·计算机工程与应用·

基于KVM的Windows虚拟机用户进程防护

陈兴蜀，赵 成，陶术松

(四川大学计算机学院 成都 610065)

【摘要】为保护Windows虚拟机中进程的内存和系统调用执行路径免受恶意代码的威胁，提出了一种基于KVM的虚拟机用户进程防护方案。结合硬件虚拟化技术，为Windows虚拟机构造一份影子内核以绕过恶意代码对原内核系统调用路径的挂钩，保护进程系统调用路径的安全。同时，在监控代理中过滤跨进程系统调用，在KVM中拦截虚拟机页表切换行为并监控虚拟机断点异常与调试异常，保护进程内存的安全。另外，构造影子监控代理，实现对虚拟机监控代理内存的安全防护。最后，实现了基于KVM的虚拟机用户进程防护系统VMPPS，并对其有效性进行了系统测试与分析。实验结果表明，在性能损失可接受范围内，进程内存与进程系统调用执行路径能够得到有效防护。

关 键 词 监控代理；安全防护；用户进程；虚拟化；虚拟机

中图分类号 TP309 **文献标志码** A doi:10.3969/j.issn.1001-0548.2016.06.013

KVM-Based Windows Virtual Machine User Process Protection

CHEN Xing-shu, ZHAO Cheng, and TAO Shu-song

(College of Computer, Sichuan University Chengdu 610065)

Abstract To protect the process memory and execution paths of system calls from the threat of malicious code on Windows virtual machine, a KVM-based virtual machine user process protection solution is proposed. Combined with hardware virtualization technologies, a shadow kernel is built for Windows virtual machine to protect the original kernel system call paths from being hooked by malicious code. Meanwhile, the process memory is secured through filtering out-of-process system calls in the monitoring agent, intercepting the switching behaviors of page tables, monitoring the exceptions of breakpoints, and debugging of the virtual machine. In addition, a shadow monitoring agent is built to safeguard the virtual machine's monitor agent memory. A prototype system VMPPS was thus designed and implemented with its validity tests and analysis results showing that process memory and execution paths of system calls of the virtual machine are effectively protected within an acceptable performance loss range.

Key words monitor agent; security and protect; user process; virtualization; virtual machine

云计算的发展使业界更加关注云计算平台的安全性[1]。Windows是虚拟机主流操作系统之一，而目前Windows平台上存在多种恶意程序，如间谍软件[2]、后门程序[3]等，可通过远程代码注入等方式入侵系统进程。内核态恶意代码[4]具有更高特权级，能够劫持进程系统调用的执行路径，窃取进程内存数据。美国国家标准机构NIST发布的《完全虚拟化安全指南》[5]指出，传统主机所面临的安全威胁同样适用于云环境下的虚拟机。由于云平台上用户、信息、资源等高度集中，而进程是操作系统资源分配的基本单位[6]，若恶意代码对虚拟机进程进行攻击，则会严重威胁云平台的安全。因此，研究云平台虚拟机进程的安全防护具有重要现实意义。

传统的基于主机的进程防护方法同样可用于云环境下的虚拟机，主要利用内核态的监控代理实施权限限制、系统访问控制等，实现对系统进程的安全防护。文献[7]为Windows系统核心资源提供了细粒度的权限控制，文献[8]利用系统调用接口服务随机化防止代码注入攻击。但该类方法易被内核态rootkit攻击或绕过。在虚拟化环境下，虚拟机监视器(virtual machine monitor，VMM)具有比虚拟机更高的特权级，可对虚拟机进程提供更高的安全性。Overshadow[9]可以将受保护的虚拟机进程加密，但在闭源的Windows虚拟机中应用能力有限。文献[10]

收稿日期：2016－01－18；修回日期：2016－05－25
基金项目：国家自然科学基金(61272447)
作者简介：陈兴蜀(1969－)，女，教授，博士生导师，主要从事信息安全、云计算安全方面的研究.

将虚拟机进程内存数据隐藏以防止其他进程访问，但受保护进程无法与其他进程传递数据。文献[11]构建的虚拟机进程内存防护系统突破了上述限制，Linux虚拟机第三方进程可读取但禁止修改受保护进程内存数据，所以受保护进程不能与其他进程进行完全的数据读写共享。其他如Flicker[12]、CHAOS[13]、Proxos[14]等原型系统需要特定的受保护进程与防护系统的配合，但不能任意指定进程进行防护。

