和江苏省高校自然科学研究基金(12KJB520001)资助.**钱振江**，男，1982 年生，博士，讲师，中国计算机学会(CCF)高级会员，主要研究方向为操作系统安全、形式化验证和嵌入式系统. E-mail：tony_h@sina.com. **刘永俊**，男，1981 年生，博士研究生，讲师，主要研究方向为操作系统安全. **汤 力**，男，1980 年生，硕士，副教授，主要研究方向为操作系统安全. **姚宇峰**，男，1981 年生，博士研究生，讲师，主要研究方向为操作系统安全. **黄 皓**，男，1957 年生，博士，教授，博士生导师，主要研究领域为系统软件、信息安全.

1 引　言

宏内核(Monolithic Kernel)操作系统(Operating System，OS)，如 Linux 等，提供了对载入模块(包括第 3 方模块和驱动程序)的支持，且允许载入模块运行在内核态特权级. 由于运行在最高特权级，载入模块对内核的核心服务的关键对象的访问难以得到系统的有效控制，恶意的第 3 方模块或者受到感染的驱动程序可以很容易地破坏系统调用表、页表、IDT 表、系统寄存器和网络端口等系统关键对象，从而破坏整个系统的完整性.

与之相比，在微内核(Microkernel)架构的 OS 中，载入模块运行在用户态模式，受到处于内核态特权级的微内核的控制，达到了很好的模块隔离性和可控性. 为此，在微内核 OS 平台上，可运行的内核病毒程序屈指可数. 而 Windows 和 Linux 等宏内核平台上内核病毒初步统计就有数十类以及不计其数的变种，其中大部分的内核攻击都是通过加载载入模块来达到的.

值得一提的是，微内核 OS 由于使用消息机制进行进程间通信(Inter-Process Communication，IPC)而存在性能问题，主要是由于通信的两个服务进程，或者服务进程与用户进程的进程空间不相交，无法直接访问而必须使用消息进行通信，而且接收进程必须等待微内核将消息复制到自己的进程空间才能对消息进行处理. L4[1] 针对上述的缺陷，改进了 IPC 的实现，将单次通信所需的 2 次消息拷贝减少为 1 次，性能得到了提高. 在宏内核平台上，载入模块与内核处于同一进程空间，同时内核服务也存在于这一进程空间，无需进行上述的消息拷贝，这是一个很好的特点，但是也带来了之前所述的安全问题.

随着硬件虚拟化技术的发展，在宏内核平台上，采用虚拟机监控器(Virtual Machine Monitor，VMM)对被监控系统进行运行时监控的研究工作[2-10]获得了很好的支持. 但是这种方法存在着 3 个方面的主要问题：(1) 系统性能. 基于 VMM 的监控方案需要管理域(Domain 0)以及虚拟机管理器(Hypervisor)的支持，同时被监控系统的 I/O 等操作需要管理域的干预，这样造成被监控系统的性能损失较大；(2) 监控粒度. 如何确定监控的粒度是一个两难的问题，监控粒度越大，对被监控系统的性能损失较小，但监控的正确度不高，监控粒度越小，监控的正确度越大，但对被监控系统的损失较大；(3) 主体标示. 在监控的过程中，首先需要对被监控系统中的各个执行主体(对象)进行标示，但由于内核和载入模块都运行在内核态特权级，并且位于同一内存地址空间，因此对于监控内核的 VMM 而言，难以具体标识当前执行主体为内核还是载入模块.

很多学者在利用 VMM 对不可信的载入模块的扩展给系统带来的安全性问题进行监控方面进行了研究[6-7]，在主体标示问题上提出了可行的方案，对执行主体和资源如内存页等进行部分的权限设置，但需要记录状态转换，并且存在着频繁切换页表问题. 同时存在代理攻击的问题，如果不可信的载入模块通过一系列的操作来请求内核服务进行一些非法的操作，VMM 往往很难辨认执行主体.

本文提出一种宏内核架构下的载入模块权能隔离方案 KCapISO，将内核(包括可信的内核扩展模块)与载入模块在权能上相互隔离，虽然它们处在同一进程空间中，但载入模块只能修改自己定义的数据. 为达到这样的目的，有两个方面的前期工作. 首先，我们对 Linux 源码进行了分析，认为 Linux 中内核与载入模块虽然同属于一个进程空间，但它们之间很少直接修改对方的数据，而主要以读取对方的可导出数据为主，这一点为我们的隔离方案提供了可能. 其次，我们在前期工作中设计并实现了一种轻型的内核完整性监控方案 HybridHP[11]，其利用硬件虚拟化机制，以完全内嵌的模块化方式在被监控系统中运行并提供监控服务，无需借助管理域以及虚拟机管理器的支持，为此被监控系统的性能损失控制在很小的范围，这一点为我们的隔离方案提供了“监控机制”的保证.

KCapISO 借助 HybridHP 的监控机制，使用页目录地址寄存器 CR3 来标示系统的执行主体，实现对内核和载入模块权能隔离. 区别于其他的注重内核接口保护的驱动隔离方案，KCapISO 以对象模型为基础，认为对象才是 OS 内核的核心，而包括内核本身、载入模块等在内的执行主体，其实质都是围绕内核对象进行操作. 在现代 OS 中，页表权限是对象权能的最直接体现，KCapISO 通过页表标识执行主体，并且进行权限控制，从而达到权能隔离的目的.

