一个基于硬件虚拟化的内核完整性监控方法

李 珣 黄 皓
(南京大学计算机科学与技术系软件新技术国家重点实验室 南京 210093)

摘 要 对操作系统内核的攻击就是通过篡改关键数据和改变控制流来危及操作系统的安全。已有的一些方法通过保护代码完整性或控制流完整性来抵御这些攻击,但是这往往只关注于某一个方面而没有给出一个完整的监控方法。通过对内核完整性概念的分析,得出了在实际系统中保证内核完整性需要的条件:保障数据完整性,影响系统功能的关键数据对象只能由指定的代码在特定情况下修改;保障控制流完整性,保护和监控影响代码执行序列改变的所有因素。并采用硬件虚拟化的Xen虚拟机监控器实现对Linux内核的保护和监控。实验结果证明,该方法能够阻止外来攻击和本身漏洞对内核的破坏。

关键词 监控,虚拟机监控器,硬件虚拟化,控制流完整性,数据完整性
中图法分类号 TP316 **文献标识码** A

Approach of Kernel Integrity Monitoring Using Hardware Virtualization

LI Xun HUANG Hao
(National Key Laboratory for Novel Software, Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)

Abstract Kernel-level attacks compromise operating system security by tampering with critical data and control flow in the kernel. Current approaches defend against these attacks by applying code integrity or control flow integrity control methods. However, they focus on only a certain aspect and cannot give a complete integrity monitoring solution. This paper analyzed the kernel integrity principle and got practical requirements to ensure kernel integrity. Critical data objects effect operating system function directly. Only certain code is able to modify critical data objects at certain conditions to ensure data integrity. All factors about code execution sequence are protected and monitored to ensure control flow integrity. Implementation in Xen VMM(Virtual Machine Monitor) using hardware virtualization, or referred to as HVM(Hardware Virtual Machine) is introduced to protect and monitor Linux kernel. Experiments show that the solution can detect and prevent attacks and bugs compromising the kernel.

Keywords Monitor, VMM, HVM, Control flow integrity, Data integrity

1 引言

操作系统安全性是计算机安全不可或缺的一部分。2009年CSI的综述[10]指出,恶意软件感染占该年度安全事件的64%。在操作系统中,当系统内核获得处理器执行权限,除非时间片用尽、触发中断或者系统内核主动放弃执行,否则内核中恶意代码的执行不会被干扰。为了使系统内核不被攻击,引入了监控器的概念。没有硬件支持的传统监控器只能在恶意的内核行为完成之后才能感知到,攻击者甚至可以在攻击后恢复系统环境来避免被探测到,并且内核具有最高的特权级,对内核的成功攻击可以使监控器失效或被绕过。

目前已有多种提高操作系统安全性的研究成果,主要可以分为两类:(1)对内核代码的保护;(2)对内核控制流的保护。第一类通常检查内存被恶意代码篡改后造成的不一致,这类方法往往缺乏实时性;第二类通过保护内核的执行路径来防止恶意代码执行,这类方法可以被旁路。

近几年随着硬件虚拟化的发展,Intel的VMX[5]和AMD的SVM[2]运行状态被引入到处理器中。对于Intel系列处理器,通常虚拟机监控器在VMX根状态下运行,而硬件虚拟化的客户操作系统则运行在VMX非根状态。虚拟机监控器可以通过VM Entry使处理器进入VMX非根状态,客户操作系统通过VM Exit可以使控制权回到VMX根状态。处理器处于VMX根状态时的功能和权限就像没有VMX的系统一样,只是添加了一些新的支持VMX的指令,而处于非根状态的处理器的行为则受到了限制。某些特定的指令、事件或状态会导致客户操作系统触发VM Exit而退出到虚拟机监控器进行特权操作,但客户操作系统与软件本身并不知道自己是否运行在虚拟机上。在硬件虚拟化的帮助下,虚拟机监控器可以收集运行在其上的客户操作系统的实时信息,包括内存、寄存器、指令等,不会造成太高的性能损失。目前已经有

到稿日期:2011-01-14 返修日期:2011-05-15 本文受江苏省高技术项目(BE2008124)和国家自然科学创新群体项目(60721002)资助。
李 珣(1986-),男,硕士生,主要研究方向为虚拟化技术,E-mail:sjtulixun@gmail.com;黄 皓(1957-),男,教授,博士生导师,主要研究方向为计算机信息安全。

