第41卷第10期 中国科学技术大学学报 Vol. 41, No. 10
2011年10月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Oct. 2011

文章编号:0253-2778(2011)10-0907-08

基于虚拟机的安全技术研究

赖英旭,胡少龙,杨 震

(北京工业大学计算机学院,北京 100124)

摘要:由于虚拟机的高隔离性以及对系统、应用的透明性,使得很多安全技术是基于虚拟机实现的.提出一种基于Xen的安全架构,可将现有安全程序移植到该架构上,并保证其功能性,如文件、内存扫描以及主动防御技术.由于大量减少处于被保护虚拟机中的安全程序组件,使得安全程序本身具有更高的安全性,同时利用半虚拟化I/O技术将系统开销降低到最小,具有实用性.该框架还可将其他基于虚拟机的安全技术整合进来,且不需要修改现有的操作系统及应用程序,因此具有较强的适用性.

关键词:虚拟机;隔离执行;内存保护;虚拟机监控;准虚拟化

中图分类号:TP338 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2011.10.011

Research of security technology based on virtualization

LAI Yingxu, HU Shaolong, YANG Zhen

(*College of Computer Science, Beijing University of Technology, Beijing* 100124, *China*)

Abstract: A universal architecture based on Xen was presented, which had traditional security tools transplanted on it and in the meantime guarantees their functions, such as memory and file system scanning, and active defense. Since most components of the security tools are transplanted out of a protected virtual machine, the architecture provides higher security than traditional ones. What's more, it uses paravirtuliation I/O technology to minimize the cost of the virtual machines. Finally, this architecture allows current security technologies based on virtual machines to be integrated into itself conveniently, with no need for the operation system and application running on it to be modified.

Key words: virtual machine; isolated execution; memory protection; VM introspection; paravirtualization

0 引言

当前,随着病毒等恶意程序变得越来越复杂,保护计算机系统变得越来越困难.对某些恶意程序来说,清除它们而不破坏原有系统是不可能的.传统安全软件通常采用提取特征码的方式检测已知病毒,对于未知病毒,则通过收集病毒程序的共同特征及行为配合各种启发式算法来检测.由于目前恶意程序的复杂性,上述方式有漏报和误报情况,没有一种算法能完全无差错地检测出未知病毒,因此导致主动防御技术的产生.这种技术在系统关键点插入检测程序,一旦检测到未定义行为便通知用户,由用户

收稿日期:2011-04-28;**修回日期**:2011-06-27

基金项目:国家重点基础研究发展(973)计划(2007CB311100),国家自然科学基金(61001178),北京市自然科学基金(4102012),北京市教育委员会科技发展计划面上项目(KM200810005030),北京市属高等学校人才强教计划项目(PHR201108016),北京工业大学青年科学基金资助.

作者简介:赖英旭(通讯作者),女,1973年生,博士/副教授.研究方向:网络安全、可信计算.E-mail: laiyingxu@bjut.edu.cn

来决定如何处理该行为.

目前主流的商业反病毒软件都采用对内存、文件系统扫描及主动防御相结合的方式,在用户介入的情况下,效果比传统反病毒软件要好.但计算机用户的噩梦仍在持续,经常遇到计算机上的安全软件自动关闭而且再也启动不起来的情况,这主要是恶意程序同样有机会获取系统最高权限.另外,当前主流操作系统都是共享内核空间的,一旦恶意软件设法驻扎在内核态,即可访问任何数据并有权限破坏安全软件.

近年来,随着硬件性能的提升,虚拟化技术开始普及到桌面用户,相关的研究工作有:监视虚拟机内部行为,虚拟机监视器层面实现对虚拟机的内存保护,隔离执行特定程序并整合执行结果等.

在本文中,首先对虚拟化技术以及现有的基于虚拟化的安全技术进行分类介绍,在此基础上设计了一个基于Xen的虚拟化安全架构,架构接口提供对被保护虚拟机的内存、文件系统访问功能以及不同虚拟机之间的高效通讯机制,通过这些接口,可以将传统安全软件移植到本文架构中,并保证其功能性.该框架通过内存和文件扫描检测已知病毒,通过主动防御技术实现对未知恶意程序的防范,且由于基于该架构的安全软件的核心组件处于特权虚拟机以及虚拟机监视器中,因此对被保护的系统不可见,从而提供更高的安全性;另外通过引入基于准虚拟化I/O的通讯方式,大大降低了由虚拟化带来的系统开销,使得该架构具有更高的实用性.实验表明,本文提出的架构可良好地实现对被保护虚拟机的内存、文件系统访问功能,而基于本架构的一个安全软件原型能够良好地实现主动防御功能,架构提供的内存保护模块能够抵御内核态的恶意代码的篡改,同时,架构引入的系统开销也在可接受范围内.