针对Windows虚拟机进程面临的威胁及现有方法的不足，本文提出影子内核机制以透明的保护虚拟机进程系统调用执行路径，利用KVM和监控代理监控虚拟机中访问进程内存的行为，并构建影子监控代理以保证监控代理的安全性，最后实现了原型系统VMPPS。VMPPS能够指定Windows虚拟机中任意进程，对其系统调用执行路径及内存进行安全防护。同时，为防止VMM自身存在的漏洞也可能被攻击者利用，导致虚拟机逃逸[15-16]等问题，本文引入HyperSafe[17]、HyperDomain[18]和Hypercheck[19]等方案加固VMM。

1　系统设计

KVM是基于硬件虚拟化实现的一个Linux内核模块，需要Intel-VT[20]或AMD-V技术[21]的支持，负责管理虚拟机内存及CPU，处理引发VM Exit的事件。以Intel-VT为例，Intel-VT为CPU增添了支持硬件虚拟化的指令，提供了Root和Non-Root两种运行模式。虚拟机运行于Non-Root模式，KVM运行于Root模式。虚拟机发生异常或执行特权指令时，引发VM Exit陷入到KVM。KVM处理完毕引发VM Exit的事件后，执行VM Entry使虚拟机继续运行。同时，KVM利用VMCS(virtual machine control structure)[22]保存宿主机与虚拟机的硬件环境、退出信息等内容，并可以指定引发VM Exit的事件，由硬件自动读写。

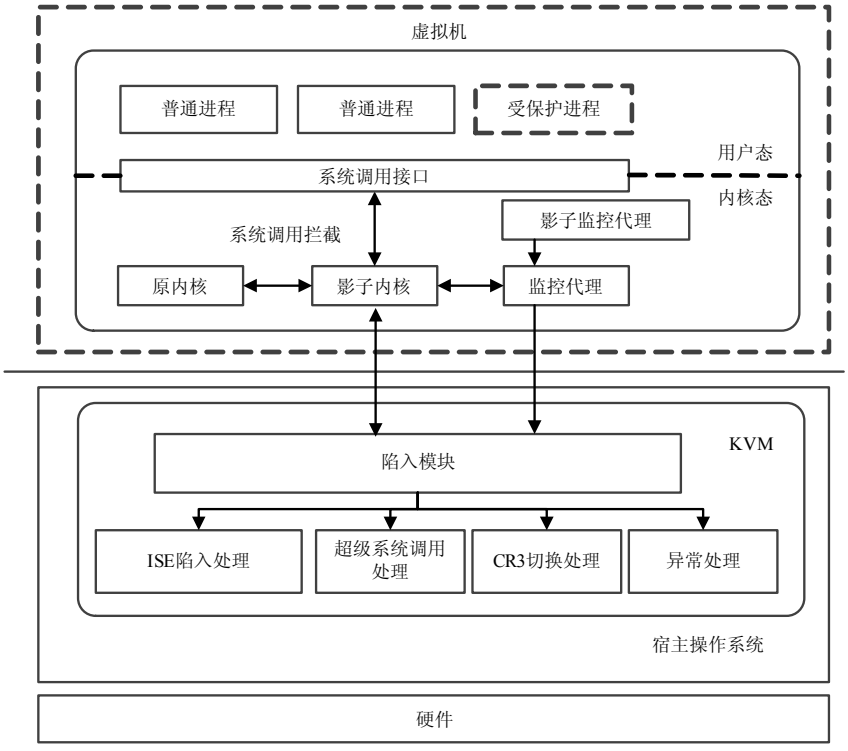

图1　VMPPS总体架构

本文基于KVM实现了原型系统VMPPS，对Windows虚拟机进程系统调用执行路径及内存进行安全防护，总体架构如图1所示。在系统调用执行路径防护方面，虚拟机运行过程中，普通进程系统调用执行路径处于原内核，而被保护进程系统调用执行路径完全处于影子内核，绕过了原内核中可能存在的恶意挂钩。影子内核独立存在，只能通过KVM实现与原内核之间的切换，因此不需对其引入额外的防护流程即可避免恶意软件对影子内核的攻击。在内存防护方面，本文利用KVM具有的高特权级特性，捕获虚拟机特定硬件动作，监控虚拟机异常调试等行为，保护虚拟机进程内存安全。其中，影子内核是监控代理利用原始内核镜像文件，在虚拟机内核非分页池中构建的一份纯净的内核可执行代

码，包含了原内核所有的系统调用执行路径，并提供给受保护进程进行调用。监控代理拦截了影子内核系统调用分发流程，对跨进程读写系统调用进行过滤，防止第三方进程读写受保护进程内存。同时，将监控代理中的过滤函数重定位到影子监控代理中，保护监控代理的安全。