利用硬件虚拟化的方法来防止代码和数据不被未经授权的修改，但是如果恶意或有缺陷的代码拥有了授权，那么就有能力篡改受到保护的数据。

为了解决以上问题，保护操作系统内核的完整性，在硬件虚拟化的基础上，提出一个更有效和安全的内核完整性保护方法。考虑了影响完整性的因素，在保证控制流完整性的基础上，结合数据完整性的保护，通过对代码的只读保护和代码执行中的动态跳转的监控来保证控制流完整性，通过监控关键数据只被安全策略允许的代码读取或修改来保证数据完整性。在安全性和性能的测试中，7 个主流的 Linux rootkit 全部被侦测到，其对系统的破坏被阻止。

本文第 2 节介绍了完整性的相关研究成果；第 3 节描述了我们对完整性的定义和满足该定义需要的条件；第 4 节给出了我们的具体实现；第 5 节为安全性和性能的测试结果；最后给出了讨论和结论。

2 相关工作

近年来内核完整性研究有了一定的进展，已经积累了一些新的方法。

Petroni 等[9]提出了基于规约的方法，指出系统运行过程中，不同对象存在着一定的联系。例如正常运行的系统中，运行进程队列的进程应该存在于进程调度链中，如果不一致，则说明该系统中存在隐藏进程的 rootkit 或者相关恶意代码。但是该方法并不具备实时性，且不是所有的恶意行为都能够找到相应的内核对象的不一致关系。

Abadi 等[1]指出了经常被使用的控制流监控的基本思想：构造出一个可以参照的进程执行路径，在进程执行路径中发生跳转时进行相应的检测，判定跳转是否合法，如果不合法则终止进程执行。在控制流中较经典的运用是利用编译的支持，在发生函数调用或者跳转时，通过插入的检测代码在跳转前检测跳转目标是否合法。检测代码中往往通过检查地址或者是否满足特征值来判定，但是在没有数据访问保护的情况下，攻击者一旦知道了防护方法之后，就能够通过修改跳转目标(hook)指向恶意代码并使其满足检测条件绕过该检测机制。Oh 等[8]通过对软件增加校验码，当软件遇到跳转时查询校验码是否合法，如果不合法则终止进程执行。但是该方法同样能够被绕过，进而篡改进程执行路径。Petroni 等[6]也指出在基于控制流的完整性保护中，可以检测操作系统内核中一些关键对象的变化。他们认为大多数的攻击方法都会造成对一定对象永久修改或者对较长时间的修改，因此只要定期检测这些关键对象，并保证内核对象的修改能够跨越检测时间点，就可以检测出是否存在破坏完整性的攻击。这是一种进程完整性和效率的折中方案，他们的结果中有一些破坏操作系统内核的攻击没有办法检测出。Pioneer[13]是控制流监控的一种变形，通过在另一台独立的可信计算系统上运行一个进程作为监控进程，在被监控进程每次发生跳转时，将相关信息和监控进程的信息进行相应的比较，如果跳转不合法，则终止进程运行。但是它不能阻止内核 rootkit 攻击发生。

Garfinkel 等[4]首先采用虚拟机进行入侵检测，其中包括对内核不变量的保护。他们的工作重点是如何用虚拟机自检查来构建入侵检测系统，其细节没有详细讨论。SecVisor[12]是一个采用硬件虚拟化保护内核不受如内核 rootkit 的代码注入攻击的微型管理程序，用户内存、内核代码和内核数据在内核态和用户态赋予不同的访问权限。这个方法没有保护内核数据。NICKLE[11]开发了叫做内存影射的技术，在透明地复制可执行代码到影子内存之前，先对其进行认证。NICKLE 管理客户的物理地址，确定内存访问的类型，分配访问实际内存还是影子内存。任何执行未认证的代码尝试都会在第一时间被挫败，但是不能防止经过认证的代码对数据进行非法的篡改。

以上分析表明，已有的完整性解决方案分别解决了一方面的问题，或者提出一个监控框架。在此基础上，提出了一个较完备的完整性定义，给出了一种监控内核完整性的方法。

3 完整性

本节将给出一个对操作系统内核的运行进行监测的方法，以确定操作系统当前是否处在完整性的状态。为此，我们先给出一系列保障操作系统内核完整性的需求，然后给出操作系统完整性的监测内容和监测方法。