1 相关工作

VMM(virtual machine monitor or hypervisor)是能够为计算机系统创建高效、隔离副本的软件.这些副本即为虚拟机,在虚拟机内虚拟处理器的指令集的一个子集能够直接在物理处理器上执行[1].其按照实现层次分为3类[2]:

①指令级虚拟化:Bochs,QEMU

②硬件级虚拟化:VMware,XEN,KVM

③操作系统级虚拟化:Linux-vServer,OpenVZ

按照虚拟机监视器所处的位置,又可分为类型Ⅰ和类型Ⅱ[1],如图1所示.类型ⅠVMM直接运行在计算机硬件系统上,负责调度和分配系统硬件资源,其中具有控制功能的称为特权虚拟机,其他虚拟机称为客户虚拟机.类型ⅡVMM则以一个应用程序的形式运行在已有的传统操作系统之上,其中实际控制系统资源的操作系统被称为宿主系统.

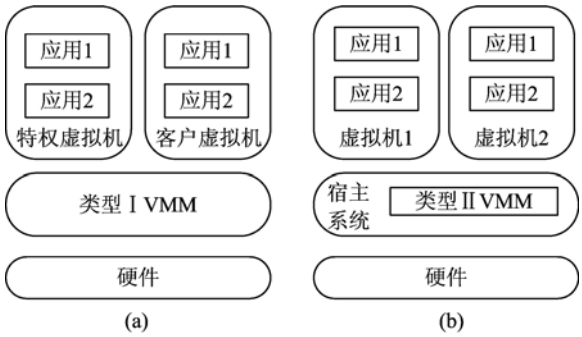

图1 类型Ⅰ和类型Ⅱ虚拟机

Fig. 1 Type Ⅰ and Type Ⅱ Virtual machine

目前基于虚拟机技术,出现了很多安全类研究成果,例如虚拟机自省技术(introspection)和OpenTC这样的项目.为了更好地理解这些技术,本文将从隔离执行、虚拟机监控和内存保护3个方面进行介绍.

1.1 隔离执行

隔离执行是指将程序隔离到一个封闭的环境中运行,按照其目的可分为隔离执行具有危险性的程序以及隔离执行需要重点保护的关键应用,目前大多数此类研究都使用了虚拟化技术.Huang[3]等基于OpenVZ实现了一个隔离执行架构,能够维护整个虚拟环境的完整性,并追踪多个应用之间的交互.这种基于操作系统级别的虚拟化技术,如OpenVZ,Linux-vServer,是共享操作系统内核的,因此无法有效抵御内核态的恶意代码,一旦恶意代码进入操作系统内核则可以绕过此类隔离机制直接访问受保护的系统资源.相比之下,硬件虚拟化的开销更大,但隔离性更强.Isolated Execution[4]是一个基于XEN[5-6]的开源项目,可以将程序发送到沙箱虚拟机中自动执行,但沙箱虚拟机中的环境需要手动配置,导致一些程序无法直接运行.SVEE[7-8]解决了在虚拟环境中,对宿主主机环境的重现问题,并实现了虚拟环境中程序运行结果的差异提交.Isolated Execution和SVEE是隔离执行具有危险性程序的代表.Terra架构[9]基于TPM将可信硬件平台分成多个独立的虚拟环境,其中有通用的环境和封闭式

的高安全性的环境，并基于 VMware GSX 实现了系统的原型. OpenTC[10] 利用开源操作系统和虚拟机开发具有示范和推广作用的可信计算系统架构，分别基于 XEN 和 L4 开发出可信操作系统，其中 XEN 用于个人用户，L4 用于移动设备. OpenTC 提供了 3 种应用原型，其中私人电子交易系统原型和 Terra 是隔离执行关键应用的代表.

