基于影子内存的无代理虚拟机进程防护

陈兴蜀，陈蒙蒙，金 鑫

(四川大学计算机学院 成都 610065)

【摘要】为了提高虚拟机中进程的安全性，避免系统调用表SSDT和系统调用执行流被恶意挂钩，提出一种基于影子内存的无代理虚拟机进程防护方案。首先利用VMM的高特权级在虚拟机的非换页内存区透明构建一块影子内存，通过向影子内存透明注入SSDT和跳转函数，构建全新SSDT和系统调用执行流，保证SSDT和系统调用执行流的完整性。通过主动挂钩影子内存中的SSDT，利用硬件虚拟化的自动陷入机制检测进程的敏感行为，在VMM中过滤针对受保护进程的非法操作，实现无代理的进程防护。实验结果表明，该方案可以有效地对虚拟机中指定进程进行防护并过滤大部分的rootkit攻击，对虚拟机性能的影响在3%以下。

关 键 词 无代理；进程；系统调用；虚拟机；VMM

中图分类号 TP309 **文献标志码** A doi:10.3969/j.issn.1001-0548.2018.01.012

Shadow Memory-Based Agentless Virtual Machine Process Protection

CHEN Xing-shu, CHEN Meng-meng, and JIN Xin

(College of Computer, Sichuan University Chengdu 610065)

Abstract To improve security of process in virtual machine (VM) and avoid system service descriptor table (SSDT) and system call execution path being hooked, a agentless method based on shadow memory of protecting process security in VM is proposed. First, a block of shadow memory is constructed in nonpaged pool of VM by using of high privilege level of virtual machine manager (VMM), then new system service descriptor table (SSDT) and system call execution path are injected to shadow memory. The process sensitive behavior is detected by using of characteristic of hardware virtualization and hook technology, and the invalid operation to targeted process is filtered in VMM so as to implement protecting process security without agent in VM. Analysis and test results show that almost all the attacks from rootkits can be prevented, and the targeted process in VM can be protected well with almost no performance loss.

Key words agentless; process; system call; virtual machine; VMM

随着云计算的发展，云平台的安全问题成为业界关心的核心问题[1]。由于云平台资源、信息高度集中，虚拟机的安全直接威胁着平台的安全[2]。MITC攻击[3]就是一种新型的云环境下利用虚拟机窃取云平台数据的攻击，此外还有各种勒索软件、Cloud weaponization等均可通过虚拟机对云平台造成攻击，窃取租户的数据，甚至破坏整个平台[4]。传统主机面临的安全威胁，在虚拟化环境下的虚拟机同样存在[5]。目前针对操作系统的恶意软件、病毒、rootkit层出不穷，严重威胁着操作系统的安全。进程作为操作系统资源分配和代码执行的基本单位，一旦被攻击，会直接威胁操作系统的安全[16]。因此，研究虚拟机中的进程防护意义重大。

目前，针对虚拟机的进程防护大部分是从全局的角度去检测恶意进程，缺乏针对特定进程的防护方案。文献[6]通过检测CR3寄存器的变化拦截进程切换动作，在VMM中调用第三方的安全模块对进程进行分析。这种方案一方面只能在恶意进程运行一段时间后分析出恶意进程的存在，存在滞后性，不能针对特定进程做防护；另一方面，每次进程切换都会发生陷入，然后传递信息到宿主机用户空间进行分析，性能损耗严重。文献[7]通过虚拟机内部代理驱动设置KiFastCallEntry和KiSystemService函数中调用KiSystemServiceRepeat函数的点为0xFFFFFFFF，使该函数被调用的时候发生陷入，在VMM中做陷入处理分析。此方式虽然可以保护指定进程，但是一方面存在虚拟机代理，代理本身安全无法保证；另一方面，陷入次数过于频繁，同样存

收稿日期：2017－03－17；修回日期：2017－06－20
基金项目：国家自然科学基金(61272447)
作者简介：陈兴蜀(1969－)，女，教授，主要从事信息安全、云计算安全方面的研究.