3.1 完整性保障

操作系统是一个服务系统，它的核心功能由内核提供，它的功能或作用就是对系统中产生的事件做出反应，这些事件是硬件产生的中断、用户进程提出的服务请求等。如果操作系统能够按照设计方式对这些事件做出反应，则说明操作系统是完整的。操作系统完整性的破坏主要是由于与操作系统内核相关的对象受到破坏、操作系统的行为受到破坏。操作系统的行为受到破坏的原因可能是操作系统的代码被篡改、操作系统要调用的函数的函数指针被篡改、跳转指令被篡改而改变了执行路径等。

通过分析发现，操作系统完整性的破坏是由于数据和执行路径被破坏。我们认为保护操作系统的完整性，关键有 4 点：(1)保护数据不被恶意篡改；(2)保证执行代码和执行路径不被改变；(3)保证监控器不被被监控操作系统破坏；(4)保证监控行为不被绕过。可以在被监控的操作系统之下插入虚拟机监控器，在虚拟机监控器中实现操作系统完整性的监控。通过控制方法防止操作系统内核的完整性被破坏，并通过监测手段监测操作系统的完整性是否受到破坏。

3.2 具体条件

根据以上分析，列出了确保一个操作系统内核完整性应该进行的保护和监控。

3.2.1 数据完整性的保护

数据的完整性对于操作系统来说是极其重要的。操作系统的服务行为要依赖于相关的数据对象，服务功能效果也部分反映在这些相关的数据对象上，例如进程控制块、页表、文件的索引节点等。把这些数据叫做关键数据。

对操作系统内核完整性的破坏很多情况下都直接修改关

键数据，例如 rootkit[9] 通过插入内核模块获得内核中关键数据 task_struct 的修改能力，可以使任意进程获得 root 权限。因此需要阻止在设计上没有访问某个关键数据的代码修改该关键数据。需要控制的代码不仅包括外来的可装载内核模块，也包括和某个关键数据功能无关的其他内核代码。

另一方面，按照操作系统的设计，在实现和运行中允许修改特定关键数据的代码也只在某些时间段或状态下修改。例如同一段内存复制 memcpy 函数的代码，在打开文件过程中允许修改的关键数据为当前进程 task_struct 的 files 指针及相关数据结构，而在进行内存管理操作过程中允许修改的关键数据为 task_struct 的 mm 指针及相关数据结构。因此，这段代码需要按照不同的运行过程进行监控。当发现处于一个过程中的代码尝试修改该过程中不允许修改的关键数据，说明该系统受到破坏或因为缓冲区溢出等漏洞受到攻击。

要控制内核修改这些关键数据，可以通过内存虚拟化机制，在设计上对可以修改关键数据的代码授权对应的权限，采用触发机制，使得非法对关键数据的修改产生异常陷入虚拟机监控器，从而在虚拟机监控器中阻止该代码对关键数据的修改。

3.2.2 控制流完整性的保护

代码的执行路径和序列必须正确完整。首先不能执行不合法的代码，其次不能按照不合法的顺序执行代码，即使这些代码是合法的。保证控制流完整性可以通过以下措施实现。

(1)对代码的只读保护。如果程序可以动态修改代码，那么获得权限的攻击者也可以，从而使程序执行与设计不符的代码来破坏执行序列。除了动态链接库的加载过程和运行时代码生成，都需要进行保护。

(2)保护动态跳转。动态跳转指代码执行点跳转到由运行时数据决定的或者寄存器中保存的地址继续执行，例如执行 ret 指令会跳转到 ecx 寄存器中保存的地址继续执行。保护动态跳转需要在跳转前验证跳转目标是否是设计上该跳转所有可能的跳转目标之一。在系统实现中，通常采用函数指针表的方式来调用一系列函数，比如 sys_call_table，执行序列根据函数指针表的地址跳转到对应函数继续执行，保护类似的函数指针没有被篡改，也进一步加强了对动态跳转的保护。对这些指针表采取和代码相同的只读保护措施，同时只有设计上允许更新函数指针表的代码才能修改其所在内存。

