软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(2):481−494 [doi: 10.13328/j.cnki.jos.004866]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于硬件虚拟化的安全高效内核监控模型*

黄 啸[1,2], 邓 良[1,2], 孙 浩[1,2], 曾庆凯[1,2]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[2](南京大学 计算机科学与技术系,江苏 南京 210023)

通讯作者: 曾庆凯, E-mail: zqk@nju.edu.cn

摘 要: 传统的基于虚拟化内核监控模型存在两个方面的不足:(1) 虚拟机监控器(virtual machine monitor,简称VMM)过于复杂,且存在大量攻击面(attack surface),容易受到攻击;(2) VMM 执行过多虚拟化功能,产生严重的性能损耗.为此,提出了一种基于硬件虚拟化的安全、高效的内核监控模型 HyperNE.HyperNE 舍弃 VMM 中与隔离保护无关的虚拟化功能,允许被监控系统直接执行特权操作,而无需与 VMM 交互;同时,HyperNE 利用硬件虚拟化中的新机制,在保证安全监控软件与被监控系统隔离的前提下,两者之间的控制流切换也无需 VMM 干预.这样,HyperNE 一方面消除了 VMM 的攻击面,有效地削减了监控模型 TCB(trusted computing base);另一方面也避免了虚拟化开销,显著提高了系统运行效率和监控性能.

关键词: 虚拟化;内核监控;特权模式切换;攻击面

中图法分类号: TP316

中文引用格式: 黄啸,邓良,孙浩,曾庆凯.基于硬件虚拟化的安全高效内核监控模型.软件学报,2016,27(2):481−494. http://www.jos.org.cn/1000-9825/4866.htm

英文引用格式: Huang X, Deng L, Sun H, Zeng QK. Secure and efficient kernel monitoring model based on hardware virtualization. Ruan Jian Xue Bao/Journal of Software, 2016,27(2):481−494 (in Chinese). http://www.jos.org.cn/1000-9825/4866.htm

Secure and Efficient Kernel Monitoring Model Based on Hardware Virtualization

HUANG Xiao[1,2], DENG Liang[1,2], SUN Hao[1,2], ZENG Qing-Kai[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

Abstract: Traditional kernel monitoring models based on virtualization have two main drawbacks: 1) Virtual machine monitor (VMM) is vulnerable to attacks due to its non-trivial complexity and considerable attack surface; 2) VMM executes redundant virtualization functionalities, leading to heavy performance loss. To address those issues, this paper proposes a secure and efficient kernel monitoring model, named HyperNE, based on hardware virtualization. In HyperNE, any virtualization functionalities that are isolation and protection unrelated are removed from VMM, and guest OS is allowed to directly conduct privileged operations with no need to interact with VMM. Meanwhile, without sacrificing isolation guarantees, HyperNE utilizes a newly supported virtualization feature to transfer execution between security monitoring applications and guest OS in a controlled manner with no VMM involvement. As a result, HyperNE can not only eliminate the attack surface of VMM and effectively reduce trusted computing base (TCB) size of monitoring model, but also greatly improve system and monitoring performance by avoiding virtualization overheads.

Key words: virtualization; kernel monitoring; privilege mode switch; attack surface

* 基金项目: 国家自然科学基金(61170070, 61572248, 61431008, 61321491); 国家科技支撑计划(2012BAK26B01)

Foundation item: National Natural Science Foundation of China (61170070, 61572248, 61431008, 61321491); National Key Technology Research and Development Program of China (2012BAK26B01)

收稿时间: 2015-02-11; 修改时间: 2015-05-08; 采用时间: 2015-06-12; jos 在线出版时间: 2015-11-12

CNKI 网络优先出版: 2015-11-11 17:04:03, http://www.cnki.net/kcms/detail/11.2560.TP.20151111.1704.005.html

如果操作系统内核受到恶意攻击,或者恶意程序获得与内核相同的运行权限,会使部署在内核中的安全监控失效.相对于内核,虚拟机监控器(virtual machine monitor,简称 VMM)具有更高的运行特权等级,不但能截获并限制被监控的操作系统(guest OS)执行的操作,而且可以将安全监控软件和 Guest OS 隔离,同时还可以借助虚拟机自省(virtual machine introspection,简称 VMI)[1,2]等方法分析 Guest OS 的内部状态.因此,基于虚拟化的内核监控模型一方面能够对 Guest OS 内核进行有效的监控和保护,另一方面能够抵御 Guest OS 对安全监控软件的恶意攻击.

然而,现有基于虚拟化的内核监控模型[3−7]普遍存在如下问题:

(1) 可信计算基(trusted computing base,简称 TCB)的规模过大.

现有研究通常基于一个功能完整的 VMM(如 Xen[8],KVM[9])来构建监控模型,并且通常假定 VMM 作为 TCB 的一部分.然而发,随着虚拟化功能的不断扩展,VMM 自身也变得更加复杂.例如,Xen 4.0 包含约 270K SLOC 代码.另外,管理域 Dom0 内核的代码量则达到千万级.因此,很难保证 VMM 自身代码的正确性.近年来,针对 VMM 的安全漏洞报告[10]不断被发布.

(2) 虚拟化方法额外引入了攻击面(attack surface).

Guest OS 的生命周期中会相当频繁地通过 VM Exit 同 VMM 进行交互,例如 Xen 4.0 中,Guest OS 每秒约发生 600 次 VM Exit[11],而每一次交互都可能是潜在的对 VMM 的攻击[11,12].每一类 VM Exit 代表一类系统事件的发生,且需要陷入到 VMM 中调用相应的处理程序,例如模拟特权指令的执行、处理缺页异常等.因此,Guest OS 可以通过 VM Exit 直接或间接地向 VMM 传递信息,如果这些处理程序存在安全漏洞,则可能会被恶意 Guest OS 通过传递精心构造的数据加以利用并实施攻击.因此,VMM 中存在广泛的攻击面.目前已经有诸多针对 VMM 的攻击案例[13−15],这会导致现有基于虚拟化的内核监控模型失效.需要指出的是,如果消除了 VMM 的攻击面,Guest OS 就不能实施对 VMM 的攻击.

(3) 性能开销严重.

基于虚拟化的内核监控模型的性能损耗主要包括两部分:1) 虚拟化自身的性能损耗,包括 CPU、内存、I/O 设备及中断虚拟化等虚拟化功能产生的性能损耗,以及 Guest OS 与 VMM 之间的 non-root/root 特权切换(包括 VM Exit 和 VM Entry)[16]所产生的性能损耗;2) 监控本身的性能损耗,包括 Guest OS 与安全监控软件之间的切换以及安全监控软件进行相应的分析处理所产生的性能开销.现有研究往往关注对 Guest OS 的监控保护而忽略这两方面的性能损耗,如 out-of-VM 方法[4−7];或仅优化监控方法提升监控效率,如 SIM[3],而忽略虚拟化开销.

为此,本文提出了一种基于硬件虚拟化的安全高效内核监控模型 HyperNE.它具有以下特点:

(1) 高安全性

首先,HyperNE 摒弃 VMM 中全部不必要的虚拟化功能,仅提供对安全监控软件的隔离保护,使得 VMM 可以十分精简,因此,HyperNE 能够有效削减监控模型的 TCB 规模,使其易于验证,更加安全可信;其次,Guest OS 在正常运行过程中无需同 VMM 进行交互,这样,相应 VM Exit 的处理函数可以被去除,进而消除了 VMM 的攻击面,使得 Guest OS 不能对 VMM 实施攻击.此外,HyperNE 将安全监控软件置于 Guest OS 地址空间内,并使用扩展页表(extended page table,简称 EPT)[16]机制将两者隔离.