1.2　虚拟机监控

虚拟机的高隔离性引入了一个新的问题，那就是虚拟环境中的用户行为不可见，既无法对其进行监控. 于是，虚拟机自省技术(VM introspection)出现了，用以消除不同虚拟机之间的语义鸿沟(semantic gap). Garfinkel 等[11] 最早阐述了这种技术，并基于 VMware Workstation 实现了原型. 而 XenAccess[12] 和 VIX[13] 都是该类技术基于 Xen 的实现，它们都能够使特权虚拟机对客户虚拟机内存进行访问，但是对文件系统的访问功能很有限. 需要注意的是，虚拟机自省技术完全实现在虚拟机监视器以及特权虚拟机层，其只提供监视功能而无法提供任何控制功能. Lares[14] 提出了一种基于 Xen 的类似于现在反病毒软件的主动防御技术，但其采用的不同虚拟域之间的通讯方式十分简单，使得其通讯效率很低，并会给系统造成很大开销，不适合需要传输大量信息的操作，如在虚拟机外部扫描其文件系统.

1.3　内存保护

内存保护的一种方法是周期地检测程序内存，从而发现恶意的修改. 基于虚拟化的内存保护技术也开始出现，如 SecVisor[15] 是一种基于虚拟机的内核代码完整性验证架构，Xu 等[16] 同样提出了一种类似的技术来实现内核代码完整性验证. Manitou[17] 使得用户可以对代码授予、追踪及收回执行权限. Lares[14] 提出了一种在虚拟机监视器层对虚拟机字节细粒度内存的写保护机制. Intel[18] 实现了一种基于轻量级虚拟机的内存保护方法，通过分离式页表将内核内存和应用程序内存分离，从而解决操作系统共享内核空间带来的安全性问题. 然而这种轻量级虚拟机不具有通用性，其上的应用程序需要进行改写以显示调用其内存保护功能.

综上所述，基于虚拟化的安全技术如果想得到广泛应用，首先应具有实用性，即能够实现传统安全软件的功能：既能够对虚拟机内部的内存和文件系统进行扫描，又能够对其内部行为进行干预，从而实现主动防御功能；同时其必须提供较传统模式更高的安全级别；另外，由于虚拟化引入的系统开销必须控制在可接受的范围内.

2　设计原则

为了达到上述目标，在设计架构时除了要保证功能性，还必须遵守以下原则：

原则 1：高隔离性. 要求使用虚拟化技术必须提供足够高的隔离性以保证关键应用的安全.

原则 2：对应用和操作系统不可见. 为了具有通用性，架构必须适应当前主流操作系统及应用.

原则 2A：除基于架构的安全程序外，其他应用程序不需要被改写.

原则 2B：架构对操作系统透明.

原则 3：高性能. 架构不能引入过高的系统开销和延迟.

原则 3A：架构采用的虚拟化技术必须提供高性能.

原则 3B：架构本身引入的开销应降至最低.

原则 4：高安全性.

原则 4A：架构在被保护虚拟域中的组件应尽可能少.

原则 4B：如果在被保护虚拟域中存在组件，则必须对其进行额外的防护.

原则 1 要求虚拟化技术提供高隔离性，即不能采用操作系统级的虚拟化技术，如 OpenVZ，Linux-vServer. 而为了实现原则 3A，类型Ⅰ的虚拟机是最好的选择，因此本文的架构最终选择基于 Xen 来实现.

原则 2 保证了架构是否可以得到广泛应用. 原则 2A 意味着如 Intel 提出的需要改写应用程序的方式[18] 是不适用的，而原则 2B 要求架构能够使用于非开源的操作系统，如 Windows.

在本文阐述的架构中，系统开销和延迟主要来自于以下几个部分：①截获系统调用；②不同虚拟域之间的通讯；③策略决策；④对被保护虚拟域中组件的额外防护. 其中①和③在传统安全系统中同样存在，②和④是本文架构所额外引入的，因此必须保证其处于可接受范围内.

根据原则 4，将安全程序核心组件从被保护的虚拟域中转移出来，而为了进一步满足原则 4B，对遗留在虚拟域中的部分实现了虚拟机监视器层的内存保护.

3 架构设计与实现

本文提出的架构是通过在 Xen 的基础上进行修改,并添加一些组件来实现,整体结构为类型Ⅰ虚拟机典型架构,除 Xen 自身组件外,包含 4 个模块:首先是处于被保护虚拟域中的前端驱动,实现为一个虚拟的 PCI 驱动,负责截获系统调用并与特权虚拟域通信;第 2 个组件是处于特权虚拟域中的后端驱动,实现为一个内核模块,它与前端驱动通信,获取系统截获信息,并传递给特权虚拟域中用户态的决策模块;决策模块用于策略制定和决策,在原型系统中,它是一个包含应用黑白名单及签名的小型数据库;第 4 个模块是处于虚拟机监视器层的内存保护模块,对遗留在被保护虚拟域中的组件提供内存写保护.架构整体结构如图 2 所示,其中,被保护的虚拟域称为客户域(guest domain),特权域(domain0)为具有控制生成客户域的特权虚拟域,客户域与特权虚拟域之间通过 I/O Ring 进行通信;内存保护模块处于虚拟机监视器中,运行在系统最高级.