(3)保护与程序运行路径有关的寄存器。在系统运行中，idtr，gdtr 等控制寄存器中保存的内存地址会间接导致执行路径跳转到不同地址的函数，因此要确保这些寄存器值的修改行为符合规定的策略。

4 实现

本节按照上节所述原理，阐述在虚拟机监控器 Xen[3] 上实现的一个操作系统内核完整性监控系统。

监控系统由 Xen 虚拟机监控器及其中的监控模块和称为 Domain0 的用于管理 Xen 的 2.6.18 Linux 特权操作系统组成。可以在监控系统上安装全虚拟的操作系统 DomainU，通过监控模块监控 DomainU 的完整性。因为可以只暴露 DomainU 给系统外的用户和攻击者，使他们无法感知和访问 Domain0 和 Xen，所以仅仅考虑 DomainU 的安全性问题，即假设 Domain0 和 Xen 是安全的并且不会被攻击。在我们的项目中，在 Xen 上安装了一个内核代码经过修改的 Linux 系统作为 DomainU。对 Linux 内核的修改仅仅为了实现完整性监控，其不影响功能和运行，修改后的 Linux 不限于运行在监控系统中。我们的目标是通过对系统内核的修改和监控来保障 DomainU 的 Linux 内核的完整性。

4.1 架构概述

系统的整体架构如图 1 所示。Domain0 中包含日志模块和策略存储模块。日志模块记录 Xen 通过事件通道传递给 Domain0 的错误信息和策略相关事件的处理结果，来判断是否运行异常或受到攻击。策略存储模块在监控开始前存储策略信息，并在适当的时候将策略传输给 Xen 中的监控模块。虚拟机监控器 Xen 中的监控模块包含 4 个子模块：策略管理模块解析策略存储模块传入的初始策略，并对其它模块提出的策略请求给出相应的响应；关键数据访问处理模块捕获对关键数据的所在内存页面和寄存器的读写和设置属性的尝试，然后调用策略执行模块获取相关策略，根据策略结果模拟执行该尝试并实现该尝试的功能，否则，不允许该尝试并报告错误处理模块；状态点处理模块接受 DomainU 在状态点的主动陷入，首先根据状态点编号请求策略管理模块给出当前状态点需要保护的关键数据和对应操作，然后设置影子页表和虚拟机控制数据结构等，使得策略中应保护的关键数据和操作必然触发 VM Exit 由虚拟机处理；错误处理模块接受关键数据处理模块的 Domian0 中非法动态跳转触发的错误，并把错误报告给 Domain0。

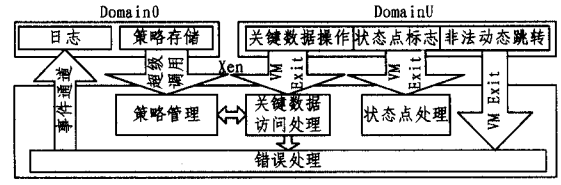

图 1 监控系统架构

4.2 策略存储与管理

因为 Xen 不能直接访问存储设备或文件系统，监控模块需要的完整性保护策略在 Domain0 启动前作为一个文件存储在 Domain0 的文件系统中。在 Domain0 启动后，策略文件被 Domian0 读取并传递给 Xen 中的策略管理模块。特权操作系统 Domain0 通过超级调用与虚拟机监控器 Xen 进行交互。超级调用类似于普通操作系统的系统调用，通过系统调用，由用户态切换进内核态，通过超级调用，由 Domain0 切换进 Xen。增加一项新的超级调用，用于将策略文件传递给 Xen。

首先 Domain0 加载可装载内核模块，该模块把策略文件读入内核的线性地址空间。然后 Domain0 通过超级调用将策略在线性地址中的位置和长度信息通知给监控模块。Xen 有能力访问所有 Domain 的地址空间，因此可以从 Domain0 的线性地址空间中将策略复制到 Xen 中监控模块的空间。最后，监控模块解析策略文本，按照策略需要进行 DomainU 的配置和等待监控模块其他子模块的查询需求。