本文第 2 节介绍系统监控和隔离方面的相关工作，并和我们的工作进行比较；第 3 节描述 KCapISO 的具体设计方案；第 4 节阐述 KCapISO 的关键技术

和实现;第 5 节阐述我们对 KCapISO 的实验和性能分析;第 6 节对本文进行总结,并对我们未来的工作进行展望.

2 相关工作

Garfinkel 等人[2]提出利用虚拟机监控器来实现一个受保护的系统完整性监视器,包括基于不变量的内核保护.其实现的 Livewire 系统能够校验内核的代码区,通过不同层次的系统检索,查找能检测到的特定类型数据攻击,并校验静态函数指针表,如系统调用表.KCapISO 基于的监控框架 HybridHP 同样利用虚拟化技术来实现,但是 HybridHP 并不是构建在现成的虚拟机之中,避免了由于对虚拟机的攻击而导致的对监控器的破坏.

Seshadri 等人[3]利用硬件虚拟化技术实现了一个轻量级的监控器 SecVisor,用于保证 OS 代码的完整性.SecVisor 保证只有用户允许的代码才能在内核中执行,确保未经授权的内核代码不会被执行.SecVisor 没用采用成熟的虚拟机监控器,目的是减少开销和简化验证.但因为需要在被监控系统和监控器之间进行频繁地页表转换,性能影响较大.

NICKLE 监控系统[4]采用内存映射的技术,使得只有经过认证的执行代码才能被复制到影子内存中,同时也只有影子内存中的代码才可以执行.但是对载入模块的认证是困难的,认证策略过分严格会影响系统的可用性,而策略过分松散又可能无法检测出恶意的代码.另一方面,内存映射需要在虚拟机中执行,并且需要能够捕获和模拟执行所有的指令,这不仅限制了硬件和虚拟机的范围,也增加了执行开销.

Sharif 等人[5]在被监控 OS 中构建了一个监控器 SIM,以便于取得被监控系统的语义.这和 HybridHP 的思想类似,不过 SIM 与被监控系统的隔离保护仍需要另一个虚拟机管理器的支持,而 HybridHP 的自我保护不依赖于其他任何安全机制.

为了监控载入模块对内核 API 的调用,Srivastava 等人[6]利用虚拟化技术设计了 Gateway 系统.Gateway 实现了一个不可绕过的监控接口,通过该接口对不可信的载入模块的运行空间进行隔离.同时,为了克服隔离所带来的性能损失,Gateway 还引入了内核代码和驱动代码的动态重写,使得监控性能损失控制在 10%以内.

HUKO[7]是一个基于 Xen 实现的内核完整性监控方案.在 HUKO 监控框架中,载入模块能够稳定地提供原有的功能,但是其行为将受到强制性访问控制的约束,因此,对系统整体性能的影响较大.

Xiang 等人[8]提出的监控框架 ComMon 从网络、进程和文件等 3 个方面对系统进行实时透明的监控.通过对系统调用的监控,ComMon 可以检测系统中隐藏的进程,并捕获进程对文件的操作.ComMon框架中加入了对网络入侵的检测,可以保护系统的网络安全.

Osiris[9]是一种针对恶意软件进行分析的系统.Osiris 通过对应用程序编程接口(API)的监控技术来分析系统的行为.测试结果表明 Osiris 取得了良好的监控效果.

vDetector[10]是一款针对 OS 隐藏对象进行关联检测的系统.与 ComMon 类似,vDetector 对网络、进程和文件等隐藏文件通过 OS 的不同视图之间的差异性进行检测,同时通过对这些隐藏对象之间的关联进行分析来检测对系统的攻击行为.

Castro 等人[12]提出的软件故障隔离方法 BGI,采用字节细粒度内存保护机制(Byte-Granularity Isolation)来有效地将内核扩展模块隔离在独立的保护域中.BGI 能够确保内核对象的类型安全性,可以检测保护域中的普通错误类型,并用于驱动测试中以寻找驱动漏洞.

Mao 等人[13]提出的 LXFI 系统对内核和内核模块进行隔离,提出了 API 完整性(API integrity)和模型主体(module principals)的概念.通过编译器插件,LXFI 根据程序员声明的 API 完整性条件和模型主体来构造代码,对模块的权限进行管理,以阻止特权提升(privilege escalation)的威胁和攻击.

本文提出的 KCapISO 区别于上述的监控和隔离方案,以对象模型为基础,借助 HybridHP,使用页目录地址寄存器 CR3 来标示系统的执行主体,实现对内核和载入模块权能隔离,使得载入模块只能修改自己定义的数据.

3 KCapISO 的设计

3.1 KCapISO 的目标

我们首先描述内核完整性的威胁模型.一般而

言，内核受到攻击的场景包括：(1) 攻击者获得了超级用户 Root 权限，然后将恶意程序加载进 OS 内核空间；(2) 攻击者利用已有的系统漏洞加载 Rootkit[14] 之类的恶意软件；(3) 粗心的用户加载了未经认证的包含恶意代码的载入模块等. 我们将 OS 内核中的执行主体分成两大类：(1) OS 内核自身以及可信的内核扩展模块；(2) 不可信的载入模块. 所有的 Rootkit 和未经验证的驱动程序都属于第 2 类.