陷入模块位于KVM中，主要包括4个功能模块：

1) ISE陷入处理模块，负责捕获虚拟机中读写IA32_SYSENTER_EIP动作，确保影子内核在第一时间接管虚拟机进程的系统调用请求；

2) 超级系统调用处理模块，负责接收监控代理向KVM传递的数据，如原内核SSDT表基地址、影子内核SSDT表基地址等；

3) CR3切换处理模块，负责捕获虚拟机进程切换行为，并根据内核栈保存的返回地址判断是否有第三方驱动读写受保护进程的行为；

4) 断点及调试异常处理模块，在KVM中捕获虚拟机断点及调试异常，负责保护虚拟机受保护进程免受恶意的调试分析。

2 系统实现

2.1 进程系统调用执行路径安全防护

2.1.1 构建影子内核

在Windows操作系统中，内核镜像是PE(portable executable)格式的可执行文件，并按照此格式被加载进内存。影子内核的构建过程与Windows操作系统在加电启动中加载内核镜像至物理内存中的过程类似，但需要额外的步骤保持影子内核数据的正确性：

1) 将内核镜像以PE文件格式进行内存映射，加载内核镜像文件中的代码及数据，形成影子内核代码初始态。

2) 影子内核数据未经历系统开机启动时刻的内核初始化，需重定位到原内核数据，影子内核才能正常执行。依据影子内核代码及数据与原内核代码及数据相对位置不变这一原理，有：

$$D_{\text{actual}}-D_{\text{actual\_imageBase}}=D_{\text{default}}-D_{\text{default-imageBase}} \quad (1)$$

式中，$D_{\text{actual_imageBase}}$是原内核加载基地址；$D_{\text{actual}}$是原内核中数据的正确值地址，也是影子内核重定向数据指向的数据位置；$D_{\text{default-imageBase}}$是影子内核加载基地址；$D_{\text{default}}$是重定向表给出的需要修复的重定向数据地址，如内核中的全局数据。

3) 修复影子内核系统调用表中系统调用的地址，使其指向影子内核纯净的可执行代码。根据影子内核加载基地址与原内核加载基地址之间的偏移，可依据式(2)确定每一个系统调用函数地址：

$$\text{NewSSDTFuncAddr}=\text{OrigSSDTFuncAddr}+\Delta \quad (2)$$

式中，NewSSDTFuncAddr是影子内核SSDT表中的某个系统调用函数地址；OrigSSDTFuncAddr是原内核SSDT表中对应的系统调用函数地址；Δ是影子内核加载基地址与原内核加载基地址之间的偏移量。

2.1.2 影子内核接管系统调用

影子内核创建完成后，VMPPS采用拦截快速系统调用机制，在第一时间接管Windows虚拟机进程系统调用执行流程，对受保护进程系统调用执行路径进行安全防护。Windows XP及后续操作系统版本引入了快速系统调用机制，将系统调用入口函数地址存放于IA32_SYSENTER_EIP寄存器，通过sysenter指令替代软中断int 2e以提高系统调用效率。

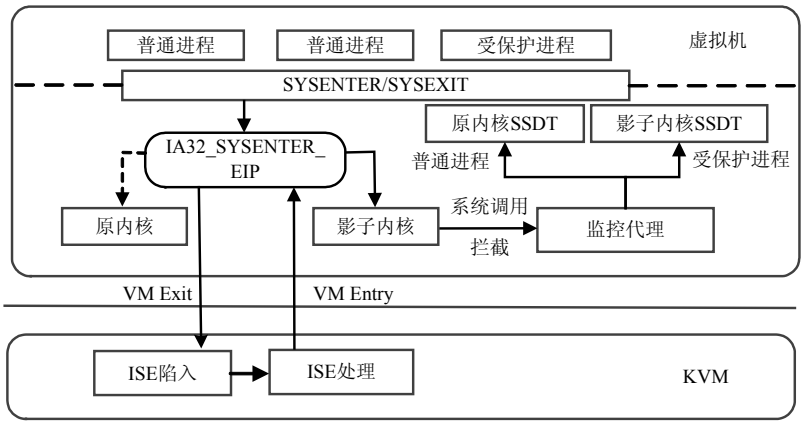

图2 进程系统调用执行路径安全防护架构

进程系统调用执行路径安全防护架构如图2所示。首先，在KVM中，将VMCS中MSR Bitmap的0x176位置1，以拦截虚拟机对IA32_SYSENTER_EIP寄存器的读写，并透明的设置IA32_SYSENTER_EIP寄存器指向影子内核系统调用入口函数地址。当虚拟机执行rdmsr/wrmsr指令对IA32_SYSENTER_EIP

寄存器进行读写时，会引发VM Exit被ISE陷入模块捕获，随后由ISE处理模块进行处理：1) 对rdmsr指令的读取操作，返回原内核系统调用入口函数值，保持VMM修改虚拟机IA32_SYSENTER_EIP寄存器操作的透明性；2) 对wrmsr指令的写入操作，则直接忽略以防止恶意篡改。最后，监控代理使用深度内联hook技术，对影子内核中的系统调用入口函数Nt!KiFastCallEntry进行挂钩，截获影子内核系统调用最初的控制流程。