3.1 前端驱动

前端驱动运行在客户虚拟域中,实现为一个虚拟 PCI 驱动,随系统启动自动初始化.前端驱动使用传统的钩挂 SSDT 表并设置跳转码来截获系统调用,截获的信息被发送到后端驱动,用于决策模块判断.在本文架构中,前端驱动与后端驱动通信时,采用了准虚拟化[19]的通信方式,由于准虚拟化要求修改操作系统内核,不适用于 Windows 这样的商业操作系统,因此借助了 PV-on-HVM 技术[20],使用这种方式在全虚拟化的虚拟机上使用准虚拟化的分离式驱动模型,从而大大提高 I/O 效率.这种技术首先要求客户虚拟域明确知道其处于虚拟机中,并移植分离式驱动模型用到的 Xen 机制的代码,包括超级调用(hypercall)、事件通道、Xenstore、授权表(grant table).超级调用是用来陷入虚拟机监视器层的特殊系统调用,事件通道是 Xen 提供的异步通信机制,授权表是 Xen 在不同虚拟域间共享内存的授权机制,Xenstore 是基于共享内存方式实现的,用于在客户虚拟域启动时读取特权域虚拟机提供的设备初始化信息,前端驱动初始化时也使用 Xenstore 与后端驱动交换初始化信息,之后通过事件通道和 I/O Ring 与后端驱动进行通信.前端驱动截获系统信息后,使用 SCHEDOP_block 向虚拟机监视器请求放弃调度,等待决策模块进行决策,相比 Lares 中使用超级调用阻塞在虚拟机监视器中的方式,大大提高了性能.

3.2 后端驱动

后端驱动被实现为一个内核模块,主要有两个功能:①与前端驱动通信获取截获的系统信息和相关数据,然后将这些数据传递给用户态的决策模块,决策模块做出判断后,将结果发送给前端驱动;②另外还需要在前端驱动初始化钩子及跳转代码后,通知虚拟机监视器中的内存保护模块保护相应的内存地址范围.由于特权虚拟域本身是准虚拟化的,其操作系统代码已经被修改过,可以直接使用 Xen 的各种机制,不需要像前端驱动移植大量代码,因此相对简洁.I/O Ring 在准虚拟化驱动模型中被广泛使用,它是在 Xen 共享内存机制上实现的分离式驱动交换数据的结构,分为两种:①固定槽(fixed slot)大小的 I/O Ring,由网络、存储设备所使用;②可变槽