KCapISO 通过强制性访问控制达到以下两个完整性目标：

(1) 内核代码与数据和完全相关数据的完整性. 所有内核代码与数据，以及安全相关数据都不允许被上述第 2 类执行主体通过直接的内存操作或者直接内存访问(Direct Memory Access，DMA)操作修改；

(2) 体系相关寄存器的完整性. 体系相关寄存器包括段寄存器、控制寄存器、以及部分标志寄存器，它们共同标志了 OS 内核的运行状态，它们不允许被上述第 2 类执行主体修改.

KCapISO 的目标是通过隔离和监控技术来实现对 OS 内核的权能隔离，但却不会带来大的性能损失. 为此，需要实现以下 3 个具体目标：

(1) 能够实现内核和载入模块权能隔离，这是我们工作的初衷；

(2) 内核和载入模块权能的隔离与监控并不会给被监控系统带来大的性能损失；

(3) 易于移植，不需要过多地修改 OS 内核本身.

3.2　KCapISO 的框架

KCapISO 提供了一个保护 OS 内核免受不可信的载入模块破坏的安全环境. 在 KCapISO 中，我们将需要保护的内核对象称为安全相关客体，这些对象包括内核代码和内核关键数据组成的集合.

为了说明 KCapISO 的思想和实现，我们首先明确以下几个概念：

(1) 对象模型. KCapISO 以对象模型为基础，认为对象才是 OS 内核的核心，而包括内核本身、载入模块等在内的执行主体，其实质都是围绕内核对象进行操作. 已有的很多基于驱动隔离的研究工作[2-10,12] 都把保护 OS 内核代码和接口放在首要的位置. 我们认为，代码和接口不是保护内核完整性的关键，因为内核的代码或者接口功能是可以在不可信的载入模块中重现的，所以即使对 OS 内核的代码和接口保护地再好，也只能说明破坏内核完整性的难度增加了. 为此，我们的工作将重点放在对内核关键数据对象的保护上；

(2) 内核主体. 在 KCapISO 框架中，我们把 OS 内核本身和载入模块标识为不同的内核执行主体；

(3) 权能. 在 KCapISO 框架中，权能是指对于各种不同对象的访问权限，具体表现为对应于不同的页表项的属性.

在 KCapISO 的监控框架中，我们需要解决的关键问题是：主体标示的方法和监控的性能. 下面对这两个问题进行阐述.

为了标识执行主体，KCapISO 首先把系统状态分成 3 类：用户态(User Mode)、核心态(Kernel Mode)以及载入模块运行状态(Third-party Service Mode). 用户态是指用户程序的运行状态；核心态是内核的运行状态，包括安全认证过的内核扩展模块的运行；载入模块运行状态是指未经过安全认证的服务模块和驱动程序等的运行状态. 需要说明的是，我们划分状态的依据并不是侧重于代码或者数据，而是注重权能，即当前运行主体所拥有的权限. 这些状态之间的转换过程包括：(1) 用户程序在运行过程中因为中断或者异常以及系统调用等原因，导致系统由用户态进入核心态，由内核进行处理；(2) 内核从中断或者异常以及系统调用的处理过程中返回，导致系统由内核态进入用户态；(3) 内核调用载入模块，或者载入模块抢占内核的运行时间，导致系统由内核态进入载入模块运行状态；(4) 载入模块运行完成，或者内核抢占载入模块的运行时间，导致系统由载入模块运行状态进入内核态. 系统状态转化关系如图 1 所示.

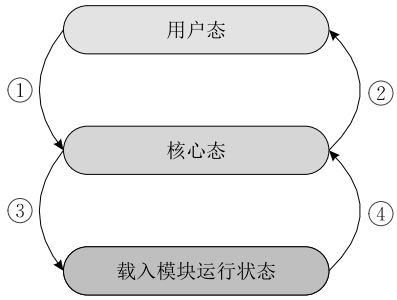

图 1　KCapISO 框架中系统状态转化图

为了实现对内核关键数据对象的保护，最简单的做法是干预对这些对象的每一次操作，这一方法的问题在于系统整体性能的损失非常大，即使被监

控的对象不多，相应的监控开销也是非常巨大的，因为很多的内核关键数据对象本身会被内核频繁地访问，比如 Linux 内核中的进程控制块对象 *task_struct*，一方面，恶意程序为了实现隐藏进程的目的，需要修改 *task_struct* 链，但另一方面，*task_struct* 也是调度程序需要经常修改的数据对象. 如果简单的干预对 *task_struct* 的每一次访问，那么由于 OS 和监控器之间频繁的上下文切换而带来的性能开销将会是非常巨大的.

为此，KCapISO 通过对载入模块设置不同的页表来达到标示主体和保证监控性能的目的. 一方面是为了标志主体，对于特定内核关键数据对象的访问，我们很难在底层进行判断是由载入模块发起的，还是由内核自己发起的；另一方面是性能方面的考虑，对相同的内核关键数据对象，由于内核和载入模块具有不同的访问权限，如果使用相同的页表，那么我们必须按照最低权限进行页表的配置，导致的结果是监控器需要干预对该对象的每一次访问，不管是内核还是载入模块的访问，这导致的结果就是监控性能的极大下降. 通过对载入模块设置不同的页表，我们在其各自的页表中设置对内核关键数据对象的访问权限来起到标示主体的作用，同时在对内核关键数据对象的访问的监控过程中，可以很好地避免频繁地干预，提高系统的整体性能. KCapISO 的隔离结构如图 2 所示.