当虚拟机进程执行系统调用时，其执行路径首先转移到影子内核，随后被监控代理截获。监控代理读取eax寄存器中的系统调用号等参数，并对进程进行判断：1) 若进程为受保护进程，则执行影子内核中的目标系统调用函数；2) 若进程为普通进程，则返回原内核并执行目标系统调用函数。

2.2　进程内存安全防护

2.2.1　跨进程读写内存防护

Windows系统的跨进程操作带来了非常大的安全隐患，如远程代码注入、第三方进程非法读写等。

进程内存安全防护设计如图3所示，图中，监控代理对影子内核中跨进程读写相关的系统调用函数进行挂钩，如NtReadVirtualMemory、NtWriteVirtualMemory等，指向相应的代理函数进行过滤操作。当某一进程试图读写其他进程内存空间时，代理函数会捕获此事件并进行判断：1) 若主体进程是普通进程，且目标进程是受保护进程，则对读写动作进行混淆或拒绝；2) 主体进程和目标进程都是受保护进程，则放行。

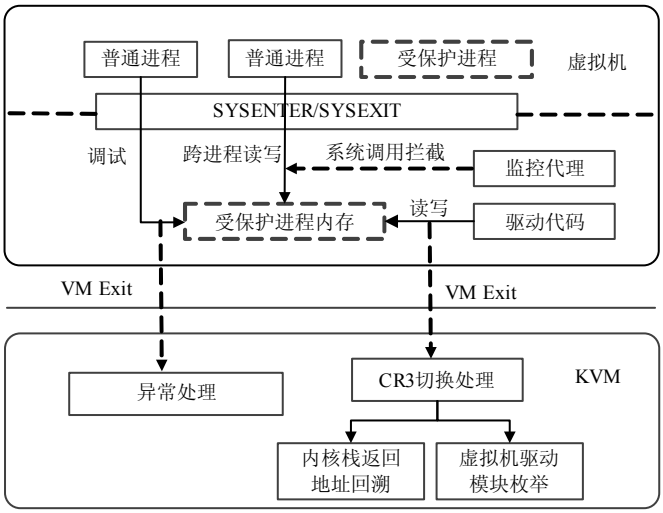

图3　进程内存安全防护设计

2.2.2　驱动模块读写内存防护

现代操作系统中，进程利用页表完成虚拟地址到物理地址的转换，实现对物理内存的访问。每个进程拥有一套独立的页表及唯一的页目录基地址。当进程切换时，CR3寄存器中的页目录基地址会同时进行切换。当恶意代码访问某一进程的内存时，必须切换到该进程对应的页表。

如图3所示，在VMCS中将CR3_target_count置0，CR3切换处理模块即可捕获虚拟机进程切换，读取其CR3值。若虚拟机CR3寄存器值为受保护进程页目录基地址，则通过内核栈帧中存储的ebp值对内核栈进行回溯，获取所有的函数返回地址，并通过虚拟机自省(virtual nachine introspection，VMI)[23-24]获取虚拟机当前所有已加载的驱动模块信息。若回溯过程中所有的返回地址均位于内核空间，说明进程被正常调度；否则根据返回地址所处的地址范围内，定位到读写受保护进程内存的具体驱动模块。

2.2.3　进程反调试防护

操作系统通过#BP异常(int 3)和#DB异常(int 1)提供断点及调试功能。通过对特定进程进行调试，能够动态修改进程代码及数据，分析进程代码逻辑及数据意义，绕过特定进程的安全防护。

如图3所示，将VMCS中Exception Bitmap的BP_VECTOR和DB_VECTOR_置1，若虚拟机中的进程被调试，将产生VM Exit陷入到KVM中的断点及调试异常处理模块。调试异常处理模块对虚拟机CR3寄存器值进行判断：1) 若受保护进程被调试，则强制向进程地址空间写入零值停止其运行；2) 若普通进程被调试，则放行。

2.3　监控代理内存安全防护

监控代理位于Windows虚拟机内核层，功能有：1) 在虚拟机启动过程中，根据原始内核镜像文

件构建影子内核；

2) 对影子内核系统调用入口函数进行拦截，获取最初系统调用执行路径的控制权；

3) 对跨进程读写内存的相关函数进行挂钩并指向代理函数，对跨进程读写操作进行过滤；

4) 当用户指定任意进程进行保护时，通过超级调用传递此进程信息到KVM中。

但是，若同处于内核层的恶意驱动攻击或绕过监控代理，则可能使防护功能失效，因此需要对监控代理进行安全防护。

本文通过构建一个影子监控代理，接管监控代理中函数的执行流程，实现对监控代理的安全防护。与构建影子内核类似，按照PE格式加载监控代理驱动文件构建影子内核之后，需根据原监控代理基地址与影子监控代理基地址的偏移关系，将系统调用代理函数地址重定向到影子监控代理中的系统调用代理函数地址。具体构建过程参考2.1.1节内容。