在性能问题。文献[8]通过设置SSDT(system service descriptor table)所在页面为只读页面，在发生直接HOOK SSDT的时候会发生陷入，之后在VMM中做处理。文中针对的是对rootkit的检测，且仅仅针对通过挂钩SSDT隐藏自身这一个特征做处理，一方面关注点太小，另一方面也不适应于特定进程防护的场景。文献[9-10]从全局的角度对虚拟机完整性做监控，侧重点不在于防护。文献[11]通过拦截CR3切换动作，在VMM中操作受保护进程的页表，对指定进程内存进行防护。这种方式同样存在文献[6]提到的性能问题。文献[12]通过构建一个影子内核，对所有系统服务重定向，可以较好对特定进程进行防护，但是其依赖于虚拟机内部的代理驱动，代理驱动自身安全存在威胁。

针对以上存在的问题，本文提出一种基于影子内存的无代理[13]虚拟机进程防护系统AVMPPS(agentless process protection system based on shadow memory)，并针对windows虚拟机进行了实现。该系统通过影子内存保证SSDT和系统调用执行流的完整性，在虚拟机外部分析处理虚拟机内部进程敏感行为，保护虚拟机中特定进程不被非法操作。基于本文方案无代理特性，规避了代理自身带来的安全威胁和资源消耗。并且该方案仅对特定系统调用做陷入设置，减少了模式切换次数，大幅减少了性能的损耗。