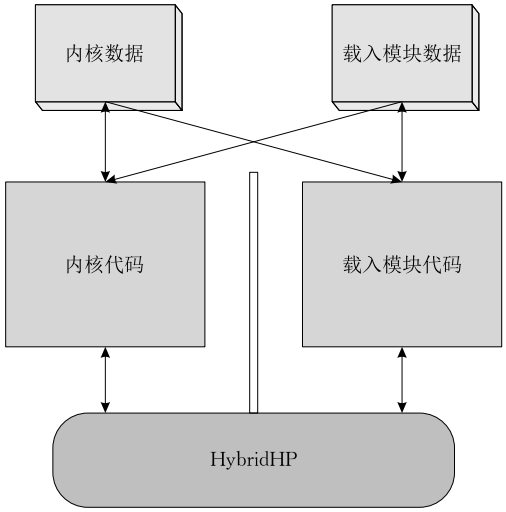

图 2 KCapISO 隔离结构图

页表的设置如图 3 所示，载入模块的页表中用户数据页和载入模块数据页属性设置为可读可写，用户代码页、内核数据页和载入模块代码页属性设置为只读，内核代码页属性设置为不映射；内核的页表中用户代码页、用户数据页和内核数据页属性设置为可读可写，内核代码页和载入模块数据页属性设置为只读，载入模块代码页属性设置为不映射.

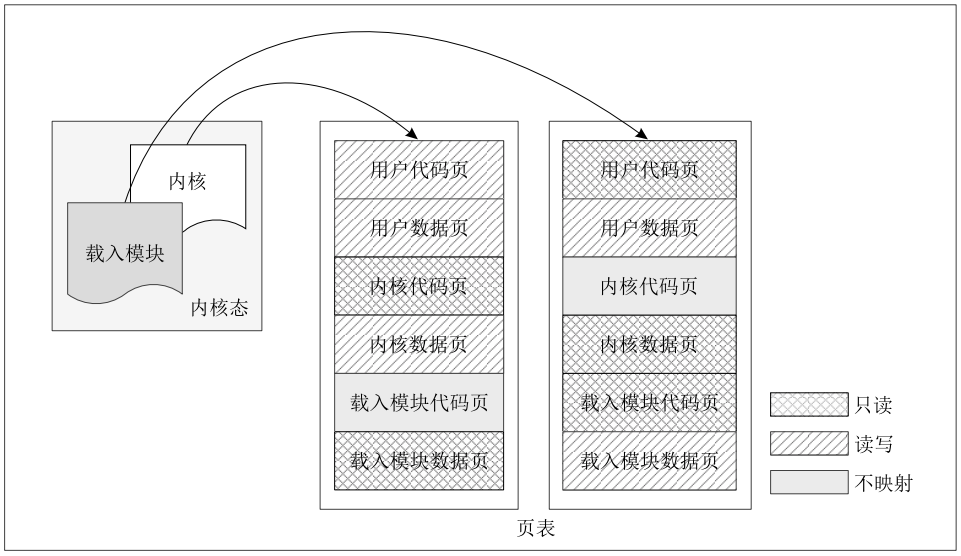

图 3 页表配置

为此，载入模块直接修改内核的数据时，由于内核的数据页对载入模块而言是只读的，因此会产生页保护错误，系统陷入 HybridHP 监控器中，HybridHP 根据预定策略进行处理，处理结束后系统从 HybridHP 监控器返回.

当载入模块直接调用、跳转到内核的代码页时，由于内核代码页在当前载入模块的页表中未映射，所以会触发缺页异常，系统陷入 HybridHP 监控器中，HybridHP 通过转换页目录地址寄存器 CR3 为其切换页表，在通过监控检查后，HybridHP 为载入

模块调用内核的接口程序.由于内核允许读取载入模块的数据,因此,无需将参数拷贝到内核中.同样,函数的返回与函数调用类似,也需通过 HybridHP 转换页表.但是载入模块内的函数调用不经过 HybridHP.上述过程保证了载入模块无法以任何方式直接调用或跳转到内核中执行,都需经过 HybridHP 的检查.

另外,内核态与用户态之间的转换不需要切换 CR3,因为系统调用的入口和返回处理的代码在内核中,所以在内核态与用户态的切换前后都一定使用的是内核页表,所以无需切换,这样很好地避免频繁地切换页表,提高系统的整体性能.

4　KCapISO 的关键技术和实现

4.1　地址空间内存隔离

在宏内核系统如 Linux 中,载入模块在运行过程中,与内核处于同一地址空间,可以看到彼此的全部内存视图.KCapISO 的驱动隔离包括两方面的概念:线性地址空间内存隔离和物理地址空间内存隔离,其具体表现为载入模块有专属的页表,并且来自载入模块的内存访问等操作不能破坏内核所在的物理内存.

为了实现地址空间内存的隔离,需要解决的问题是如何在不过多修改 Linux 系统内核的情况下,实现内存视图的隔离.为此,在系统初始化阶段,为处在虚拟内存 3 GiB～(4 GiB－1)的内核空间生成载入模块用页表和内核用页表,两份页表中线性地址到物理地址的映射相同,此时载入模块用页表和内核用页表均为空页表.