构造影子监控代理可有效降低防护系统被攻破的可能性。防护系统使用内核非分页池中的影子监控代理驱动代码后，即使当原驱动代码被修改，仍然不会危及到防护系统的功能。同时，非分页池中的内存随机分配，恶意代码难以定位影子监控代理驱动代码，若扫描全部内核地址空间，不仅效率低下，而且将会因访问内核地址空间空洞引发蓝屏。

3 实验结果及分析

VMPPS系统测试侧重两方面：1) 防护能力；2)系统性能。实验环境：硬件配置为华为Tecal RH2288H v2、Intel Xeon E5-2609、2.4 G主频、32 GB内存；宿主机为Ubuntu 12.04、内核版本3.10.1，KVM及QEMU版本分别为3.10.1、1.7.1；虚拟机为32位Windows 7 SP1操作系统。

3.1 防护效果验证

本文收集了多个样本，分别对进程系统调用执行路径安全防护功能、进程内存安全防护功能及监控代理自身安全防护功能的有效性进行了测试，同时自行实现了恶意驱动模块MalDri弥补样本测试功能的不足。测试样本名称、功能及测试结果如表1所示。

表1 测试样本及结果

样本名称	功能	实验结果	
		关闭	开启
ByteHook	挂钩SSDT表项，劫持系统调用	成功	无效
RegKeyHide	挂钩SSDT表项，劫持系统调用	成功	无效
Migbot.sys	深度内联Hook，劫持系统调用	成功	无效
Sysenter.sys	修改SYSENTER_EIP寄存器，劫持系统调用入口	成功	失败
CHeat Engine	查看进程内存，动态修改内存代码	成功	失败
windbg	调试分析进程内存	成功	失败
PCHunter	内核hook检测软件，检测系统调用完整性	成功	无效
MalDri	挂钩系统调用内部调用函数，根据EPROCESS结构体读写进程内存，修改监控代理代码及数据	成功	失败

以WINWORD.EXE进程作为受保护进程，CHeat Engine作为测试样本为例，对原型系统VMPPS有效性的测试过程分为两个步骤：

1) 关闭VMPPS，运行样本CHeat Engine后，可查看并修改WINWORD.EXE进程内存内容，结果如图4所示。其中，右下角标示为WINWORD.EXE进程内存对应的正在编辑的内容，中间标记为对当前WORD代码的反汇编结果。