Xen 使用与所有 Domain 完全隔离的物理和逻辑空间，在策略传递给 Xen 中的监控模块后，不会被恶意的客户操作系统或其中的代码篡改。

4.3 状态点与关键数据保护

在虚拟机监控器中监测当前执行的代码是否有权限更改关键数据，主要解决以下几个问题：

为了让监控模块能够根据当前系统执行的代码实施对应的策略，需要在修改更新安全策略的时候，有一个从客户操作系统陷入到虚拟机监控器的机制。通过修改客户操作系统源码，加入不影响功能的特殊指令与编号解决。客户操作系统不会自动陷入到虚拟机监控器中来，但是 Intel 的 VT 技术提供了当客户操作系统执行了敏感指令，或者触发某些条件时产生类似于操作系统中软中断和异常的 VM Exit 而陷入到虚拟机监控器中的方法。一部分能够触发 VM Exit 的指令在 ring3 级程序中不能被执行，一部分指令功能重要或者触发频繁，因此，需要选择在任何程序中都能够执行，并且对系统运行没有影响的指令作为主动陷入虚拟机监控器的触发指令。在 Xen 中，DomainU 的 CPUID 指令为模拟执行，即在客户机因为执行 CPUID 指令陷入虚拟机监控器后，在输出的寄存器中写入相应的值，并且控制 DomainU 的 eip 指令寄存器跳过该 CPUID 指令，直接执行后续指令。因此，使用 CPUID 指令触发 VM Exit，并定义未被使用的输入参数作为主动陷入的标志。需要更新策略时，内核代码设置陷入参数并执行 CPUID 指令。虚拟机监控器在 VM Exit 处理过程中根据参数进行判断，是正常 CPUID 指令则正常模拟执行，是陷入指令那么更新策略。实现过程中，在虚拟机监控器 Xen 中对标识状态点的 CPUID 和陷入参数引起的 VM Exit 处理流程如图 2 所示。

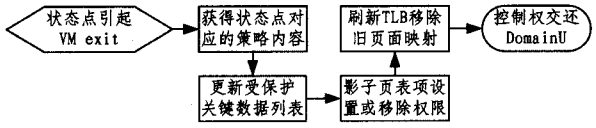

图 2 状态点陷入 Xen 处理流程

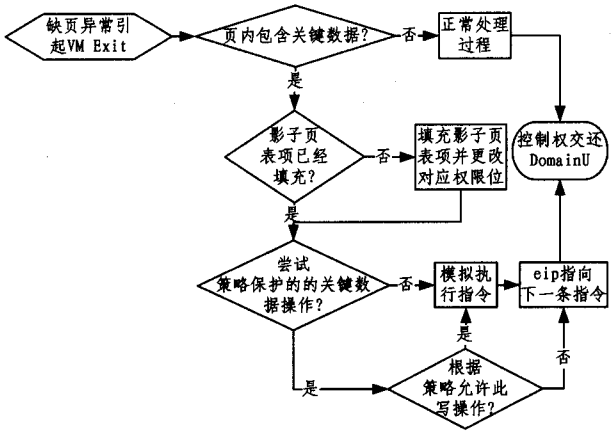

图 3 缺页异常陷入 Xen 处理流程

为了检测对关键数据的读写尝试是否符合安全策略，透明地阻止不符合策略的尝试，虚拟机必须介入每一个不符合安全策略的操作。通过制造被监控系统的页表与其影子页表的不一致来解决。Xen 利用影子页表机制来实现硬件虚拟客户操作系统的内存虚拟，提供客户操作系统线性地址到宿主机物理地址的转换。客户操作系统维护自己的页表，Xen 根据该页表维护 Xen 内部的影子页表。客户操作系统对于由线性地址到机器地址的翻译实际上由影子页表负责。当客户操作系统的页表与影子页表出现不一致时，访问不一致的内存地址会触发缺页异常并引起 VM Exit 陷入 Xen。在状态点陷入时，针对策略指定的关键数据的所在页在影子页表中对应的页表项，如果需要保护关键数据读取，我们设置页表项的存在位为不存在。如果需要保护关键数据写入，我们设置页表项的读写位为只读。经过这样的设置后，在执行被监控的客户操作系统时，如果对策略保护的关键数据的所在页进行了对应的操作，缺页异常引起的 VM Exit 陷入将把执行权限交给 Xen。在 Xen 中就可以进一步检查是否真的改写了不应该改写的关键数据，或是数据对象可改写但正好这个页中有不可改写的数据对象，然后做出相应的反应。对缺页异常陷入的处理流程如图 3 所示。