(2) 高性能

首先,HyperNE 允许 Guest OS 直接访问并管理硬件资源,执行特权操作,因而在正常运行过程中无需陷入 VMM,从而消除 non-root/root 特权切换和虚拟化功能的开销,提升 Guest OS 运行性能;其次,HyperNE 使用 EPTP 切换(EPTP switching)[16]机制,使得 Guest OS 与安全监控软件间的切换无需 VMM 干预,进而提升监控效率.此外, HyperNE 允许安全监控软件在运行过程中处理中断,从而提高系统响应能力.

1 HyperNE 内核监控模型

HyperNE 内核监控模型的整体框架如图 1 所示.其核心特点是:运行在 non-root 模式下的 Guest OS 能够直接访问和管理硬件资源,执行特权操作,而 VMM 没有虚拟化功能,其作用仅仅是为安全监控软件提供一个安全

隔离的运行环境;同时,Guest OS 与安全监控软件之间的切换也无需 VMM 干预.

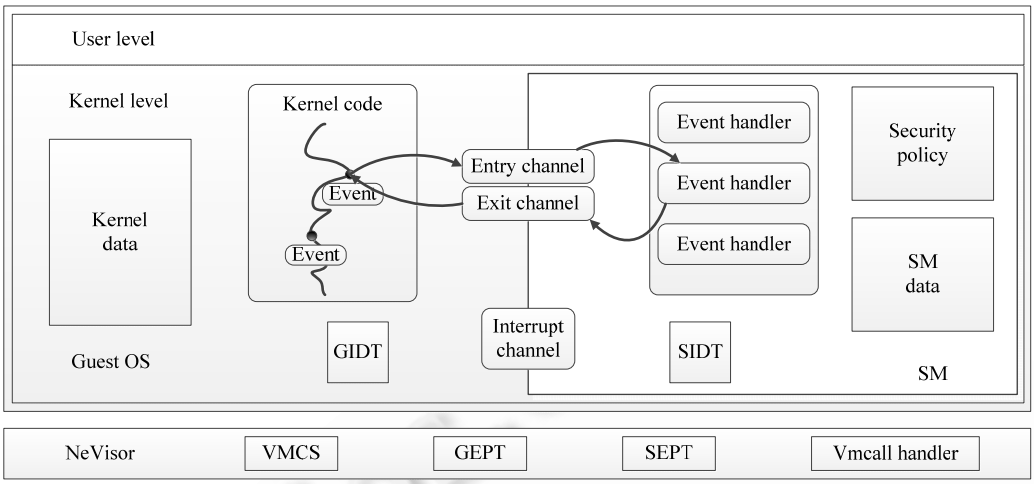


Fig.1 Framework of HyperNE kernel monitoring model

图 1 HyperNE 内核监控模型整体框架

下面具体介绍 HyperNE 中的相关组件.

1.1 相关组件

安全监控软件(SM). SM 置于 Guest OS 地址空间内部,封装了对 Guest OS 内核进行安全监控的相关代码以及运行时所需要的数据段和工作栈,构成一个封闭的执行环境.其中,具体的安全策略依据不同的需求来制定.此外,SM 中还包含中断处理相关组件.HyperNE 确保 SM 和 Guest OS 隔离.

切换通道(switch channel). 依据不同作用,切换通道可以分为出入通道(exit/entry channel)和中断通道(interrupt channel),是 Guest OS 和 SM 之间进行控制切换的唯一接口.Guest OS 内核中用于截获相应事件的钩子(hook)被触发后,控制流会通过出入通道转入 SM 进行相应处理,然后再通过出入通道返回 Guest OS 继续执行.类似地,在 SM 执行过程中产生的中断,需要通过中断通道转入 Guest OS 进行处理.

被监控系统(guest OS). Guest OS 运行在 non-root 模式下不可信的操作系统,是监控的对象.

虚拟机监控器(NeVisor). NeVisor 运行在 root 模式,相当于一个简易的 VMM,其作用是将 SM 同 Guest OS 隔离.NeVisor 允许 Guest OS 在运行时直接访问硬件资源,因此在图 1 中用虚线框表示.

1.2 系统假设与攻击模型

现有研究通常直接将 VMM 归为 TCB 的一部分,相比之下,本文对 VMM 的假设更为合理:由于 VMM 不可避免地存在安全漏洞并且存在攻击面,使得 Guest OS 对 VMM 的攻击成为可能,从而能够破坏整个内核监控模型.由于本文无需 VMM 与 Guest OS 间的交互,因而能够避免此类攻击.

本文假定 Guest OS 能够得到可信启动[17],而在运行时容易受到恶意攻击,并执行恶意代码、访问修改任意的数据.SM 内部具体的安全策略错误或漏洞不在本文考虑范围内,同样,本文也不考虑针对硬件的攻击.

1.3 设计目标

HyperNE 的设计目标是在对 Guest OS 监控的基础上,着力提高整个监控模型的安全性和性能.具体包括以下要求:

- 安全性方面

安全性要求 S1(TCB 小型化及攻击面消除). NeVisor 仅提供对 SM 的隔离保护,不能包含复杂的虚拟化功

能,要保持足够精简;且需要消除 NeVisor 的攻击面,即 Guest OS 在正常执行过程中无需同 NeVisor 其进行交互.

安全性要求 S2(SM 与 guest OS 隔离). HyperNE 需要为 SM 提供一个与 Guest OS 隔离的执行环境,以抵御 Guest OS 的攻击.包括:保护 SM 中代码数据的保密性与完整性;保护 SM 控制流的完整性;同时确保 SM 和 Guest OS 之间的切换只能通过指定的切换通道进行,且仅当放置在 Guest OS 内部的钩子被触发时才会通过切换通道执行相应的处理函数.

- 运行性能方面

性能要求 P1(无虚拟化开销). Guest OS 在正常执行过程中可以直接处理特权操作,直接访问和管理硬件资源,而无需陷入 root 模式中的 NeVisor 进行处理.

性能要求 P2(监控效率提升). SM 和 Guest OS 之间实现高速切换,没有 NeVisor 的干涉;SM 在需要时可以直接访问 Guest OS 中的数据,没有 NeVisor 的干涉.

性能要求 P3(保持系统响应). SM 执行时,不能让 Guest OS 挂起过长时间,需要保持对系统应急事件的响应能力.

需要提到,当前没有任何模型能够同时满足上述全部要求.具体来说,无论是 SIM[3]还是其他 out-of-VM 方法[4−7],均不能满足 S1 和 P1.SIM 能够满足 P2.而 S2 则是虚拟化监控方法的共有特征.

2 系统设计

本节将围绕设计目标继续讨论 HyperNE 各个组件的具体设计.

2.1 总体概括

如图 1 所示,HyperNE 内核监控模型的关键在于:在 NeVisor 中赋予了处于 non-root 模式下的 Guest OS 足够多的运行权限,消除了 Guest OS 运行过程中不必要的 non-root/root 特权切换;使用不同的 EPT,分别控制 Guest OS 和 SM 的运行,以实现 SM 和 Guest OS 之间的隔离;利用 EPTP Switching 机制,可以在 non-root 模式下直接进行 EPT 切换;通过配置 EPT,允许 SM 在运行过程中能够直接访问 Guest OS 的内存区域,而禁止 Guest OS 访问 SM 的内存区域;使用放置在 Guest OS 中的钩子来截获特定事件,并通过切换通道将控制流转移到 SM 中对应的处理函数,从而实现对 Guest OS 的主动监控;此外,为了保持 SM 运行时 Guest OS 的响应能力,HyperNE 将产生的中断传递到 Guest OS 中,并在完成之后返回 SM 继续执行.