通过配置 HybridHP 监控器中的虚拟化控制结构 VMCS,使得 HybridHP 监控陷入指令 CPUID 的执行.当加载载入模块时,通过在载入模块初始化系统调用的头部加入陷入指令 CPUID,使得在执行模块初始化之前,系统陷入 HybridHP 中.HybridHP根据当前载入模块的标示(*init_module*)取得载入模块代码的首地址、载入模块代码的长度、载入模块数据的首地址、载入模块数据的长度,并取得内核代码的首地址和长度、内核数据的首地址和长度,计算出载入模块和内核所在的内存页,从而根据图 3 对载入模块用页表和内核用页表进行页属性的设置.设置完成后,系统从 HybridHP 返回.同样,当卸载载入模块时,恢复相应的页表设置.

当 OS 创建新进程时,为进程创建两份页表,其中 0～(3 GiB－1)的用户空间的页表相同,而 3 GiB～(4 GiB－1)内核空间的页表则分别复制上述设置好的载入模块用页表和内核用页表,如图 4 所示.

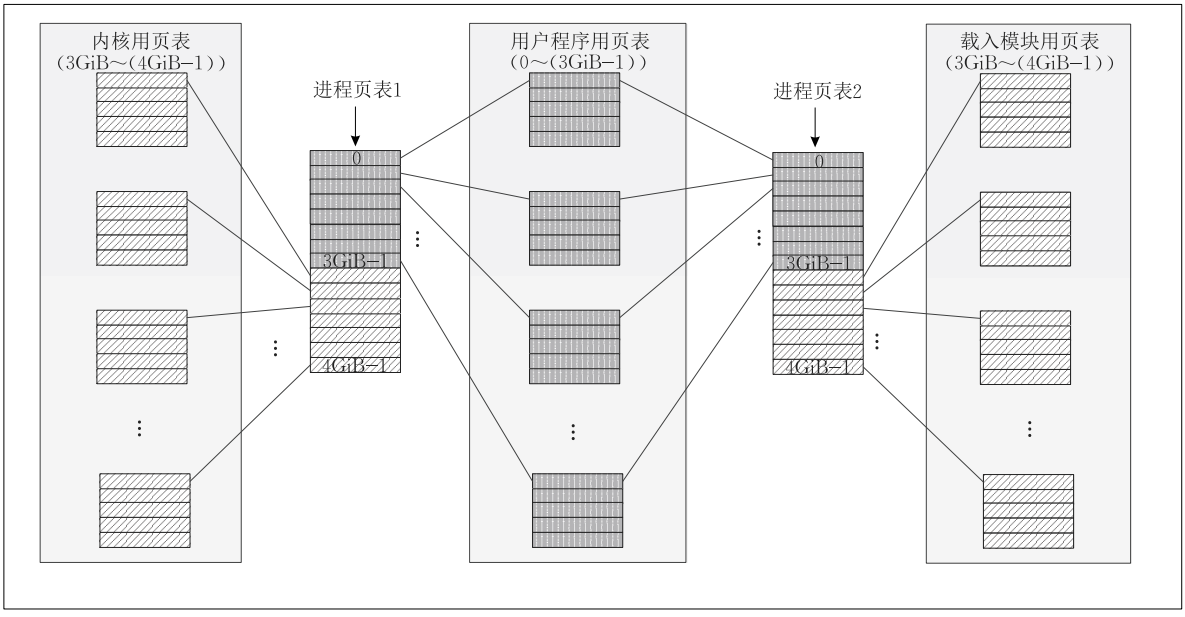

图 4　进程页表初始化和同步

4.2　内核栈的隔离设计

为了隔离内核和载入模块的代码、数据和堆栈,除代码、静态数据使用独立的页外,还需为每一个进程临时分配载入模块的内核栈.在 KcapISO 框架中,每一个进程占用 1 个 4 KB 大小的页来作为载入模块的内核栈.为了管理这些页,在内核中增加一个

管理链表,链表节点包括载入模块内核栈的页首地址、栈顶、栈中保存的返回值地址和栈中位置等信息,并根据进程号索引该临时栈的链表节点.

4.3 动态申请的数据空间保护

一个程序的资源除了静态的代码、数据、堆栈、页表等,还包括动态申请的数据空间,如何对这些数据空间进行隔离保护也是 KCapISO 需要考虑的问题. 通过配置 HybridHP 中的虚拟化控制结构 VMCS,使得 HybridHP 监控动态内存空间分配(kmalloc、vmalloc 等)函数的调用,并且编写和设置单独的动态内存空间申请函数(cmalloc),当载入模块调用动态内存空间分配函数申请动态内存时,系统陷入 HybridHP 中,HybridHP 调用 cmalloc. cmalloc 函数的执行流程是向内核申请一个整页,并在此后再次收到载入模块的请求时,直接在该页中分配空间,直到不够分配为止,再重新向内核请求整页的空间. 每当分配到一个新页,则根据图 3 所示的页表策略设置载入模块用页表和内核用页表的页属性,与动态内存对应的载入模块用页表中载入模块数据页属性设置为可读可写,与动态内存对应的内核用页表中载入模块数据页属性设置为只读. 完成动态内存的分配之后,系统从 HybridHP 返回.

4.4 基于 HybridHP 的页表切换设计

KCapISO 的页表切换是指内核页表和载入模块页表之间的切换.