1　AVMPPS系统设计

AVMPPS系统总体架构如图1所示。

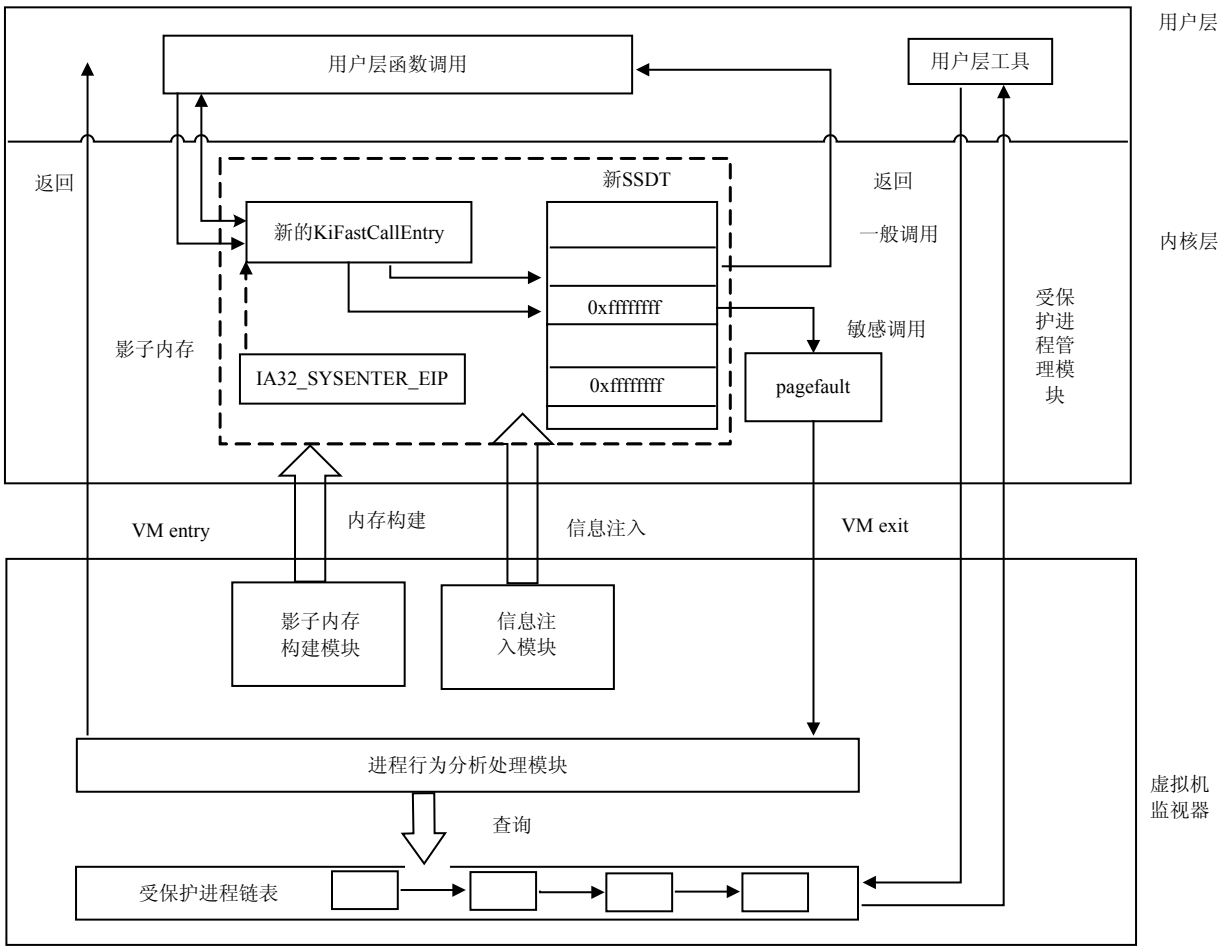

图1　AVMPPS系统结构

AVMPPS系统分为4个模块：受保护进程管理模块、影子内存分配模块、信息注入模块和进程行为分析处理模块。在AVMPPS系统初始化阶段，首先在VMM中由影子内存分配模块在虚拟机非换页内存区构建一份纯净的影子内存作为后续信息注入的页面，该影子内存位于虚拟机的非换页内存区，页表在虚拟机初始化内存时已经建立，且在虚拟机运行期间不会被换出，可以被虚拟机直接访问；接着由信息注入模块向影子内存注入信息构建全新的SSDT和系统调用执行流，其中进程敏感行为的相关系统调用地址被设置成非法地址，这样在用户请求该系统调用的时候会发生取指令类型的pagefault。

注入完成后在VMM中设置系统调用入口为新构建的KiFastCallEntry，实现系统调用执行流的重定向；在VM-entry之前根据VMCS结构设置取指令类型的pagefault发生陷入，从而对进程敏感行为做拦截处理，最后设置IA32_SYSENTER_EIP寄存器的读写陷入，屏蔽虚拟机中对该寄存器的读写行为，隐藏实际的系统调用入口。在AVMPPS系统运行阶段，当虚拟机内部进程执行敏感行为，触发SSDT中的钩子继而陷入到VMM中，进程行为分析处理模块处理该陷入，根据受保护进程链表对进程敏感行为过滤，禁止受保护进程被第三方进程操作，保障受保护进程的安全。受保护进程链表的维护则由受保护进程管理模块实现，该模块通过超级系统调用机制增加、删除、查询当前虚拟机的受保护进程。

2 AVMPPS系统的关键技术

2.1 受保护进程管理

支持硬件虚拟化的CPU会提供一种特殊的操作模式即VMX操作模式，具体分为root模式和non-root模式。虚拟机运行在non-root模式，而VMM则运行在root模式。通过VMX快速系统调用指令VMCALL，CPU可以无条件的从non-root模式退出到root模式，将控制权交还给VMM[14]。

利用这一特性，在虚拟机内部构建用户层工具，直接和用户交互，完成受保护进程信息的构建。由于虚拟机和主机是两个独立的操作系统，不能通过普通堆栈进行参数传递，在用户层工具构建好受保护进程结构后，借助寄存器向VMM传递进程结构的地址，然后执行VMCALL。VMM获得控制权后，根据相应寄存器中的地址，透明读取虚拟机内存中的进程名称，并构建进程保护结构体，作为一个节点插入到受保护进程链表中，链表头保存在VMM中对应虚拟机的KVM结构体中(删除和查询操作类似)。整个结构如图2所示。