2.2 NeVisor设计

在 HyperNE 内核监控模型中,NeVisor 运行在 root 模式,具有最高执行权限.NeVisor 用于提供对 SM 的隔离保护,并限制 Guest OS 的运行.NeVisor 无需为 Guest OS 执行复杂的虚拟化功能,相反,允许 Guest OS 直接控制并管理系统资源以及执行特权操作.这也使得 NeVisor 区别于 Xen,KVM 等通用 VMM.Guest OS 在正常执行过程中,无需产生 VM Exit 以请求 NeVisor 服务.如果出现 VM Exit,则意味着 Guest OS 违反了某些安全限制,例如 Guest OS 非法访问了 SM 所在的内存区域.

传统的 VMM 主要为 Guest OS 提供 CPU、内存、I/O 外设以及中断虚拟化等虚拟化功能.这里也将从以下 3 个角度阐述 NeVisor 的设计原则.

(1) NeVisor 消除处理器执行特权操作产生的 VM Exit.

通常,Guest OS 执行某些特权操作时,处理器产生 VM Exit 陷入 VMM 并进行相应模拟或者其他处理.在 NeVisor 中,Guest OS 直接执行特权操作,类似于非虚拟化环境,因此这些 VM Exit 是不必要的,应当消除.

在 Intel VT[16]技术中,产生 VM Exit 的特权操作可分为 3 类:执行有条件产生 VM Exit 的指令、执行无条件产生 VM Exit 的指令和非指令事件导致的 VM Exit.

对于有条件产生 VM Exit 的指令和诸如中断、异常、NMI 等事件,通过配置 VMCS[16]中相应字段,使 Guest OS 在 non-root 模式直接处理,无需产生 VM Exit.当然,对于 HyperNE 禁止的 Guest OS 操作,例如第 2.7 节提到的对相关 LBR 寄存器的写保护,仍需在 VMCS 中加以限制.

在 Non-root 模式下执行 CPUID 指令会无条件产生 VM Exit.HyperNE 采用 NoHype[11]的处理方式:在 Guest OS 启动时执行 CPUID 指令,并将结果保存,修改内核或者应用程序,使其以调用的形式直接获得 CPUID 信息.

其他一些无条件产生 VM Exit 的指令或事件,Guest OS 在正常执行过程中并不会使用,可以不考虑.

(2) NeVisor 预分配所有的物理内存.

通常,在虚拟化环境中由 VMM 管理真实的物理内存,并对 Guest OS 采取请求分页的措施:当 Guest OS 访问一个尚未分配的内存页面时,会产生 EPT_VIOLATION 类型的 VM Exit,然后交由 VMM 为其分配.

在 HyperNE 模型中,NeVisor 无需进行内存虚拟化,而交由 Guest OS 直接管理物理内存.由于 Guest OS 自身页表只能控制从客户虚拟地址(guest virtual address,简称 GVA)到客户物理地址(guest physical address,简称 GPA)的映射,而无法直接访问真实的宿主物理内存(host physical address,简称 HPA),为此,NeVisor 采用了预分配的策略[11],将全部的物理内存预先分配给 Guest OS,即在 NeVisor 初始化 EPT 时就包含对整个 HPA 空间的映射,这样使得 Guest OS 可以访问和管理全部物理内存.

(3) NeVisor 允许 Guest OS 直接访问外设.

X86 架构通过 I/O 指令和内存映射 I/O 两种方式向外设发送 I/O 命令.通常,VMM 会通过 VM Exit 截获 I/O 命令,并进行相应的模拟操作.

NeVisor 允许 Guest OS 直接访问外设.通过配置 VMCS 中的 I/O bitmap[16],使得 Guest OS 能够在 non-root 模式下直接执行 I/O 指令.同时,NeVisor 对物理内存的预分配也保证了 Guest OS 可直接以内存映射 I/O 方式访问外设,均无需 NeVisor 的干预.

为了防止 NeVisor,SM 等组件所属物理内存被 DMA 访问和修改,HyperNE 利用 IOMMU[18]来进行 DMA 保护.同时,禁止 Guest OS 访问 PCI/PCIe 配置空间,以防分配给设备 I/O 内存的基地址被修改.

2.3 内存访问权限

HyperNE 实现 SM 与 Guest OS 隔离的关键在于使用两个不同的 EPT 来控制 GPA 对 HPA 的访问,使 Guest OS 和 SM 在运行时具有不同的内存视图:当 Guest OS 正常运行时,NeVisor 中控制内存访问的 EPT 称之为 Guest EPT(简称 GEPT);当运行切换到 SM 中时,使用另外一个 EPT,称为 Secure EPT(简称 SEPT).GEPT 和 SEPT 均能映射整个 HPA 空间,两者的区别在于,对不同组件赋予不同的内存访问权限,具体设置如图 2 所示.

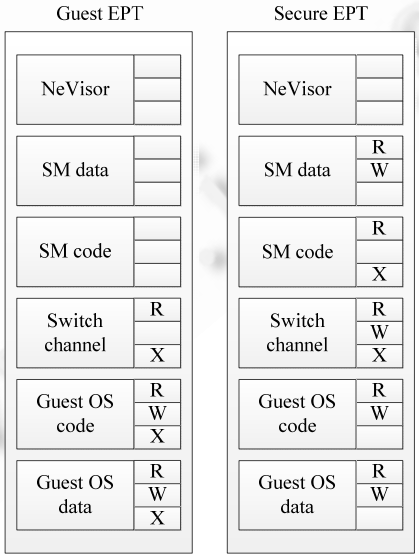


Fig.2 Memory view of guest OS and SM

图 2 Guest OS 和 SM 的内存视图

HyperNE 设置各个组件内存访问权限主要依据如下原则:

(1) 当 SM 运行时,SEPT 中的 SM 代码和数据部分分别具有可执行和读写权限.由于此时 Guest OS 部分不可执行,因此在 SEPT 中,Guest OS 的代码和数据部分均无可执行权限.同时,SEPT 中为 Guest OS 设置读写权限,使得 SM 中的代码能够直接访问 Guest OS 部分.

(2) 当 Guest OS 运行时,Guest OS 对 SM 内容没有读、写或执行权限,GEPT 中没有任何权限设置.此外,内存具体的用途由 Guest OS 页表控制,因此在 GEPT 中,对 Guest OS 代码、数据以及其他空闲内存均赋予读、写和执行权限.当然,为了保护设置在 Guest OS 内核中的钩子,HyperNE 对包含钩子的内存区域实行写保护,这部分设置没有在图 2 中标识.

(3) 切换通道用于实现 GEPT 和 SEPT 间的切换,是唯一在 Guest OS 和 SM 中均具有可执行权限的组件.需要指出的是,在 GEPT 中切换通道没有写权限,以防止被 Guest OS 恶意修改.

(4) 对 NeVisor 自身的隔离.前面提到,NeVisor 中的 EPT 能够映射到全部 HPA.为了将 NeVisor 与 Guest OS 隔离,NeVisor 自身所在内存区域对 Guest OS 和 SM 均不可见.因此,GEPT 和 SEPT 中均没有设置任何访问权限.