通过配置 HybridHP 中的虚拟化控制结构 VMCS,使得 HybridHP 监控缺页异常. 当载入模块直接调用、跳转到内核的代码页时,按照图 3 的页表设置情况,由于内核代码页在当前载入模块用页表中未映射,所以会触发缺页异常,系统陷入 HybridHP 中,HybridHP 通过转换 CR3 为其切换页表,从载入模块用页表切换到内核用页表. 在通过 HybridHP 检查后,HybridHP 取得引起陷入监控器的跳转地址,检查跳转地址的合法性;保存当前的栈顶指针寄存器 ESP,切换 ESP 到内核的栈中,将载入模块的内核栈中之前压入的输入参数压入到内核的栈中,恢复当前的 ESP;修改 HybridHP 中的虚拟化控制结构 VMCS,将其中的指令寄存器 EIP 指向目标跳转地址,ESP 指向内核的栈的栈顶. 然后 HybridHP 为载入模块调用内核的接口程序,系统从 HybridHP 返回,内核的接口程序继续执行.

通过配置 HybridHP 中的虚拟化控制结构 VMCS,使得 HybridHP 监控内核的接口程序的返回动作. 当内核的接口程序执行完成并返回时,系统再次陷入 HybridHP 中,HybridHP 通过转换 CR3,从内核用页表切换到载入模块用页表,HybridHP 取得引起陷入监控器的跳转地址,检查跳转地址的合法性;保存当前的栈顶指针寄存器 ESP,切换 ESP 到载入模块的内核栈中,弹出载入模块的内核栈中压入的输入参数,恢复当前的 ESP;修改 HybridHP 中的虚拟化控制结构 VMCS,将其中的指令寄存器 EIP 指向目标返回地址,ESP 指向载入模块的内核栈的栈顶,然后系统从 HybridHP 返回到载入模块.

通过配置 HybridHP 中的虚拟化控制结构 VMCS,使得 HybridHP 监控页保护错误. 当载入模块直接修改内核的数据页时,按照图 3 的页表设置情况,由于内核的数据页对载入模块而言是只读的,则会触发页保护错误,系统陷入 HybridHP 中,HybridHP根据预定策略进行处理,处理结束后系统从 HybridHP 返回.

4.5 混合页的处理

内存管理是以页为单位,而 OS 内核管理则是以对象为单位,这带来的问题是具有不同权限的内核对象可能处于相同的页表中. 对混合页的处理所带来的监控性能损失是肯定的.

在传统的虚拟化监控方案中,混合页是影响监控性能的一个重要方面. 在 KCapISO 方案中,为内核和载入模块配置不同的页表,在核心态下,系统处于可信任状态,无须受控,因此混合页的问题不再存在. 在载入模块运行状态下,混合页的问题依然存在,这是难以避免的. 总的来说,由于地址空间内存的隔离以及 HybridHP 的内嵌方式,我们可以很大程度地降低这种开销.

5 实验与性能分析

KCapISO 方案对内核和载入模块的权能隔离以及监控性能是实验的重点. 我们的测试平台是 HP Pavilion 500-171cn 台式电脑,i7-4770 处理器,8GB DDR3 内存. 测试的软件环境为 Ubuntu 14.04 LTS x86 32bit 系统,3.14.4 版本 Linux 内核.

5.1 监控效果分析

为了说明 KCapISO 方案的有效性,我们使用如表 1 所示的 rootkit 进行测试. 我们对表 1 中的 rootkit 进行了修改,使其能在测试系统上运行. 这些攻击代表了公认系统漏洞数据库(NVD①)典型的

① National Vulnerability Database. http://nvd.nist.gov/

内核攻击方式，都是通过类似载入模块的形式被加载到内核空间，从而修改系统核心数据来破坏内核的运行.

表 1　内核 rootkit 攻击类型

rootkit 名称	加载方式	附注
adore 0.42	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
adore-ng 0.56	第 3 方模块方式加载	修改/proc 和/ext3 文件系统的 4 个函数指针
all-rootkit	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
enyelkm	第 3 方模块方式加载	修改 idt_handler，伪造 sys_call_table
kdb v2	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
kis 0.9	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
knark 2.4.3	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
linspy v2beta2	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
maxty 0.1	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
modhide	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
mood-nt 2.3	驱动方式加载	修改 sys_call_table，替换系统调用
override	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
phantasmagoria	第 3 方模块方式加载	修改代码段和 DO_SYSCTL 函数
phide	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
rial	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
rkit 1.01	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
taskigt	第 3 方模块方式加载	通过修改 proc_dir_entry 的函数指针，调用恶意代码
Shtroj2	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
SucKIT 1.3b	驱动方式加载	修改系统调用代码
SucKIT2 prev	驱动方式加载	修改系统调用代码
Superkit	驱动方式加载	修改系统调用代码
synapsys 0.4	第 3 方模块方式加载	修改 sys_call_table，替换系统调用
THC Backdoor	第 3 方模块方式加载	修改 sys_call_table，替换系统调用

实验结果表明，KCapISO 方案通过对载入模块设置不同的页表，并借助 HybridHP 的监控机制，将内核与载入模块在权能上相互隔离，虽然它们处在同一进程空间中，但载入模块只能修改自己定义的数据，无法破坏内核的运行状态，为此能有效地监控和阻止表 1 所示的 rootkit 攻击.

5.2　系统性能分析

为了分析 KCapISO 方案对系统整体性能的影响，我们使用常用程序（解压文件和编译）以及基准测试程序 UnixBench① 来进行测试. 我们采取与传统的基于 XEN 的和 SIM 方案的性能对比来说明 KCapISO 的性能. 在对 XEN 和 SIM 方案的测试环境中，管理域 Domain0 以半虚拟化方式运行.