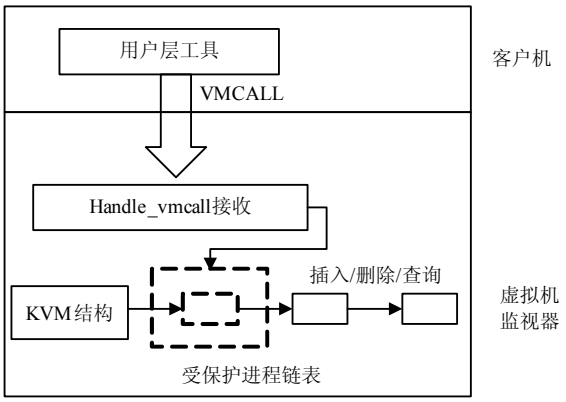

图2 受保护进程管理模块架构

2.2 影子内存构建

构建影子内存需要通过分析虚拟机内存管理架构，在VMM中按照虚拟机正常内存分配流程，获取一块可用的内存。这块内存需要保证3点：

1) 这块内存虚拟机不能已经使用；

2) 这块内存虚拟机以后不能被再次分配出去；

3) 这块内存的页表必须已经建立，且不会被换出到外存。基于以上要求，所构建的内存必须位于虚拟机的非换页内存区。

2.2.1 Windows内存管理架构

Windows 7及其以后的系统采用非一致性内存架构NUMA(non-uniform memory access)来管理内存。这种架构下，每个CPU被分配属于自己的一块内存称之为local memory，处理器访问属于自己的local memory的速度是最快的，只有当local memory不足的时候才会访问其他CPU的memory。CPU通过node和内存相关联，每个node负责管理一定的内存区，Windows对node实现为KNODE结构，每个KNODE管理一组内存区，包括换页和非换页内存。对于非换页内存的管理，Windows利用空闲页面自身，构建了3条链表，根据连续页面的数量，分为单页面链表、双页面链表和多页面链表。KNODE结构中的NonPagedPoolSListHead字段保存这些链表头[15-18]。

2.2.2 影子内存获取

X86架构下，IA32_SYSENTER_EIP寄存器保存系统调用入口函数的地址，当该寄存器被填充时意味着系统调用机制开始启用。同时，这也是构建影子内存的时间点，此时系统核心部分初始化刚刚完毕，驱动和应用程序尚未加载，可以保证在系统调用机制启用之前完成系统调用流的重定向。在Windows虚拟机中，当CPU运行在内核模式时，FS寄存器指向处理器全局结构KPCR，在KPCR结构的0x5ec偏移处，保存当前处理器的KNODE结构。KNODE结构中保存有节点关联的非换页内存页链表表头NonPagedPoolSListHead数组。链表通过MMFREE_POOL_ENTRY结构连接各个非换页内存页面，该结构位于页面的头部，结构的地址就是非换页内存页的地址。需要注意的是获取操作对于虚拟机要保持透明，因此要使表头指针_SLIST_HEADER指向链表第二个页面，以保证本次获取的页面之后不会被系统再次分配。链表操作结构如图3所示。

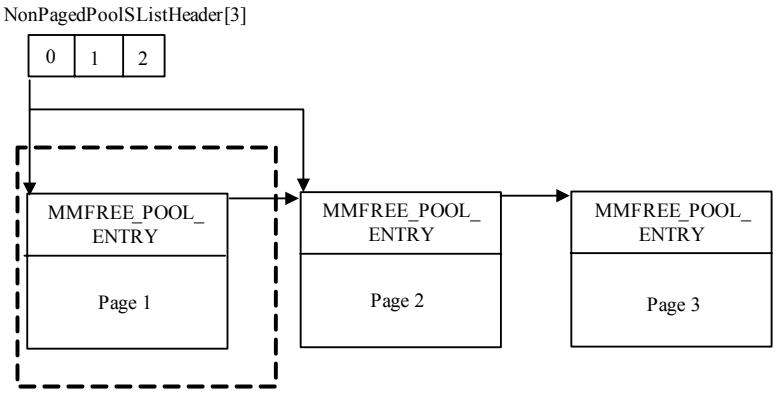

图3　非换页内存页链表结构

2.3　虚拟机信息注入

该模块存位于VMM中，在影子内存中构建全新的SSDT和系统调用执行流。注入信息在影子内存中的结构安排如图4所示。

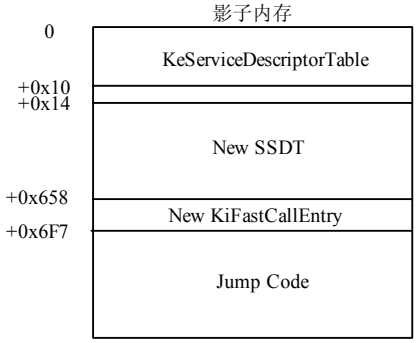

图4　影子内存组织结构

2.3.1　SSDT信息注入

虚拟机通过一个全局的SSDT控制结构KeServiceDescriptorTable操作SSDT，为了保障新的SSDT的隐蔽性，KeServiceDescriptorTable结构需要重新构造。且为了对虚拟机产生尽可能小的影响，这里仅仅设置结构中的ssdt指针，其他的不做修改。