2.4 EPT切换

为了使 Guest OS 和 SM 在运行时具有不同的内存访问权限,需要在各自运行时分别使用 GEPT 和 SEPT 控制内存访问.因此,Guest OS 和 SM 之间进行切换的同时,也需要进行 GEPT 和 SEPT 的切换.

在 Intel VT 技术中,EPT 是由 VMM 创建并控制的,VMCS 中的扩展页表指针(extended-page-table pointer,简称 EPTP)字段包含 EPT 分页结构基地址等信息.切换 EPT 需要修改 EPTP,然而 non-root 模式下的 Guest OS 无法直接访问 EPTP,需要陷入到 root 模式中由 NeVisor 进行处理.这与 HyperNE 的性能要求 P2 相违背,因此需要采用新的机制来绕过 root 模式的干预.

Intel VT 技术在 Haswell[16]架构的处理器中新增加了对 VM Functions 机制的支持,所谓 VM function,是在 non-root 模式下通过 VMFUNC 指令能够直接执行的操作,不会产生 VM Exit.RAX 寄存器指定要调用 VM function 的编号,其中,编号为 0 的 VM Function 称为 EPTP switching.EPTP switching 允许在 non-root 模式下为 EPTP 加载一个新值,从而建立另外一个 EPT 分页结构.EPTP 的候选值需要在 root 模式中提前配置好,存放在 EPTP 列表(EPTP list)结构中.

HyperNE 借助 EPTP switching 来实现 GEPT 和 SEPT 之间的切换:首先,需要在 NeVisor 分别构建完整的 GEPT 和 SEPT 分页结构,并将对应的 EPTP 值 GUEST_EPTP 和 SECURE_EPTP 存放在 EPTP list 中;随后,在 non-root 模式下执行 VMFUNC 指令,RCX 寄存器用于选择 EPTP list 表项.EPT 切换由切换通道中的代码完成,将在下一节具体介绍.

2.5 切换通道

切换通道用于实现 Guest OS 和 SM 之间的转移,是唯一在 GEPT 和 SEPT 中均具有可执行权限的组件;同时,切换通道在 GEPT 中写保护,确保 Guest OS 不能恶意篡改切换通道.切换通道可以分为出入通道和中断通道.本节只讨论出入通道.

出入通道中所执行代码如图 3 所示,其中,入口通道执行以下 3 类操作.

(1) 保存 Guest OS 在切换前的寄存器信息,用于返回时恢复;

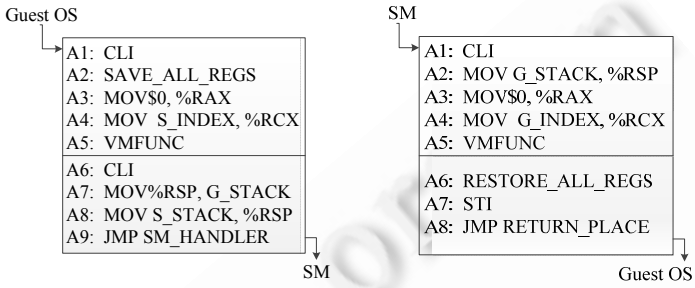
(2) 执行 EPT 切换操作,由 GEPT 切换为 SEPT.使用 VMFUNC 指令完成,其中,RAX=0 为 EPTP switching,RCX 存放所需 EPTP 在 EPTP list 中的索引;

(3) 配置新的执行环境,主要是设置 SM 的运行工作栈 S_STACK.

随后,入口通道转入 SM 对应的处理程序继续执行.其中,在具体执行安全策略之前,应当检查调用的合法性(第 2.7 节).在此之前,应当处于关中断状态,确保执行过程的原子性.因此,入口通道首先执行关中断操作以确保控制流不会转到其他地方.如果攻击者直接跳转到 A2 处执行,使得在开中断条件下检查调用合法性,可能会导

致不可预知的行为.可以在 A6 处再次执行关中断指令来解决这一问题.

同样地,出口通道用于从 SM 返回 Guest OS 继续执行,主要工作是恢复 Guest OS 在转入 SM 运行之前的工作栈 G_STACK,执行 EPT 切换操作,然后恢复 Guest OS 寄存器上下文信息.为避免中断干扰,出口通道同样关中断执行,并在结束时开中断.



(a) 入口通道 (b) 出口通道

Fig.3 Switch channel

图 3 切换通道

2.6 中断响应

前面提到,在 HyperNE 内核监控模型中,Guest OS 可以直接处理中断而无需 NeVisor 干预.然而,当在 SM 执行期间产生中断时,由于外部 Guest OS 的中断处理程序没有可执行权限而无法直接处理.为了满足性能要求 P3,在 SM 执行时,HyperNE 必须能够响应中断.

在 x86 架构中,中断的处理程序在中断描述符表(interrupt descriptor table,简称 IDT)中设置,IDT 基地址存放在 IDTR(interrupt descriptor table register)中.当外部中断发生时,处理器通过 IDTR 定位 IDT,并使用中断向量(interrupt vector)索引 IDT,然后跳转到 IDT 表项所指定的中断处理程序执行.

因此,当 Guest OS 运行时,使用原有的 IDT,称为 Guest IDT(简称 GIDT),正常处理产生的中断.当 SM 在执行时,使用自己维护的 Secure IDT(简称 SIDT).在 SM 运行过程中产生的中断会被 SIDT 的中断处理程序截获,然后再将它们转发给 GIDT 中对应的中断处理程序执行.

具体来说,SIDT 的中断处理程序包含以下操作:

(1) 保存并清空 SM 运行时的寄存器信息,以免泄露给 Guest OS.

(2) 构造 Guest OS 处理中断的栈帧.当中断产生时,处理器会自动将 SS,RSP,RFLAGS,CS,RIP 寄存器值保存在当前 S_STACK 上.因此,HyperNE 在 G_STACK 上构造同样的信息.相关寄存器的值应设置为之前在 Guest OS 中运行时的原始值.值得注意的是,为了确保中断能够正常返回到 SM 中继续执行,RIP 的值应设置为中断返回通道的入口地址.

(3) 通过中断入口通道,从 SM 切换到 Guest OS 中.中断通道的功能和操作与前面出入通道类似,这里不再赘述.

(4) 从 GIDT 对应的中断处理程序继续执行.

中断执行完毕后,会通过中断返回通道切换到 SM 中,此时仅需要恢复 SM 被中断的上下文即可继续执行.

由于中断栈帧存放在 G_STACK 上,Guest OS 可以任意修改.如果返回地址 RIP 被修改,控制流就可能不会返回到 SM 中.为了防止这一情况的发生,HyperNE 使用 Intel VT 提供的抢占计时器(VMX-preemption timer)机制[16]来解决.该机制允许我们配置一段时间长度,当在 non-root 模式下运行这段时间后,会无条件产生 VM Exit 陷入 root 模式.具体做法是:SM 中断处理程序在切换 EPT 之前,通过超级调用(第 3.2.1 节)在 NeVisor 中配置抢占计时器.如果在这段时间内中断正常返回,则再次在 SM 中通过超级调用清除该计时器;如果没有返回,那么 NeVisor 通过检查 G_STACK 中中断栈帧的内容来判断是否被恶意篡改,或者强制将控制流切换回 SM 继续执

行,之后再恢复到 Guest OS 中.

2.7 SM模块

SM 中的处理函数具体执行相应安全策略,并依据分析结果采取相应的响应措施.