图 5 说明了被监控系统在 3 种监控方式下的性能对比，结果与 HybridHP 方案单独运行时的对比实验[11] 基本相同. 由于 HybridHP 没有管理域 Domain0，在这些对比实验中，作为整个监控框架的 KCapISO 对被监控系统的性能影响控制在很小的范围，均接近实际物理系统的性能.

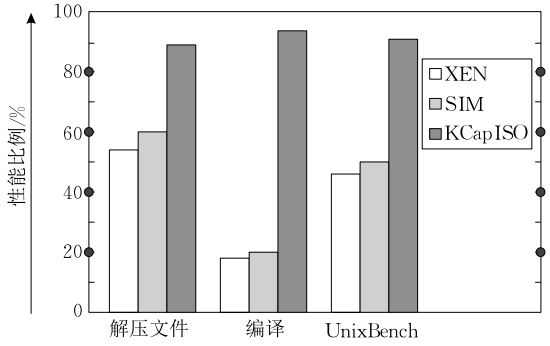

图 5　被监控系统性能对比图

为了说明 KCapISO 对系统创建进程行为的性能影响，本文在上述 3 种监控方式下对创建进程进行单独实验. 表 2 说明了 3 种监控方式下监控机制带来的性能影响. 与 SIM 相比，KCapISO 的时间增长比率为 10.76%，在创建进程的过程中，KCapISO 为进程创建和复制两份页表所耗费的时间占了一定的比例. 为此，我们可以认为 KCapISO 对被监控系统的性能影响控制在很小的范围.

表 2　创建进程操作的性能对比

监控类型	平均时间/μs	时间增长比率/%
裸机	3.262	NULL
XEN	29.137	793.20
SIM	3.754	15.08
KCapISO	3.613	10.76

本文对 KCapISO 方案中系统运行模式切换的时间进行了测试. 在 KCapISO 方案中，我们通过修改内核，在系统运行模式切换的响应和处理过程中以获取系统时间的方式来计算模式切换的时间. 在上述裸机环境下，一次模式切换耗时约 1 μs. 在 KCapISO 方案中，除了系统运行模式的切换外，还有切换 CR3 所引起的 TLB(Translation Look-aside Buffer)表刷新. 通过测算，每次切换最多需要 15 μs. 但是，对于进程管理等大部分系统调用无需载入模块的参与，并且内核正常情况下不会修改保护数

① UnixBench. http://www.tux.org/pub/benchmarks/System/unixbench/

据,如不修改页表等,为此系统性能几乎与不安装 KCapISO 时的相差无几.同时,我们对 KCapISO 方案中页表切换所带来的性能损失进行了测试,如表 3 所示.从表 3 的测试结果来看,在实际的系统运行过程中,即使频繁地进行页表切换,KcapISO 对被监控系统的性能影响控制在很小的范围.所以本文认为,通过隔离载入模块,同时对内核采用较宽松的策略,能够使系统在其性能下降较小的情况下实现对载入模块的权能隔离.

表 3 KCapISO 方案中页表切换对系统性能的影响

测试项	时间增长比率(相对裸机环境)/%	描述
载入模块,被载入的模块不调用内核函数	6.34	内核初始化模块的过程,载入模块的初始化函数被调用,并且载入模块不调用内核函数
载入模块,被载入的模块调用内核函数	12.58	内核初始化模块的过程,载入模块的初始化函数被调用,并且载入模块调用内核函数,如 printk 等
在监控策略视 vfat 文件系统模块为可信模块的条件下,执行文件操作的系统调用,如 open、write、read 和 close 等操作	2.60	监控策略视 vfat 文件系统模块为可信模块,反复读写 U 盘文件(100 M)100 次
在监控策略视 vfat 文件系统模块为不可信模块的条件下,执行文件操作的系统调用,如 open、write、read 和 close 等操作	9.42	监控策略视 vfat 文件系统模块为不可信模块,反复读写 U 盘文件(100 M)100 次

同时,在不考虑为载入模块所申请的临时内核栈的情况下,KCapISO 仅使用 16 个内存页作为监控器内存使用,对系统执行的影响控制在很小的范围.

6 结 论

本文提出一种宏内核架构下的载入模块权能隔离方案 KCapISO,将内核与载入模块在权能上相互隔离,虽然它们处在同一进程空间中,但载入模块只能修改自己定义的数据,并且载入模块无法以任何方式直接调用或跳转到内核中执行,都需经过 KCapISO 的监控和检查.KCapISO 借助 HybridHP 的监控机制,使用页目录地址寄存器 CR3 来标示系统的执行主体,进行权限控制,从而达到权能隔离的目的.

KCapISO 的实现还处于原型阶段,对于多核的支持以及对 slab 算法分配的小数据结构所占用的内存页实现动态、高效的语义分析将是下一步的主要工作.值得一提的是,在多核环境中,由于内核和系统模块可以分布在多个物理核上,甚至内核本身可以有多个镜像同时分布在多个物理核上,多个核之间以及与系统模块之间对页表等数据的访问存在竞争和同步的问题,如何保证 KCapISO 方案在多核环境中监控能力的不可绕过性以及页表的多份拷贝等是需要重点解决的问题.

致 谢 本文作者感谢所有本文的匿名审稿者,感谢你们对本文提出宝贵的意见!