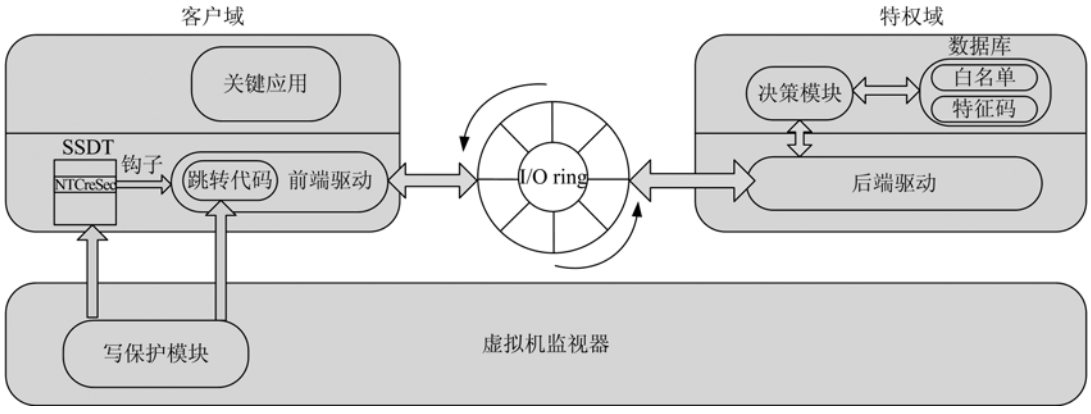

图 2 本文提出的基于 XEN 的安全架构

Fig. 2 Our security architecture based on XEN

(variable)大小的 I/O Ring,由 Xenstore 所使用.在本文架构中,需要传输的数据主要有两种:文件信息及其 HASH 值和需要被扫描的文件.为了简化设计,在原型系统中对这些信息采用统一固定大小槽的 I/O Ring,在客户和虚拟机中共享 16 个页面,每个槽 4k 字节,并用一个标志域来区分数据类型.当后端驱动初始化时,它分配一个未绑定的事件通道,并初始化 I/O Ring,然后将事件通道端口号和 Ring 地址写入 Xenstore.前端驱动通过 Xenstore 读取这些信息,绑定该事件通道,映射 Ring 地址,并使用超级调用通知虚拟机监视器通知该虚拟域时应该使用的 IRQ 号,同时为此 IRQ 注册一个处理函数.前、后端驱动交互过程如图 3 所示,具体为:①后端驱动分配未绑定事件通道和共享页;②后端驱动初始化 I/O Ring;③后端驱动将 Ring 地址和通道端口号写入 Xenstore;④前端驱动从 Xenstore 读取相应信息;⑤前端驱动绑定事件通道,映射 Ring 内存;⑥前端驱动通知虚拟机监视器所使用的 IRQ 号;⑦前端驱动分配一个事件通道给后端驱动绑定,建立双向连接.

3.3　决策模块

决策模块处于特权虚拟域中的用户空间,它的功能包括一个存储应用程序的黑白名单和签名的小型数据库,主要用来判断一个应用程序的合法性,并决定是否允许该应用程序执行,另外包括一个用户界面以便用户可以介入决策.决策模块使用普通的通讯方式与内核模块通信.在本文的架构中,决策模块所获取的信息是由前端驱动截获并发送回来的,并没有使用虚拟机自省技术,需要指出的是,虚拟机自省技术可无缝地整合进本架构,如 XenAccess.

3.4　内存保护模块

此时核心组件已经从被保护的客户虚拟机中转移出来,使得恶意程序无法对其进行攻击,但在架构设计原则中,原则 4B 要求对遗留在其中的组件进行额外保护,需要指出的是,为了保证功能性和实用性,被保护虚拟域中的组件是必需的,而内存保护模块正是用来满足这一要求的.在这一部分中,借鉴了 Lares 中提出的字节细粒度的内存保护方法.在 Xen 中,虚拟 Windows 必须使用硬件辅助虚拟化技术[21-22],在硬件辅助虚拟域(HVM)中,使用影子列表(SPT)来实现内存虚拟化,SPT 维护着虚拟地址到真实机器地址的映射,当客户虚拟机运行时,虚拟机监视器使用 SPT 代替客户虚拟机页表(GPT),借助于影子页表来实现内存写保护.

首先记录下需要保护的内存地址范围,SPT 初始时是空表,随着系统运行逐渐建立,在为一个页面生成 SPT 的过程中,检测 GPT 中的这页是否包含需要保护的内存地址:如果是,则将该页在 SPT 中标记为只读,这样就实现了页级写保护;但是一个页面中并不是所有字节都是需要保护的,这时如果发生一个写异常,就需要在处理函数中进一步检查引起该异常的语句中写入的地址是否是要保护的地址,将 CR2 寄存器中引起异常的地址与被保护地址进行比较,如果是则向客户虚拟机返回一个页异常,否则需要模拟该写操作.该流程如图 4 所示.