为了防止 Guest OS 对 SM 的任意调用,当控制流通过切换通道转入 SM 时,应当立即检查调用的合法性,即确保对切换通道的调用是由放置在内核中相应的钩子发起的.这是由于切换通道对 Guest OS 可见,Guest OS 可以从任意地方调用切换通道.HyperNE 采用 SIM[3]的方法来解决这一问题,即利用 LBR(last branch recording)来记录执行过程中最近遇到的分支指令的源地址和目的地址,并在调用检查程序中予以检查.

SM 的处理函数由具体的安全策略来指定.例如,检测内核 Rootkits 的可能操作是监控内核模块加载这一事件,并扫描分析待加载的模块代码;或者在进程创建或进程切换时,扫描分析 Guest OS 的进程链表以及运行队列.具体的监控要求和安全策略设计已超出本文讨论范围,这里不再赘述.

需要强调的是,SM 中处理函数的控制流不应当依赖 Guest OS 所控制的控制流转移结构,以免控制流被恶意篡改.由于 Guest OS 不可信,SM 处理函数应当是完全封闭的.如果其执行代码依赖于 Guest OS 的功能,可以利用 Inktag[19]等工作的方法加以验证和保护.

3 系统实现

本文以 Ubuntu 12.04 64-bit LTS,内核版本 Linux 3.4.61 为被监控的 Guest OS 实现了 HyperNE 内核监控模型的原型系统.

3.1 NeVisor启动过程

NeVisor 采用了 Intel VT 技术,其启动过程是整个虚拟化环境的配置过程,包括:

(1) 构造 NeVisor 所需的运行环境,包括堆栈和其他所需内存空间.

(2) 执行 VMXON 指令进入 VMX 操作模式,开启硬件虚拟化机制.

(3) 分配 VMCS 区域(VMCS region)并设置各字段,其中,针对 non-root 模式的执行控制域具体设置见表 1.

(4) 构建 Guest OS 运行所需的全部 GEPT 分页结构,包含整个 HPA 空间.为简单起见,本文采用恒等映射的方式进行构建.此外,NeVisor 采用大页面(1GB 或者 2MB)的方式,以提高 TLB 命中率.

(5) 使用 VMLAUNCH 指令切换到 non-root 模式执行 Guest OS.由于 VMCS 的配置,Guest OS 在运行过程中无需请求 NeVisor 的服务.

Table 1 VMCS configurations in NeVisor startup

表 1 NeVisor 启动时 VMCS 的配置

VMCS 设置	作用
MSR bitmap	禁止 non-root 模式修改 LBR 相关的 MSR 寄存器
I/O bitmap	禁止 non-root 模式访问 PCI/PCIe 配置空间
Enable EPT	开启 EPT 机制
Enable VM functions	开启 VM functions 机制

3.2 SM加载与配置

为了使 HyperNE 内核监控模型能够有效地运行,需要加载并正确配置 SM 和切换通道.HyperNE 采用内核模块(LKM)的方式将 SM 装载到 Guest OS 中.由于假定 Guest OS 已经执行了可信启动,可以认为此时的 Guest OS 处于可信状态.该 LKM 的初始化函数执行的主要任务包括:分配并设置运行所需的内存空间,通知 NeVisor 创建 SEPT 分页结构并进行相应访问权限设置,设置内核钩子并创建切换通道和调用检查程序等.

3.2.1 超级系统调用

HyperNE 采用超级调用的方式使 SM 与 NeVisor 进行交互.目前,NeVisor 中为 SM 提供了 3 个超级系统调

用.所谓超级系统调用,实际上是利用了 VMCALL 指令.在 non-root 模式下执行 VMCALL 指令时,会主动产生 VM Exit 陷入 root 模式,以请求 VMM 的服务,VMCALL 指令使用寄存器传递参数信息.NeVisor 提供的 3 个超级系统调用如下:

1) HYPERCALL_CREATE:用于创建 SEPT 分页结构.
2) HYPERCALL_SET:用于设置指定内存区域的访问权限.
3) HYPERCALL_CHECK:用于通知 NeVisor 执行某些检查工作.

当 SM 调用上述的超级系统调用时,处理器会陷入 NeVisor 中,NeVisor 则会根据具体的参数信息对该超级系统调用进行处理.

3.2.2 创建 SEPT 与内存访问权限设置

在 SM 初始化函数中,通过执行 HYPERCALL_CREATE 超级系统调用来通知 NeVisor 进行 SEPT 的创建.NeVisor 初始创建的 SEPT 实际上是 GEPT 的一份拷贝,不同之处在于取消了所有页表项的可执行权限.同时设置 VMCS 以启用 EPTP switching 机制,并分别将 GUEST_EPTP 和 SECURE_EPTP 存放在 EPTP list 的前两项中.

随后,SM 会陆续通过 HYPERCALL_SET 超级系统调用,通知 NeVisor 进行各组件的内存访问权限的设置.NeVisor 会根据图 2 列出的访问权限分别在 GEPT 和 SEPT 中进行对应设置.

3.2.3 SM 设置工作

为使整个监控架构能够有效地运转,SM 的初始化函数还需要在内核中增加所需钩子以截获相应系统事件,从而在内核执行该事件时将控制流转移到 SM 中.HyperNE 不限制添加钩子的具体方式.同时,为每个内核钩子设置对应的出入通道,包括:在入口通道中设置 SM 处理程序入口地址,确保能够进行相应的处理;在出口通道设置返回地址,以确保能够正常返回 Guest OS 继续执行.此外,还需要记录钩子和切换通道所在地址,确保能够正确检查调用合法性.

4 系统评价

本节将从 HyperNE 内核监控模型如何满足全部安全性要求和性能要求两个方面加以详细论述.

4.1 安全性要求分析

HyperNE 内核监控模型能够满足第 1 节提出的全部安全性要求.

(1) HyperNE 满足安全性要求 S1(TCB 小型化及攻击面消除)的分析

HyperNE 内核监控模型的设计理念就是底层的 NeVisor 仅仅注重隔离保护机制,因此在设计之初就剔除了通用虚拟机监控器所包含的 CPU、内存、I/O 设备及中断等复杂虚拟化功能;相反,NeVisor 利用 Guest OS 来直接管理系统资源,因此能够保证 NeVisor 自身足够简洁和轻量.在本文实现的原型系统中,NeVisor 包含不到 2 000 行的 C 语言代码以及 200 行左右的汇编语言代码,这使其自身更易于可信验证.

与此同时,Guest OS 正常运行过程可以直接处理特权操作,而无需产生 VM Exit 陷入 NeVisor 进行处理,NeVisor 中也无需提供相应 VM Exit 的处理函数,避免了 Guest OS 通过精心构造的数据来触发 NeVisor 中可能存在的漏洞,从而避免造成数据泄漏或者特权提升攻击的风险.Guest OS 与 NeVisor 之间没有交互的需求,从而消除了虚拟化引入的攻击面.此外,由于 EPT 中的设置,Guest OS 不能直接访问或破坏 NeVisor 的任何代码和数据,保证了 NeVisor 自身的完整性.

综上所述,相对于其他基于虚拟化的内核监控模型,HyperNE 有效削减了 TCB 规模,消除了虚拟化额外引入的攻击面,使自身更加安全、可信.

(2) HyperNE 满足安全性要求 S2(SM 同 Guest OS 隔离)的分析

首先,HyperNE 能够保证 SM 中代码和数据的保密性与完整性.具体来说:① 由于 GEPT 的设置,Guest OS 不能访问和修改 SM 中的任何内容,任何形式的非法访问均会产生 EPT_VIOLATION 类型的 VM Exit 而被 NeVisor 截获;② 当 SM 切换到 Guest OS 时,所使用的寄存器内容会在转出前保存、清空,Guest OS 也无法读取并修改 SM 的寄存器内容,因此能够保证 SM 内存和寄存器的保密性和完整性.