4.4 动态跳转与执行路径

以下 3 个方面联合起来，能够保证控制流的完整性和执行路径的正确性。

通过初始和永久的策略，在 Xen 的影子页表中将所有代码所在页的页表项设置为只读，以防止恶意或缺陷代码修改内核代码或旁路验证代码。

基于控制流完整性[1]原则的方法，重写了内核代码，对动态跳转点和跳转目标添加了验证代码。当执行至动态跳转点时，验证代码检查跳转目标的特征值是否相符。如果相符，那么同未重写一样照常执行，否则，说明执行的路径与预想的执行路径不相符合，执行上面介绍过的 CPUID 指令并给出相应参数，引起 VM Exit 陷入虚拟机监控器来处理非法的跳转。典型的重写的简单示意如表 1 所列。

表 1 保护控制流代码进行的重写

原始汇编代码	注释	重写汇编代码	注释
call func	调用函数	call func	调用函数
pop %ebx	后续指令	jump 1f	跳过 ID
		.long $ID	本跳转 ID
		1:pop %ebx	后续指令
func:	函数	func:	函数
ret	函数返回	mov %ecx,(%esp)	取得返回地址
		add %esp, $ESPOF	调整 esp 位置
		cmp $ID, $JMPOF(%ecx),	比较跳转前后 ID
		je 1f	一致则正常执行
		mov %eax, $WrJmp	因跳转错误
		mov %ebx, $ID	跳转错误的 ID
		cpuid	陷入处理
		1:jmp %ecx	跳转到返回点

我们用与保护关键数据相同的方法保护函数指针，同时对影响控制流的寄存器，如 idtr，gdtr 等的改写同样可以利用处理器的虚拟化技术。当这些寄存器被修改时，就会触发 VM Exit 陷入到虚拟机监控器中，从而可以得到控制。

5 实验结果

本节描述了在被监控操作系统上进行的一系列实验，给出了在 Linux 上的安全性实验结果。

5.1 安全性

我们部署了本文方法，然后在 Linux 2.6.18 内核上测试了一个如表 2 中 7 种常用 rootkit 的集合：Adore-ng，enyelkm，Mood-nt，Override，SucKIT2，Superkit，Taskigt。分析了这些

rootkit，它们或修改函数指针或寄存器而修改了控制流，或修改了不允许修改的内核对象。通过使用预先定义好的策略，能够检测到这些 rootkit 并阻止它们对内核的破坏，从而使 rootkit 的功能失效。

表 2 Rootkit 的功能和测试结果

Rootkit 名称	控制流修改	内核对象修改	被检测并阻止
Adore-ng	是	是	是
Enyelkm	是	否	是
Mood-nt	是	是	是
Override	是	否	是
SucKIT2	是	否	是
Superkit	是	是	是
Taskigt	否	是	是

5.2 性能

为了测试本文方法对性能的影响，在 Intel Core 2 Quad Q6600(2.66 GHz)的计算机上配置 CentOS 5.3Domain0 和 Linux 2.6.18 作为被监控的 DomainU，分配 2GB 总内存中的 512MB 和 4 个核中的 1 个给 DomainU，其他资源给 Domain0。

本文方法对未受到攻击和破坏的系统性能的主要影响有两点：在切换受保护关键数据和写入与关键数据同一页面的非关键数据时的额外陷入，上述陷入对应的策略分析和影子页表操作。在 Core 架构的处理器上，处理一次典型的缺页异常触发的 VM Exit 与 VM Entry 行为至少需要 1400 个时钟周期。表 3 中测试了几种反映操作系统性能的典型行为在部署我们的监控方法前后的时间。由其中的数据可以看出，额外的监控工作对系统性能造成的损失在 10%以内。此外，对于运行在内核态时间远少于用户态的进程，几乎没有影响性能。这种性能下降所换取的系统的安全性对于安全性要求比较高的用户是可以接受的。

表 3 性能比较

系统行为	原始时间	监控后时间	性能影响
open+write+close 调用	17.55μs	19.18μs	9.2%
压缩	15.784s	16.122s	2.1%
编译	22.187s	22.552s	1.6%