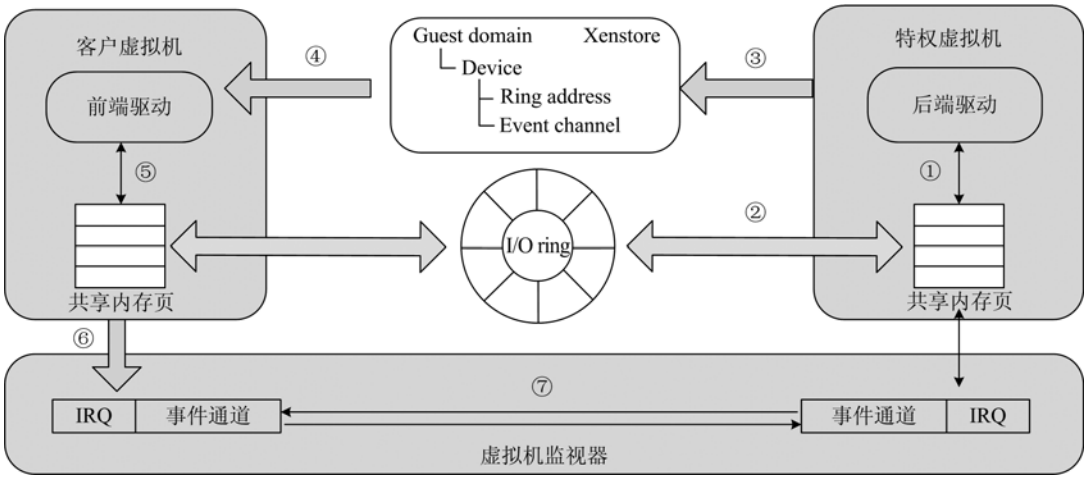

图 3　前后端驱动交互过程

Fig. 3　Interaction between frontend and backend

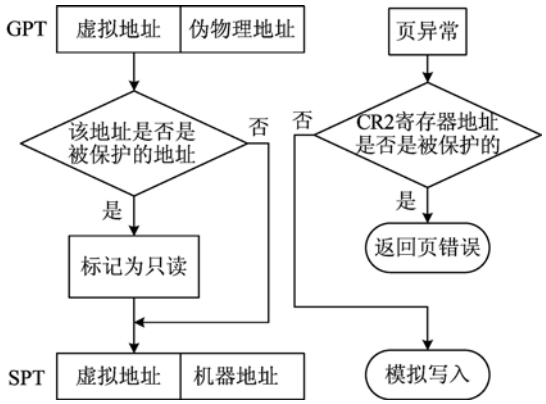

图 4 影子页表生成过程及写异常时的判断流程

Fig. 4 Shadow page generation process and judgment of write expectation

4 性能与评测

本文的原型系统模拟了一个简单的基于本文架构的安全软件，实现了简单的进程截获，发送进程信息到特权域的决策模块进行判定，实验显示该系统可良好地截获进程创建，并提示给用户判断：在客户域中体现为，用户运行某个进程，没有任何响应，其他任务正常运行；特权域中提示该客户域中进程信息，如选择允许创建，客户域中进程正常启动，如选择禁止，系统提示该程序为非法应用程序并退出.

本文架构的应用场景与当前大多数基于虚拟机的安全技术不同，如基于虚拟机的沙盒、蜜罐系统等. 这些系统的主要目的是分析、取证、重现攻击过程，其内部系统并不是用户日常使用的计算环境，其安全性并不高，或者以牺牲一部分功能换取一定的安全性，如沙盒. 而本文架构的目的在于为用户生成日常使用的虚拟机环境，用户在该虚拟机中进行日常工作、电子交易等，通过本架构提供的内存、文件扫描以及主动防御技术，由于其采用了分离架构以及虚拟机监视器层的内存保护技术，使之具有高度的安全性.

对于内存写保护的测试，采用一段驱动代码模拟恶意程序篡改 SSDT 表：

```
__asm
{   push eax
    mov eax,CR0
    and eax,0FFFEFFFFh
    mov CR0,eax
    pop eax
}   //打开保护
//修改 SSDT
__asm
{   push eax
    mov eax,CR0
    or eax,NOT 0FFFEFFFFh
    mov CR0,eax
    pop eax
}   //关闭保护
```

实验显示开启内存写保护前，SSDT 表可成功被篡改，注册 SSDT 表内存范围到内存保护模块后，SSDT 表无法被篡改.

本文架构性能的优越性主要体现为：由于采用前后端驱动通讯方式，使得在客户虚拟域进程等待特权虚拟域判定时，系统开销减少，加快了对客户域文件扫描的速度.

Lares 通过超级调用陷入虚拟机监视器等待，特权虚拟域判定的方法会使虚拟机监视器占用大量 CPU 时间，假设采用由用户判定的主动防御方式，用户反应可能需要几十秒钟或者更久，如果碰巧有多个虚拟机同时进入等待，那么造成的系统开销是灾难性的，这也违反了不应该将大负荷的运算引入虚拟机监视器层的原则.