其次,HyperNE 能够保证 SM 控制流的完整性.具体来说:① 前述对 SM 内存完整性的分析,保证了 Guest OS 无法破坏 SM 执行代码和堆栈;② 设计时要求 SM 不依赖于能够被 Guest OS 所控制的控制流转移结构,确保 Guest OS 不会直接影响 SM 运行时的控制流;③ 在 SEPT 控制时,Guest OS 不具有可执行权限,确保 SM 在运行过程中 Guest OS 不会直接获得执行权限;④ HyperNE 对中断的处理(第 2.6 节)确保了 SM 的控制流不会受到影响,因此 Guest OS 无法破坏 SM 的控制流完整性.

最后,HyperNE 能够保证通过指定的切换通道进行切换.具体来说:① HyperNE 在 GEPT 中对切换通道进行写保护,确保 Guest OS 无法篡改切换通道内容;② Guest OS 无法自行执行 VMFUNC 指令切换到 SM 中,这是由于一旦执行该指令,将会立即切换到 SEPT 中,并产生 EPT_VIOLATION 类型的 VM Exit 而被 NeVisor 截获,因此,切换通道是 SM 和 Guest OS 之间切换的唯一途径;③ HyperNE 为每个内核钩子设置对应的切换通道,并在切换到 SM 后立即进行调用合法性检查,确保对 SM 的调用只能是在对应的内核钩子被触发时.

综上所述,HyperNE 能够和其他虚拟化方法一样,将 SM 与 Guest OS 隔离并抵御 Guest OS 对 SM 的攻击.

(3) 可能存在的攻击

在 HyperNE 内核监控模型中,对 Guest OS 内核实施主动监控,依赖于放置在 Guest OS 内部的钩子、切换通道以及 SM 等.由于 Guest OS 自身的内核页表不受 HyperNE 控制,恶意的 Guest OS 可以通过修改内核页表的映射关系,将原先映射到钩子、切换通道或者 SM 所在页面的虚拟地址映射到其他的客户物理地址,这样就会绕过所设置的这些组件,导致监控机制的失效.需要指出:当进入到 SM 之后,在 SM 中可以构建自己的客户页表结构,即使 Guest OS 恶意篡改 SM 映射所在内存位置的内核页表,也不会产生安全问题.

解决上述问题的关键在于对 Guest OS 的内核页表加以控制,使得对 Guest OS 内核页表的修改能够被检测到或者加以验证.因此,目前可能的解决方法有两种,能够有效提升对 Guest OS 内核页表实施攻击的难度:

1) 在 NeVisor 中,对 Guest OS 内核页表中钩子、切换通道等所在页面的映射进行检测,确保其客户物理地址与原先注册时的相同.具体做法是:利用抢占计时器机制(VMX-preemption timer)[16],通过配置该计时器,定期强制 Guest OS 陷入 NeVisor 中,此时可以实施对 Guest OS 内核页表映射的检查;并且可以通过配置不同的时间长度,增加攻击者在间隔期内消除修改痕迹的难度.
2) 验证对 Guest OS 内核页表的修改,阻止恶意篡改.HyperSafe[20]提出的对页表的自保护方法可以应用到本文环境中,使得对 Guest OS 内核页表的更新需要进行合法性验证.值得注意的是:要通过此类攻击达到预期效果,实施起来也较为困难.这是由于通常 Guest OS 内核页表的映射较为固定和稳定,修改内核页表可能会使相应组件所在页面的内核功能失效,甚至导致内核崩溃.

4.2 性能要求分析

HyperNE 内核监控模型能够满足第 1 节提出的全部性能要求.

(1) HyperNE 满足性能要求 P1(无虚拟化开销)的分析

NeVisor 允许 Guest OS 直接管理物理内存、控制和访问设备,并且赋予 Guest OS 直接执行特权指令及处理特权事件的权限.Guest OS 在正常执行过程中,不会因为执行特权操作而产生 VM Exit 陷入 NeVisor,因而消除了 Guest OS 与 NeVisor 之间的 non-root/root 特权切换,也无需 NeVisor 执行相应虚拟化功能.

综上所述,HyperNE 没有虚拟化开销.

(2) HyperNE 满足性能要求 P2(监控效率提升)的分析

HyperNE 利用 Intel 硬件虚拟化技术提供的 EPTP Switching 机制,使得能够直接在 non-root 模式下完成 EPT 切换,而无需陷入 root 模式进行切换操作.此外,SEPT 中 Guest OS 部分具有读写权限,因此,SM 在执行过程中可以直接访问 Guest OS,获取所需数据,而无需 NeVisor 干预.

综上所述,类似于 SIM,HyperNE 能够确保对内核事件监控的快速响应和处理,从而提升监控效率.

(3) HyperNE 满足性能要求 P3(保持系统响应)的分析

由于 SM 在运行时自主控制 IDT,能够对系统产生的中断予以响应,并切换到 Guest OS 对应的中断处理函数进行执行,从而避免了 SM 占用处理器过长时间,使得系统应急事件得到及时响应.

5 性能测试

本节测试 HyperNE 内核监控模型的监控效率以及整个系统运行效率,并将测试结果与现有内核监控模型对比.测试使用的硬件环境:CPU 为 Intel Core i7 4770k,内存为 4GB DDR3,1TB SATA 硬盘.所涉及到的操作系统均为 Ubuntu 12.04 64-bit LTS,内核版本 Linux 3.4.61.由于目前实现的 HyperNE 原型只支持单核,所以所有测试均在处理器开启单核模式下进行.

5.1 监控效率

我们将使用微测量基准来测量 HyperNE 内核监控模型的监控效率,包括 Guest OS 与 SM 之间的切换以及 SM 进行相应处理所产生的性能损耗,并且与 SIM[3]以及"out-of-VM"[4−7]方法进行对比.

5.1.1 切换机制比较

HyperNE 中使用 EPTP Switching 机制完成从 Guest OS 切换到 SM.SIM 使用 Intel 提供的 CR3-TARGET 机制[16]来完成影子页表(shadow page table)[8]的切换,而其他 out-of-VM 方法则通过 VM Exit 实现 non-root 模式和 root 模式间的转换,并以此实现切换.这 3 种不同切换机制在本文实验中的具体时间开销见表 2.

Table 2 Overhead of three different switching mechanism (μs)

表 2 3 种不同切换机制的开销 (μs)

EPTP switching	CR3-Target	Non-Root/Root
0.042	0.045	0.429

从表 2 中可以看出:由于均不需要进行比较费时的 non-root/root 特权切换,HyperNE 和 SIM 的切换机制能够达到很高的效率.值得注意的是,HyperNE 架构中 Guest OS 在执行过程中无需产生 VM Exit 以请求 NeVisor 服务,因此消除了 non-root/root 特权切换及虚拟化功能的开销;而 SIM 中,VMM 仍需频繁干预 Guest OS 的执行.

5.1.2 监控调用开销

本文以 *execve*()系统调用为例,测试 HyperNE 监控效率.应用程序通过执行 *execve*()系统调用来执行自己的代码,内核中,*execve*()对应的处理函数为 *sys_execve*(),该函数的第 1 个参数为可执行文件路径名的地址,存放在 RDI 寄存器中.本文修改内核系统调用表 syscall table,使得控制流首先进入 SM 中进行分析.与 SIM[3]类似,本文采取的安全策略是提取 *execve*()的可执行文件名,并且与 SM 中维护的可执行文件白名单进行对比,以确定是否为允许的程序镜像.测试结果见表 3.