2) 开启VMPPS，运行样本CHeat Engine，发现无法查看或修改WINWORD.EXE进程内存数据，结果如图5所示。

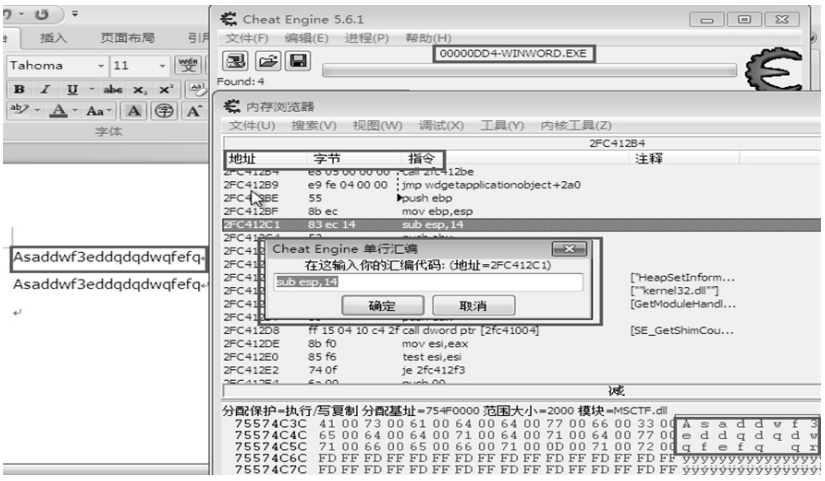

图4 VMPPS关闭进程内存安全测试

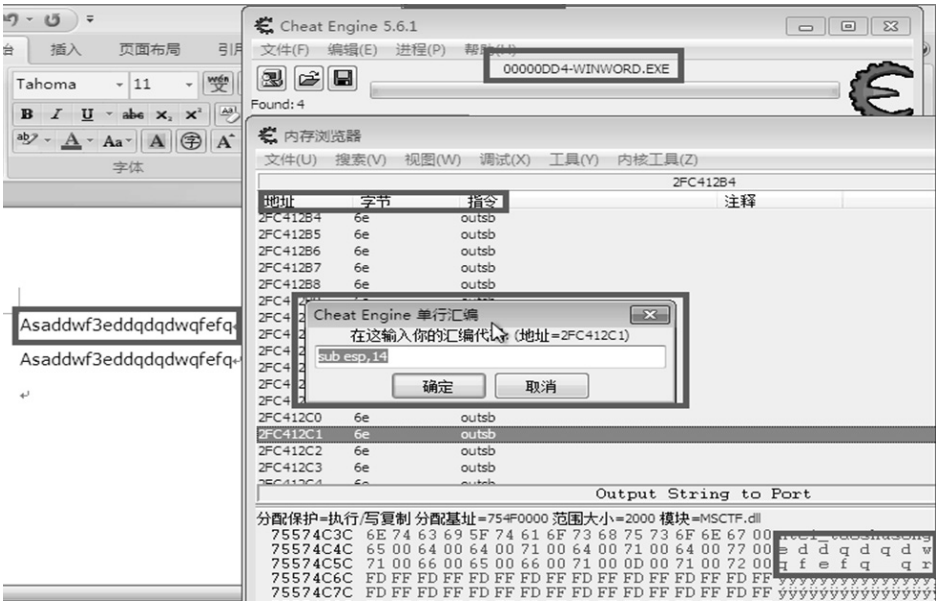

图5　VMPPS开启进程内存安全测试

需要指出，开启原型系统VMPPS时，样本ByteHook、RegKeyHide和Migbot.sys依然能够挂钩原内核系统调用，但影子内核绕过了原内核系统调用的执行流程，因此受保护进程能够正常执行。实验结果表明，VMPPS利用影子内核机制可以透明的为虚拟机提供有效的进程系统调用执行路径的安全防护，通过KVM和监控代理可对进程的内存安全进行防护。

表2　VMPPS与现有方法对比

方法	系统调用执行路径防护	内存防护	进程间通信	任意进程防护	恶意软件不可绕过
Overshadow	×	√	×	√	×
Flicker	×	√	√	×	√
RandSys	√	×	√	√	×
文献[10]	×	√	×	√	√
VMPPS	√	√	√	√	√

本文对前文所述方法进行了多方面的对比，如表2所示。可以看出，现有一些虚拟机进程防护方法存在着不足，如无法全面防护虚拟机系统调用执行路径和进程内存、影响进程通信或可被恶意软件绕过等。而VMPPS有效弥补了上述不足，全面防护了任意虚拟机进程系统调用执行路径和内存，并保证了自身安全性。

3.2　性能测试

原型系统VMPPS在原有KVM虚拟化架构中添加了额外功能模块与处理流程，干预了原系统正常的执行流程，会带来一定的性能损失。本文设计了相关测试，分别测试原型系统对虚拟机进程切换、计算密集型、I/O密集型及整体性能的影响，性能损耗率=(部署VMPPS–未部署VMPPS)/未部署VMPPS，性能测试单位为时间(ms)。进程创建、计算密集型、I/O密集型及整体性能对应的基本测试实例为：创建记事本进程、空循环一百万次、写入128 K文件内容及综合测试(利用可执行文件创建空循环一百万次进程)，测试结果如图6～图9所示。