需要注意的是，模拟这样一个超级调用运行时，无法简单地从特权虚拟域或者其他客户域 CPU 使用率来发现这一问题. 这是因为类型 Ⅰ 的虚拟机监视器之上所有域均为虚拟机，包括特权虚拟域，其 CPU 均为虚拟 CPU(VCPU)，在真实 CPU 上不停进行调度，因此无法体现物理 CPU 的真实负荷. 一种方法是，首先确定所有虚拟域 VCPU 都会参与调度，即没有被绑定到某个物理 CPU 上，包括特权虚拟域，并且 VCPU 总数大于物理 CPU 总数. 运行程序造成所有 VCPU 百分之百负荷，在特权虚拟域下，查看/proc/stat 中 CPU 一行的第 8 个值，这个值叫做被盗取值(stolen value)，代表了运行在虚拟环境时被其他虚拟域占用的 CPU 时间. 在其他虚拟域调用这样一个超级调用后，stolen value 单位时间增加值会明显增加. 而采用本文的方法，则基本不变. 模拟 Lares 的方法，构造一个超级调用在虚拟机监视器中忙等，在 domain0 中调用该超级调用，在 CPU 为双核 2.33 GHz 的主机上，当传入的 time 值大于 10 万次，即会造成可感知的整个系统停滞，包括特权虚拟域和其他的客户虚拟机，无法进行任何操作.

本文架构提供对保护虚拟域内部文件系统的访问功能，使得基于本文架构的安全软件能够对客户域文件系统进行扫描，实验表明其性能比较理想，而前端驱动给客户域带来的额外系统开销也在可接受范围内，实验结果如图 5 和图 6 所示.

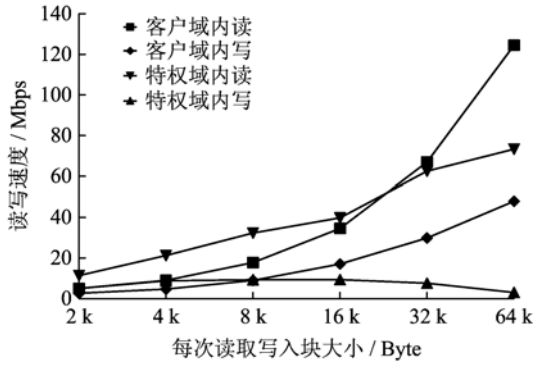

图 5　读写速度

Fig. 5　Compare of writing and reading speed

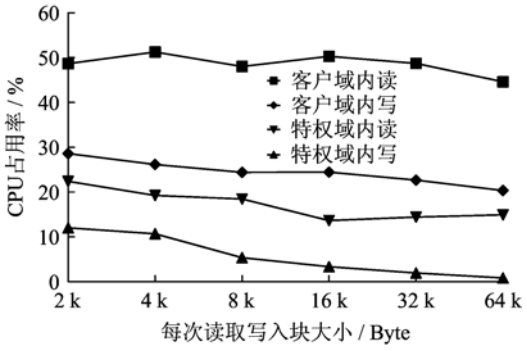

图 6　CPU 占用率对比

Fig. 6　Compare of CPU usage

为了进行对比，在客户域中对相同的文件进行同类操作，测定两者的读写速率以及 CPU 使用率，客户域使用了 QEMU 模拟 I/O，测试环境为：

①CPU：Intel Core 2 Quad Q9500 2.83 GHz

②特权域：RHEL5.6-2.6.18.238.el5xen

③客户域：WinXP，vcpu = 1，device = qemu-dm，image=raw

从图 5 可以看到，基于准虚拟化的优势，本文架构对客户域中文件系统的读操作在 16 k 以前，写操作在 8 k 前甚至比在客户域中直接访问速度更快，随着读写块大小的增加，性能开始低于模拟 I/O. 本架构目前只使用了读取的功能，总体来说性能还是比较理想的.

从图 6 可以看出在 CPU 使用率上，较模拟 I/O 大大降低，这也得益于基于准虚拟化 I/O 的方式. 这里 CPU 使用率的统计仍然采用了传统方法，由于系统中只有特权域和一个客户域，因此可以看作客户域独占一个物理 CPU.

5　结论

本文提出的架构采用了类型Ⅰ的虚拟机，具有比较好的隔离性，无论使用本架构隔离执行具有危险性的程序或者隔离执行需要重点保护的程序，都能够提供比较好的安全性和实用性：通过整合虚拟机自省技术，可以对被保护虚拟机的内存进行访问；而对于需要重点保护的应用，该架构不仅能够提供传统安全软件的常用功能，如文件系统、内存扫描、特征码提取及主动防御技术，而且由于将系统核心组件转移到特权域中，使得被保护系统中的恶意程序无法对其进行攻击，提高了系统自身的安全性. 同时，由于采用了基于准虚拟化方式的通讯机制，适合于如文件扫描等需要传输大量数据的功能，具有很高的实用性，并且具有良好的性能. 该架构适用于目前主流的操作系统和应用，在本文的原型系统中，客户虚拟机运行 Windows 系统，其前端驱动部分可被方便地移植到 Linux 系统中.