在线程结构Kthread中有个ServiceTable字段，记录其适用的SSDT。若当前线程是GUI线程，则ServiceTable指向KeServiceDescriptorTableShadow，否则指向KeServiceDescriptorTable。在AVMPPS系统部署时，还是内核初始化阶段，当前线程必定指向KeServiceDescriptorTable。根据此结构，读取虚拟机内部实际的SSDT内容，按照顺序写入到影子内存中。为了区分，在KeServiceDescriptorTable结构和具体SSDT之间放置4个字节做隔离。

2.3.2　系统调用执行流

1) 系统调用分发函数构建

如图1所示，IA32_SYSENTER_EIP寄存器保存有系统调用入口函数的地址，在VMM中通过该寄存器即可获取系统调用入口函数的地址。这里不需要构建完整的函数，只需要保证具体系统调用被执行之前的代码的完整性即可。在Windows下从函数开始到具体call指令执行，包含159个字节，写入影子内存的正是这159个字节，在call指令执行后，要使执行流转回到原始的分发函数中。

2) SSDT重定向

原始的系统调用入口函数引用的是指向原始SSDT的控制结构KeServiceDescriptorTable，要让影子内存中的调用函数使用新的SSDT，需要对函数调用SSDT的时候进行重定向。

由于AVMPPS系统未在虚拟机内部安装任何代理软件，只能在VMM层根据函数结构，在适当的位置注入跳转代码。由于GUI线程和普通线程使用同一套系统调用机制，而GUI线程使用的是KeServiceDescriptorTableShadow，故需要添加验证逻辑。碍于字节数的限制，本方案借助二级跳转机制，构建两块内存分别用于逻辑验证与SSDT地址设置。这里所有的代码均位于影子内存中，未对原始函数做任何修改，有效避免了恶意程序对函数进行溯源。具体的执行流如图5所示。

内核空间对系统调用的使用是通过ZW*函数实现的。最终会调用KiSystemService函数，该函数并不直接使用KeServiceDescriptorTable，而是在中途跳转到了KiFastCallentry函数的指定位置，故这里要对KiSysemService函数进行深度内联，让其跳转到前面构造的新KiFastCallentry函数。为了增强跳转模块的隐蔽性，AVMPPS系统在原始函数JMP指令的前面4个字节执行挂钩，整个跳转逻辑如图6所示。