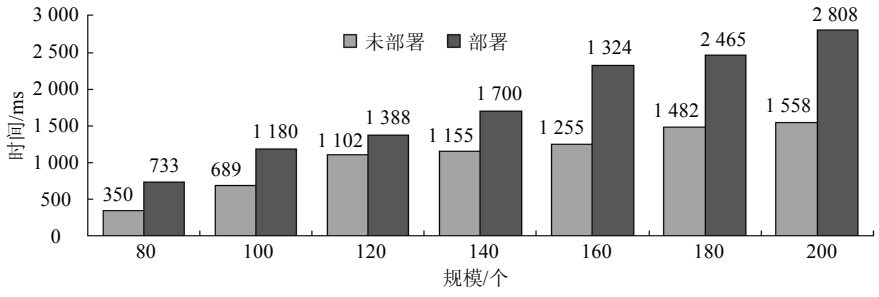

图6　进程创建实验测试数据

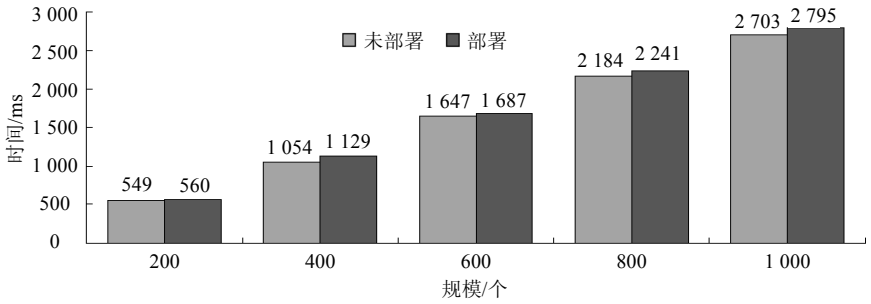

图7 计算密集型实验测试数据

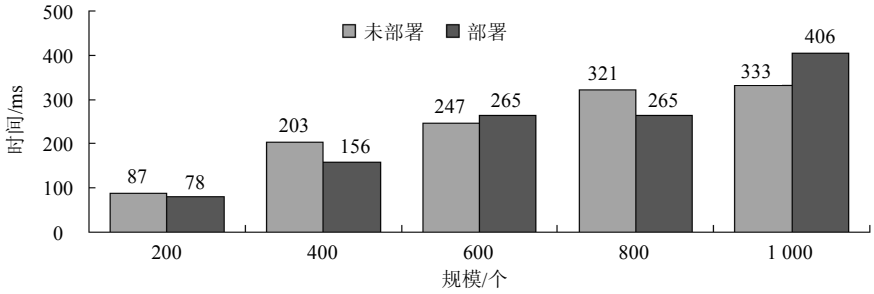

图8 I/O密集型实验测试数据

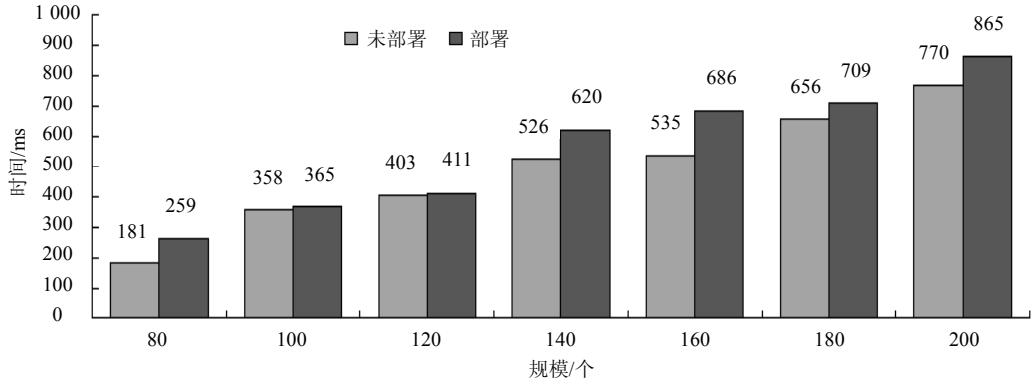

图9 整体性能测试实验数据

根据各项指标测试数据可以看出，对于系统大量的进程创建动作，VMPPS会带来一定的性能损失，各测试项的性能损耗率间的平均值约为0.6，这主要是因为密集的进程创建动作会带来大量的系统调用及进程切换的开销，使原型系统在短时间内达到一个比较高的值，但操作系统正常运行时不会出现这种情况。在平稳运行的系统中，VMPPS对计算密集型和I/O密集型进程的运行几乎没有影响，但是由于I/O缓冲区大小、磁盘运转等系统参数的影响，会使I/O密集型进程的性能出现一定的波动。综合测试的性能消耗主要源于进程创建过程，在可接受范围内，但由于利用可执行文件创建进程涉及I/O过程，所以性能损耗会出现一定的波动。

4 结 束 语

随着云计算的发展，云环境下的虚拟机安全受到越来越多的关注。本文针对Windows进程面临的威胁，提出了一种基于KVM的Windows虚拟机用户进程防护方案。通过构建影子内核，结合硬件虚拟化技术，实现了对虚拟机进程内存和系统调用执行路径的安全防护。实验结果表明，在可接受的性能损失范围内，VMPPS能够有效保护虚拟机用户进程的系统调用执行路径安全和内存安全。下一步将研究如何保证VMM的安全性，并提高VMPPS的效率。

参 考 文 献

[1] FENG Deng-guo, ZHANG Min, ZHANG Yan, et al. Study on cloud computing security[J]. Journal of Software, 2011, 22(1): 71-83.

[2] VALLS-PRIETO J. Digital crime, cyberspace security, and information assurance[M]. Pennsylvania, USA: IGI Global, 2014.

[3] ZAYTSEV O. Rootkits, spyware/adware, keyloggers and backdoors: Detection and neutralization[M]. [S.l.]: A-List

Publishing, 2006.

[4] SESHADRI A, LUK M, QU N, et al. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes[J]. ACM SIGOPS Operating Systems Review, 2007, 41(6): 335-350.