参考文献(References)

[1] Goldberg R P. Architectural Principles for Virtual Computer Systems [D]. Cambridge: Harvard University, 1972:1-5.

[2] Adams K, Agesen O. A comparison of software and hardware techniques for X86 virtualization [C]// Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2006:2-13.

[3] Huang Y, Stavrou A, Ghosh A K, et al. Efficiently tracking application interactions using lightweight virtualization [C]// Proceedings of the 1st ACM Workshop on Virtual Machine Security. New York: ACM, 2008:19-28.

[4] Isolated Execution. [2008-11-21]. http://isolated-exec.sourceforge.net.

[5] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization[C]//Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. New York: ACM, 2003:164-177.

[6] Pratt. Xen 3.0 and the art of virtualization [C]// Proceedings of the Ottawa Linux Symposium. Ottawa: 2005 Linux Symposium, 2005: 65-78.

[7] Wen Yan, Wang Huaimin. A isolated execution model

based on local virtualization technology [J]. Journal of Computer, 2008,10: 1 768-1 779.
温研, 王怀民. 基于本地虚拟化技术的隔离执行模型研究[J]. 计算机学报, 2008,10: 1 768-1 779.

[8] Wen Yan, Liu Bo, Wang Huaimin. A safe virtual execution environment based on the local virtualization technology[J]. Computer Technology and Science, 2008, 30 (4):1-10.
温研, 刘波, 王怀民. 基于本地虚拟化技术的安全虚拟执行环境[J]. 计算机工程与科学, 2008, 30 (4): 1-10.

[9] Garfinkel T, Pfaff B, Chow J, et al. A virtual machine-based platform for trusted computing[C]// Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. New York: ACM, 2003:193-206.

[10] OpenTC. [2005-11-15]. http://www.opentc.net

[11] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection [C] //Proceedings of the Internet Society's 10th Symposium on Network and Distributed System Security. San Diego: ISOC, 2003:1-16.

[12] Payne B D, Carbone M, Lee W. Secure and flexible monitoring of virtual machines[C]// Proceedings of 23rd Annual Computer Security Applications. Washington, DC: IEEE Computer Soc, 2007:385-397.

[13] Hay B, Nance K. Forensics examination of volatile system data using virtual introspection [J]. ACM SIGOPS Operating Systems Review, 2008:74-82.

[14] Payne B D, Carbone M, Sharif M, et al. An architecture for secure active monitoring using virtualization[C] // Proceedings of the 2008 IEEE Symposium on Security and Privacy. Washington, DC: IEEE Computer Soc, 2008: 233-247.

[15] Seshadri A, Luk M, Qu N, et al. A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes[C]//Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles. New York: ACM, 2007:335-350.

[16] Xu M, Jiang X, Sandhu R, et al. Towards a VMM-based usage control framework for OS kernel integrity protection [C]//Proceedings of the 12th ACM Symposium on Access Control Models and Technologies. New York: ACM, 2007: 71-80.

[17] Litty L, Lie D. Manitou: A layer-below approach to fighting malware [C] //Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability. New York: ACM, 2006:6-11.

[18] Dewan P, Durham D, Khosravi H, et al. A hypervisor-based system for protecting software runtime memory and persistent storage [C] // Proceedings of the 2008 Spring Simulation Multiconference. New York: ACM, 2008:828-835.

[19] Fraser K, Hand S, Neugebauer R, et al. Safe hardware access with the Xen virtual machine monitor [C] // Proceedings of OASIS ASPLOS 2004 Workshop. New York:ACM, 2004:1-6.

[20] PV on HVM. [2011-10-18]. http://wiki.xen.org/xenwiki/XenLinuxPVonHVMdrivers

[21] Uhlig R, Neiger G, Rodgers D, et al. Intel virtualization technology[J]. Computer, 2005, 38(5): 48-56.

[22] Dong Y, Li S, Mallick A, et al. Extending Xen with intel virtualization technology [J]. Intel Technology Journal, 2006, 10:1-6.