6 讨论

本节讨论我们方法中存在的问题和可能的改进方向。

分散的关键数据。如果在一个页内既包含需要保护的关键数据，又包含其他数据，那么改写页中不保护数据的时候会导致额外的 VM Exit 和相应的切换和处理工作，因此把关键数据和非关键数据分别搜集起来放在不同的页上，能够提高性能。Wang 等[15]提出了一个基于硬件页保护的重新分布受保护的内核函数指针到特定的地址空间的可行方法，可以用来解决这个问题。

被监控系统内的监控器。基于硬件虚拟化的监控器处在被监控系统内能够提供处在虚拟机中相同的安全性[14]，同时能解决监控器难以获得被监控系统语义的问题，把我们的方法迁移到虚拟机内的监控器上能够提高性能。

结束语 本文分析了内核完整性的概念，内核的代码执行行为没有受到破坏和内核的数据没有受到非法的修改。并给出了具体的保护条件：根据内核执行点的不同，保护不同的关键数据不被非法访问和篡改，来保护数据完整性，通过代码只读、保护动态跳转、保护函数指针和保护关键寄存器来保护控制流完整性。在此基础上，给出在 Xen 虚拟机监控器上利用硬件虚拟化机制针对以上各点的监控和保护在 Linux 操作系统内核的实现：修改内核源码插入主动陷入指令来传递给 Xen 状态点，控制影子页表使得代码只读，非法访问和篡改关键数据与函数指针表能够被 Xen 捕获和阻止，重写内核代码在动态跳转前后做检测，会控制影响执行路径的寄存器的修改。实验结果证明，该方法能发现若干主流 rootkit，并阻止它们对系统的破坏，保护和监控对性能的影响较小。

参考文献

[1] Abadi M, Erlingsson M B U, Ligatti J. Control Flow Integrity: Principles, Implementations, and Applications[C]// Proc. of the ACM CCS. 2005

[2] AMD. AMD64 Virtualization Codenamed "Pacifica" Technology: Secure Virtual Machine Architecture Reference Manual[S]. 2005

[3] 石磊，邹德清，金海. Xen 虚拟化技术[M]. 武汉：华中科技大学出版社，2009

[4] Garfinkel T, Rosenblum M. A Virtual Machine Introspection Based Architecture for Intrusion Detection[C]// Proc. of NDSS. 2003

[5] Neiger G, Santoni A, Leung F, et al. Intel Virtualization Technology[J]. Intel Technology Journal, 2006, 10

[6] Petroni N L, Hick M. Automated Detection of Persistent Kernel Control-flow Attacks[C]// Proc. of ACM CCS. 2007

[7] Oh N, Shirvani P P, Mccluskey J. Control-flow Checking by Software Signatures [J]. IEEE Transactions on Reliability, 2002, 51(1): 111-122

[8] Storm P. All-root [EB/OL]. http://packetstormsecurity.org/UNIX/penetration/rootkits/all-root.c, Jan. 2009-15

[9] Petroni N, Fraser T, Walters A, et al. An Architecture for Specification-based Detection of Semantic Integrity Violations in Kernel Dynamic Data[C]// Proc. of USENIX Security. 2006

[10] Richardson R. CSI Computer Crime and Security Survey[R]. Computer Security Institute, 2009

[11] Riley R, Jiang X, Xu D. Guest-transparent Prevention of Kernel Rootkits with VMM-based Memory Shadowing[C]// Proc. of the 11th Symposium on Recent Advances in Intrusion Detection (RAID). 2008

[12] Seshadri A, Luk M, Qu N, et al. SecVisor: A Tiny Hypervisor to Guarantee Lifetime Kernel Code Integrity for Commodity OSes [C]// Proc. of ACM SOSP. 2007

[13] Seshadri A, Luk M, Shi E, et al. Pioneer: Verifying Integrity and Guaranteeing Execution of Code on Legacy Platforms [C]// Proc. of SOSP. 2005

[14] Seshadri M, Lee W, Cui W, et al. Secure In-VM monitoring using Hardware Virtualization[C]// Proc. of ACM CCS. 2009

[15] Wang Z, Jiang X, Cui W, et al. Countering Kernel Rootkits with Lightweight Hook protection[C]// Proc. of ACM CCS. 2009