Table 3 *execve*() monitoring performance results (μs)

表 3 对 *execve*()监控性能测试结果 (μs)

HyperNE	SIM	Out-of-VM
3.489	3.527	10.156

从实验结果可知:和 SIM 一样,由于 HyperNE 在切换到 SM 过程中无需 non-root/root 特权切换,因此 HyperNE 监控模型具有很高的监控效率,甚至能够稍稍领先 SIM.

5.2 整体运行效率

传统的虚拟化监控模型往往忽略虚拟化功能产生的额外开销,因此无法真正评估 Guest OS 所获得的真实性能.为了评估 HyperNE 的性能,本文使用 lmbench 测试用例集[21]进行测试.通过与直接运行在硬件上 Linux 系统(native Linux)以及 SIM 监控模型进行对比来评估 HyperNE 的性能.具体实验结果见表 4 和表 5.

表 4 显示了 lmbench 基准测量操作系统中常见的系统调用以及实际内存操作的执行时间.在表 4 上半部分,null call 测量应用程序调用一次简单的系统调用(getpid)消耗的时间,而 open/close,file create 和 file delete 对应文件系统相关的系统调用.这些系统调用均不需要 root 模式的干预,能够在 Guest OS 内直接执行,因此,HyperNE 和 SIM 的测试结果均能达到或接近 native Linux 的性能.在表 4 下半部分,page fault 测量缺页中断处理执行时间,Mmap Latency 测量应用程序调用 *mmap*()系统调用映射一块内存所消耗的时间.fork 和 exec 测量

进程创建以及执行可执行文件的时间,在此过程中,需要频繁的内存映射操作.由实验结果可见,HyperNE 相对于 SIM 有明显的性能提升.这是由于这些测试均涉及到 Guest OS 页表更新的操作,由于 SIM 中依赖复杂影子页表机制来进行内存虚拟化操作,需要陷入到 root 模式更新对应的影子页表,从而产生明显的性能开销.而在 HyperNE 中,内存的管理和分配完全由 Guest OS 控制,无需 root 模式干预.HyperNE 的主要开销在于:当 TLB 缺失时,需要额外一轮页表地址转换,但使用 EPT 大页面能够有效提升 TLB 命中率.

Table 4 Results of system calls and memory performance benchmarks in lmbench (μs)

表 4 lmbench 中不同系统调用和内存性能测试基准结果 (μs)

Benchmarks	native Linux	HyperNE	SIM	HyperNE 开销(%)	SIM 开销(%)
null call	0.03	0.03	0.03	0	0
open/close	0.56	0.56	0.59	0	5.36
0k file create	2.844 7	2.862 6	3.031 7	0.63	6.57
0k file delete	2.017 6	2.105 1	2.141 7	4.33	6.15
Page fault	0.665 20	0.701 00	2.714 10	5.38	308
Mmap latency	6 807.0	6 963.0	29.8k	2.29	338
fork	44.0	45.1	361	2.50	720
exec	151	158	975	5.30	545
2p/0k ctxsw	0.55	0.58	1.31	5.45	138

此外,2p/0k ctxsw 测量进程切换消耗的时间.同样地,SIM 需要 root 模式截获 MOV TO CR3 操作,以更新相应的影子页表基地址,产生明显的性能开销.而 HyperNE 则接近 native Linux 性能.

表 5 显示了相应的通信带宽.由于 HyperNE 中的所有操作均无需 root 模式的干预,因此实验结果接近甚至达到了 native Linux 的性能,同时也说明使用 EPT 具有很好的性能.而在 SIM 中,由于需要频繁陷入 root 模式以进行内存虚拟化、I/O 及中断虚拟化等操作,产生了较为明显的性能损失.

Table 5 Results of communication bandwidths in lmbench (MB/s)

表 5 lmbench 中通信带宽测试结果 (MB/s)

Benchmarks	native Linux	HyperNE	SIM	HyperNE 性能(%)	SIM 性能(%)
Pipe	8 216	8 152	6 346	99.2	77.2
AF UNIX	8 564	8 532	7 361	99.6	85.9
TCP	5 370	5 320	4 151	99.1	77.3
File reread	8 721.4	8 710.6	8 683.3	99.9	99.5
Mmap reread	15.7k	15.5k	15.1k	98.7	96.2
Bcopy (libc)	10.4k	10.1k	6 578.0	97.1	63.3
Mem read	16.k	16.k	14.k	100	87.5
Mem write	9717	9689	9517	99.7	97.9

综合以上测试结果可以得出:相对于其他基于虚拟化的内核监控模型,HyperNE 能够有效提升 Guest OS 的运行效率.

6 相关工作

虚拟化技术具有运行特权等级高、隔离性强、兼容性强等诸多优势,被广泛应用于操作系统安全领域.在内核监控方面,VMI[1,2]技术使得从外部监控系统状态成为可能,但 VMI 属于被动监控(passive monitoring)[5]范畴,无法对系统状态的更改做出实时响应.Lares[5]首次提出主动监控(active monitoring)模型,通过在内核中添加钩子来获取相应内核事件,但虚拟机间的切换使得监控效率低下.SIM[3]将安全监控软件置于 Guest OS 内部并使用影子页表实现两者的隔离,同时使用 CR3-TARGET 机制,使得从 Guest OS 切换到安全监控软件无需通过 VMM,提高了监控效率.但是在 SIM 中,Guest OS 的运行依然需要 VMM 频繁的干预,尤其是对影子页表更新和进程切换的处理,需要频繁地产生 VM Exit 陷入 VMM,导致 VMM 存在广泛的攻击面.本文利用 EPT 实施隔离,并使用 EPTP switching 机制完成 EPT 切换.此外,本文允许 Guest OS 执行特权操作,自行管理系统资源,而无须 VMM 干预,消除 VMM 的攻击面,同时提升系统运行效率.

一些研究致力于削减或最小化应用程序 TCB 规模.TrustVisor[22]构造了一个特殊用途的 VMM,为应用程序

中的安全敏感代码片段提供一个隔离的执行环境,确保其代码和数据的保密性和完整性以及执行完整性;此外,TrustVisor 在 VMM 中利用软件模拟的方法进行可信度量等相关操作,避免了频繁使用 DRTM 机制而导致巨大的性能开销.在此基础上,SSMVisor[23]允许安全敏感代码调用外部功能函数并无需独占使用 CPU,进一步提高系统的灵活性.相比之下,本文则注重于削减内核监控模型的 TCB;此外,TrustVisor 以及 SSMVisor 中对安全敏感代码的调用需要和返回需要产生 VM Exit 陷入 VMM 中进行处理,会产生一定的性能损耗,而本文则能够快速实现 Guest OS 和安全监控软件之间的切换.

随着 VMM 越来越复杂,VMM 自身的安全性也得到越来越多的关注.HyperSafe[20]对 VMM 运行时的页表及控制流完整性进行了保护;HyperLock[24]对 KVM 模块和内核进行了地址空间隔离;HyperLet[25]则将 KVM 放置在用户态,降低了其运行时的权限;CloudVisor[12]利用嵌套虚拟化技术,将 VMM 的安全保护功能和资源管理功能分离.与现有方法不同的是,本文提出的 HyperNE 则完全抛弃虚拟化功能,消除了 VMM 攻击面.