参 考 文 献

[1] Liedtke J. On μ-kernel construction//Proceedings of the 15th ACM Symposium on Operating System Principles. New York, USA, 1995: 237-250

[2] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection//Proceedings of the 10th Symposium on Network and Distributed System Security. San Diego, USA, 2003: 191-206

[3] Seshadri A, Luk M, Qu N, et al. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes//Proceedings of the 21st ACM Symposium on Operating Systems Principles. Stevenson, USA, 2007: 335-350

[4] Riley R, Jiang X, Xu D. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing//Proceedings of the 11th Recent Advances in Intrusion Detection. Boston, USA, 2008: 1-20

[5] Sharif M, Lee W, Cui W, et al. Secure In-VM monitoring using hardware virtualization//Proceedings of the 16th ACM Conference on Computer and Communications Security. Chicago, USA, 2009: 477-487

[6] Srivastava A, Giffin J. Efficient monitoring of untrusted kernel-mode execution//Proceedings of the 18th Symposium on Network and Distributed System Security. San Diego, USA, 2011

[7] Xiong X, Tian D, Liu P. Practical protection of kernel integrity for commodity OS from untrusted extensions//Proceedings of the 18th Symposium on Network and Distributed System Security. San Diego, USA, 2011

[8] Xiang G, Jin H, Zou D. A comprehensive monitoring framework for virtual computing environment//Proceedings of the 2012 International Conference on Information Networking. Piscataway, USA, 2012: 551-556

[9] Cao Y, Liu J, Miao Q, et al. Osiris: A malware behavior capturing system implemented at virtual machine monitor layer//Proceedings of the 8th International Conference on Computational Intelligence and Security. Piscataway, USA, 2012: 534-8

[10] Li Bo, Wo Tian-Yu, Hu Chun-Ming, et al. Hidden OS objects correlated detection technology based on VMM. Journal of Software, 2013, 24(2): 405-420(in Chinese)

(李博，沃天宇，胡春明等. 基于 VMM 的操作系统隐藏对象关联检测技术. 软件学报，2013，24(2)：405-420)

[11] Qian Zhen-Jiang，Liu Wei，Huang Hao. HybridHP：A verified lightweight approach to provide lifetime kernel integrity surveillance. Chinese Journal of Computers，2012，35(7)：1467-1474(in Chinese)
(钱振江，刘苇，黄皓. HybridHP：一种轻型的内核完整性监控方案及其形式化验证. 计算机学报，2012，35(7)：1467-1474)

[12] Castro M，Costa M，Martin J，et al. Fast byte-granularity software fault isolation//Proceedings of the 22nd ACM Symposium on Operating System Principles. Big Sky，USA，2009：45-58

[13] Mao Y，Chen H，Zhou D，et al. Software fault isolation with API integrity and multi-principal modules//Proceedings of the 23rd ACM Symposium on Operating Systems Principles. Cascais，Portugal，2011：115-128

[14] Petroni N L. Property-Based Integrity Monitoring of Operating System Kernels [Ph. D. dissertation]. University of Maryland，College Park，USA，2008

QIAN Zhen-Jiang，born in 1982，Ph. D.，lecturer. His current research interests include operating system security，formal verification and embedded systems.

LIU Yong-Jun，born in 1981，Ph. D. candidate，lecturer. His current research interests focus on operating system security.

TANG Li，born in 1980，M. S，associate professor. His current research interests focus on operating system security.

YAO Yu-Feng，born in 1981，Ph. D. candidate，lecturer. His current research interests focus on operating system security.

HUANG Hao，born in 1957，Ph. D.，professor，Ph. D. supervisor. His current research interests include system software and information security.

Background

This work is mainly supported by the National Natural Science Foundation of China under Grant No. 61402057，the Natural Science Foundation of Jiangsu Province under Grant No. BK20140418，the China Postdoctoral Science Foundation under Grant No. 2015M571737，the "Six Talents Peak" High-Level Personnel Project of Jiangsu Province under Grant No. 2011-DZXX-035，and the University Natural Science Research Program of Jiangsu Province under Grant No. 12KJB520001. They all aim to develop a secure and trusted microkernel operating system. The group has studied novel security technologies and mathematical formal methods. The group also studied formal specifications and descriptions of the designed microkernel operating system，and strictly verified the correctness of its implementation. In the past years，the group has done a lot of related works including a trusted boot revolution，a verified lightweight approaches to providing lifetime kernel integrity surveillance (HybridHP)，an operating system object semantics model (OSOSM)，trusted verified operating system prototypes (VTOS，VSOS) and their partial formal specifications and verification with Isabelle/HOL theorem prover.

In the paper "Qian Zhen-Jiang，Liu Wei，Huang Hao. HybridHP：A verified lightweight approaches to providing lifetime kernel integrity surveillance"，the authors aim to resolve how to provide lifetime kernel integrity surveillance using virtual technology，but without loss of performance of monitored OS. That paper proposes and implements HybridHP，a lightweight approach to provide lifetime kernel integrity surveillance. HybridHP merges the management domain (e. g. Domain0) and virtual machine monitoring functionality into the monitored system，and provides monitoring services in the way of embedded module.

In this paper，the authors present a HybridHP-based capability isolation method of loaded modules on monolithic kernel OS，named KCapISO. The kernel and loaded modules are isolated from the aspect of the capability，although they run in the same process space. KCapISO ensures that the loaded modules can only modify their own data，but cannot directly call or jump into the kernel for execution in any way. With the monitoring mechanism of HybridHP，KCapISO uses the page directory address register CR3 to mark the running subjects in the system，and performs the access control，so as to achieve the purpose of capability isolation.