[5] 王惠莅, 杨晨, 杨建军. 美国NIST云计算安全标准跟踪及研究[J]. 信息技术与标准化, 2012(6): DOI: 10.3969/j.issn.1671-539X.2012.06.013.
WANG Hui-li, YANG Chen, YANG Jian-jun. Research on clouds computing security standards of NIST[J]. Information Technology & Standardization, 2012(6): DOI:10.3969/j.issn.1671-539X.2012.06.013.

[6] 孟庆倩, 李清宝, 魏珉. 基于Windows环境进程监控的设计与实现[J]. 信息工程大学学报, 2007, 8(1) : 26-29.
MENG Qing-qian, LI Qing-bao, WEI Min. Design and realization of process-monitoring based on windows[J]. Journal of Information Engineering University, 2007, 8(1): 26-29.

[7] 张涛, 牛伟颖, 孟正, 等. 基于Windows内核模式下进程监控的用户权限控制系统设计与实现[J]. 信息网络安全, 2014(4): 13-19.
ZHANG Tao, NIU Wei-ying, MENG Zheng, et al. The design and implemention of windows system access control based on process monitoring[J]. Netinfo Security, 2014(4): 13-19.

[8] JIANG X, WANG H J, XU D, et al. Randsys: Thwarting code injection attacks with system service interface randomization[C]//26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007). Beijing, China: IEEE, 2007: 209-218.

[9] CHEN X, GARFINKEL T, LEWIS E C, et al. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems[J]. ACM SIGOPS Operating Systems Review, 2010, 42(2): 2-13.

[10] YANG J, SHIN K G. Using hypervisor to provide data secrecy for user applications on a per-page basis[C]// Proceedings of the fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York, USA: ACM, 2008: 71-80.

[11] SILAKOV D V. Using virtualization to protect application address space inside untrusted environment[J]. Programming and Computer Software, 2012, 38(1): 24-33.

[12] MCCUNE J M, PARNO B J, PERRIG A, et al. Flicker: an execution infrastructure for TCB minimization[C]//ACM SIGOPS Operating Systems Review. New York, USA: ACM, 2008, 42(4): 315-328.

[13] CHEN H, ZHANG F, CHEN C, et al. Tamper-resistant execution in an untrusted operating system using a virtual machine monitor[J]. Technical Report, 2007, 12(3): 123-138.

[14] TA-MIN R, LITTY L, LIE D. Splitting interfaces: Making trust between applications and operating systems configurable[C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Nerkeley, USA: USENIX Association, 2006: 279-292.

[15] WOJTCZUK R, RUTKOWSKA J, TERESHKIN A. Xen owning trilogy[EB/OL]. [2015-03-20]. http:// invisiblethingslab.com /itl/Resources.html.

[16] Rapid7. Advanced exploitation of virtualBox 3D acceleration VM escape Vulnerability (CVE20140983) [EB/OL]. [2015-03-20]. https://vulners. com/metasploit/ MSF:EXPLOIT/WINDOWS/LOCAL/VIRTUAL_BOX_OPENGL_ESCAPE.

[17] WANG Z, JIANG X. Hypersafe: a lightweight approach to provide lifetime hypervisor control-flow integrity[C]// 2010 IEEE Symposium on Security and Privacy (SP). California, USA: IEEE, 2010: 380-395.

[18] YANG W, MA L. HyperDomain: Enabling inspection of malicious VMM's misbehavior[C]//2012 IEEE 11th International Conference onTrust, Security and Privacy in Computing and Communications (TrustCom). Washington, USA: IEEE, 2012: 588-595.

[19] WANG J, STAVROU A, GHOSH A. HyperCheck: a hardware-assisted integrity monitor[C]//Recent Advances in Intrusion Detection. Berlin Heidelberg: Springer, 2010: 158-177.

[20] Intel. Intel® virtualization technology (Intel® VT) [EB/OL]. [2015-03-20]. http://www.intel. com/content/ www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html.

[21] AMD. Secure virtual machine architecture reference manual[EB/OL]. [2015-03-20]. http://www.mimuw. edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf.

[22] Intel. 64-ia-32-architectures- software-developer- manual-325462[M/OL]. Washington, USA: Intel, 2014. [2015-03-20]. http://www.intel.com/content/www/us/en/ processors/architectures-software-developer-manuals.html.

[23] PAYNE B D, De CARBONE M D P, LEE W. Secure and flexible monitoring of virtual machines[C]//Twenty-Third Annual Computer Security Applications Conference. Florida, USA: IEEE, 2007: 385-397.

[24] JIANG X, WANG X, XU D. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction[C]//Proceedings of the 14th ACM Conference on Computer and Communications Security. New York, USA: ACM, 2007: 128-138.

编 辑 漆 蓉