虽然 NoHype[11]也进行 VMM 攻击面的消除工作,但 NoHype 针对完整的虚拟机运行环境的构建,而本文则关注对系统内核的监控.

7 总 结

本文提出了一个安全、高效的虚拟化内核监控模型 HyperNE.底层 VMM 抛弃了与隔离保护无关的虚拟化功能,允许 Guest OS 自行管理系统资源并执行特权操作,而无需与 VMM 进行交互.此外,HyperNE 将安全监控软件置于 Guest OS 内部,使用 EPT 对二者实施隔离,并通过 EPTP switching 机制,减小 Guest OS 与安全监控软件之间的切换开销.因此,HyperNE 在具有高监控效率的同时,一方面消除了虚拟化引入的攻击面,有效削减了监控模型的 TCB 规模;另一方面也避免了虚拟化开销,显著提高了 Guest OS 的系统性能.

References:

[1] Saberi A, Fu Y, Lin Z. HYBRID-BRIDGE: Efficiently bridging the semantic gap in virtual machine introspection via decoupled execution and training memoization. In: Proc. of the 21st Annual Network and Distributed Systems Security Symp. (NDSS). San Diego: Internet Society, 2014. 1−15. [doi: 10.14722/ndss.2014.23226]

[2] Jain B, Baig MB, Zhang D, Porter DE, Sion R. SoK: Introspections on trust and the semantic gap. In: Proc. of the 2014 IEEE Symp. on Security and Privacy (S&P). San Jose: IEEE, 2014. 605−620. [doi: 10.1109/SP.2014.45]

[3] Shar M, Lee W, Cui W. Secure in-vm monitoring using hardware virtualization. In: Proc. of the 16th ACM Conf. on Computer and Communications Security (CCS). New York: ACM Press, 2009. 477−487. [doi: 10.1145/1653662.1653720]

[4] Pham C, Estrada Z, Cao P, Kalbarczyk Z, Iyer R. Reliability and security monitoring of virtual machines using hardware architectural invariants. In: Proc. of the 44th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN). Atlanta: IEEE, 2014. 13−24. [doi: 10.1109/DSN.2014.19]

[5] Payne BD, Carbone M, Sharif M, Lee W. Lares: An architecture for secure active monitoring using virtualization. In: Proc. of the 2008 IEEE Symp. on Security and Privacy (S&P). Oakland: IEEE, 2008. 233−247. [doi: 10.1109/SP.2008.24]

[6] Carbone M, Conover M, Montague B, Lee W. Secure and robust monitoring of virtual machines through guest-assisted introspection. In Balzarotti D, Stolfo SJ, Cova M, eds. Proc. of the 15th Int'l Symp. on Research in Attacks, Intrusions and Defenses (RAID). Berlin, Heidelberg: Springer-Verlag, 2012. 22−41. [doi: 10.1007/978-3-642-33338-5_2]

[7] Srinivasan D, Wang Z, Jiang X, Xu D. Process out-grafting: An efficient "out-of-vm" approach for fine-grained process execution monitoring. In: Proc. of the 18th ACM Conf. on Computer and Communications Security (CCS). New York: ACM Press, 2011. 363−374. [doi: 10.1145/2046707.2046751]

[8] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. In: Proc. of the 19th ACM Symp. on Operating System Principles (SOSP). New York: ACM Press, 2003. 164−177. [doi: 10.1145/945445.945462]

[9] Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. KVM: The linux virtual machine monitor. In: Proc. of the Linux Symp. Ottawa: OLS, 2007. 225−230.

[10] Common vulnerabilities and exposures. http://cve.mitre.org/

[11] Szefer J, Keller E, Lee RB, Rexford J. Eliminating the hypervisor attack surface for a more secure cloud. In: Proc. of the 18th ACM Conf. on Computer and Communications Security (CCS). New York: ACM Press, 2011. 401−412. [doi: 10.1145/2046707.2046754]

[12] Zhang F, Chen J, Chen H, Zang B. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In: Proc. of the 23rd ACM Symp. on Operating Systems Principles (SOSP). New York: ACM Press, 2011. 203−216. [doi: 10.1145/2043556.2043576]

[13] Elhage N. Virtualization under attack: Breaking out of kvm. 2011. http://www.blackhat.com/html/bh-us-11/bh-us-11-briefings.html

[14] Kortchinsky K. Cloudburst: A VMware guest to host escape story. In: Proc of the Black Hat in Las Vegas. 2009. http://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloudburst-SLIDES.pdf

[15] Wojtczuk R, Rutkowska J, Tereshkin A. Xen 0wning trilogy. 2008. http://invisiblethingslab.com/itl/Resources.html

[16] Intel Corporation. Intel 64 and ia-32 Architectures Software Developer's Manual. Vol.3: System Programming Guide. 2013.

[17] Intel Corporation. Intel Trusted Execution Technology, Software Development Guide, Measured Launched Environment Developer's Guide. 9th ed., 2013.

[18] Intel Corporation. Intel Virtualization Technology for Directed I/O Architecture Specification. Rev. 2.2, 2013. 1−258.

[19] Hofmann OS, Kim S, Dunn AM. Inktag: Secure applications on an untrusted operating system. In: Proc. of the 18th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). New York: ACM Press, 2013. 265−278. [doi: 10.1145/2451116.2451146]

[20] Wang Z, Jiang X. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In: Proc. of the 2010 IEEE Symp. on Security and Privacy (S&P). Oakland: IEEE, 2010. 380−395. [doi: 10.1109/SP.2010.30]

[21] McVoy L, Staelin C. Lmbench: Portable tools for performance analysis. In: Proc. of the USENIX 1996 Annual Technical Conf. (ATC). San Diego: USENIX Association Berkeley, 1996. 279−294.

[22] McCune JM, Li YL, Qu N, Zhou ZW. TrustVisor: Efficient TCB reduction and attestation. In: Proc. of the 2010 IEEE Symp. on Security and Privacy (S&P). Oakland: IEEE, 2010. 143−158. [doi: 10.1109/SP.2010.17]

[23] Li XQ, Zhao XD, Zeng QK. One-Way isolation execution model based on hardware virtualization. Ruan Jian Xue Bao/Journal of Software, 2012,23(8):2207−2222 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4131.htm [doi: 10.3724/SP.J.1001.2012.04131]

[24] Wang Z, Wu C, Grace M, Jiang X. Isolating commodity hosted hypervisors with hyperlock. In: Proc. of the 7th European Conf. on Computer Systems (EuroSys). New York: ACM Press, 2012. 127−140. [doi: 10.1145/2168836.2168850]

[25] Wu C, Wang Z, Jiang X. Taming hosted hypervisors with (mostly) deprivileged execution. In: Proc. of the 20th Annual Network and Distributed Systems Security Symp. (NDSS). San Diego: Internet Society, 2013. 1−15.

附中文参考文献:

[23] 李小庆,赵晓东,曾庆凯.基于硬件虚拟化的单向隔离执行模型.软件学报,2012,23(8):2207−2222. http://www.jos.org.cn/1000-9825/4131.htm [doi: 10.3724/SP.J.1001.2012.04131]

黄啸(1991－),男,安徽桐城人,硕士,主要研究领域为信息安全,虚拟化.

孙浩(1987－),男,博士生,主要研究领域为信息安全,程序分析.

邓良(1987－),男,博士生,主要研究领域为信息安全,操作系统.

曾庆凯(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信息安全,分布计算.