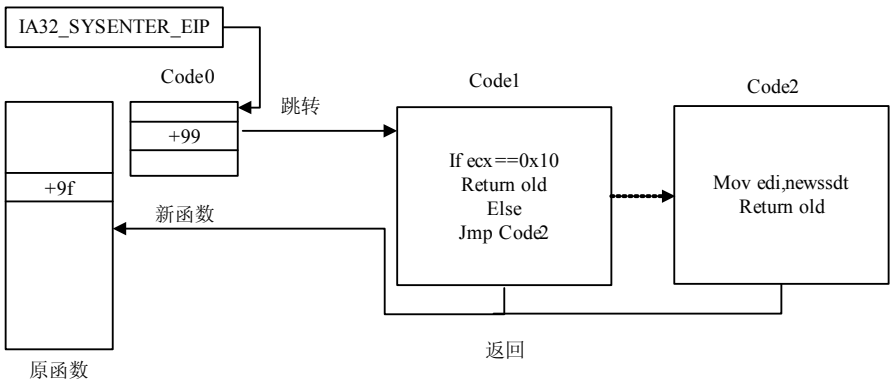

图5 KiFastCallEntry跳转逻辑

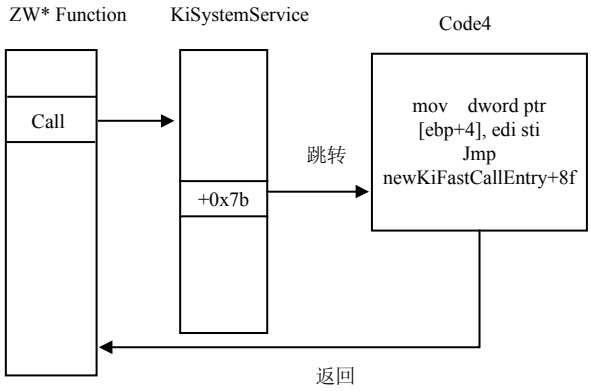

图6 KiSystemService 跳转逻辑

2.3.3 系统调用函数地址的隐藏

目前虚拟机虽然不能直接HOOK实际的SSDT和系统调用入口函数，但恶意rootkit还可能对系统调用函数执行inline HOOK。虚拟机中的rootkit获取系统调用函数地址大致有两种方式：通过SSDT或者通过内核映像文件。为了阻断rootkit获取系统调用函数地址，防止其被inline HOOK；AVMPPS系统对原始的SSDT进行抹除。另外，把打开文件的系统调用NTCreateFile加入敏感行为列表，由VMM过滤打开的文件。具体处理在进程行为分析处理模块。这样就阻断了绝大部分rootkit获取原始调用地址的途径，对系统调用执行流进行有效防护。

2.3.4 系统调用入口隐藏

如图1所示，系统调用入口保存在IA32_SYSENTER_EIP寄存器中，该寄存器属于MSR寄存器，Intel为MSR寄存器的读写陷入设置了开关[14]。Intel规定，在对MSR寄存器进行读写时，如果满足以下条件，可以发生陷入：1) 寄存器地址范围在00000000～00001FFFFH；2) Bit n(MSR寄存器的地址)在Read bitmap/Write bitmap中为1；3) VM-execution中Use MSR bitmap控制位为1。

基于以上条件，在VM-entry前，设置相应的位，让虚拟机在对IA32_SYSENTER_EIP寄存器读写的时候陷入到VMM中，由VMM做出处理。具体的处理方案为：

RDMSR(读取)：给虚拟机返回原始入口函数地址，隐藏实际的函数地址。

WRMSR(写入)：VMM不写入，但是正常返回虚拟机，对虚拟机进行欺骗。

2.4 进程行为分析处理

该模块位于VMM中。通过设置影子内存中新的SSDT中的敏感系统调用地址为非法地址，在VMM中设置取指令类型的pagefault发生陷入，在VMM的异常处理函数handle_exception中处理陷入，对目标进程和源头进程作出分析，屏蔽针对受保护进程的敏感操作。

2.4.1 敏感调用捕获

根据Intel手册[14]，pagefault异常的陷入是可选陷入，默认pagefault由虚拟机自己处理，通过设置VMCS结构中的相应字段，可以令某种类型的pagefault透明陷入，由VMM处理该异常。是否陷入取决于以下两个公式：

$$\text{PFEC} \,\&\, \text{PFEC\_MASK} = \text{PFEC\_MATCH} \quad (1)$$

$$\text{ExceptionBitmap} >> 0\text{x}14 = 1 \quad (2)$$

式中，PFEC是发生异常的错误码；PFEC_MASK和PFEC_MATCH是VMCS结构中的两个字段，用来控制具体pagefault异常的陷入；式(2)中的ExceptionBitmap是VMCS结构中的一个32位的字段，每位控制一种异常，位14控制pagefault异常。

当虚拟机发生pagefault异常时，处理器根据式(1)判断是否成立，如果不成立，由虚拟机内部的异常处理机制处理该异常；否则检查式(2)是否成立，如果式(2)成立，则发生VM-exit；否则，虚拟机自己处理。

Intel官方定义的pagefault异常有5种：存在位异

常、read/write异常、模式异常、保留位异常和取指令异常，由PFEC的最低5位标识，取指令异常对应于第5位。根据式(1)对PFEC_MASK和PFEC_MATCH进行设置，只让虚拟机在发生取指令异常的时候陷入到VMM中，其他无关异常则在虚拟机内部处理，减少虚拟机陷入次数，从而降低性能损耗。

2.4.2 敏感行为处理

基于上述设置，在进程发起敏感行为时会陷入到VMM中，由于本系统采用无代理方式，底层只能获取原始二进制代码，要准确识别出具体的进程，需要根据特定的参数传递规范，在VMM中透明读取虚拟机内核调用栈，解析出具体的调用句柄参数，根据进程句柄解析当前进程句柄表，获取进程对象。

1) 调用参数解析

操作系统禁止内核空间和用户空间共享堆栈，在具体的系统调用函数被执行之前，位于用户空间堆栈中的参数会被复制到内核空间堆栈，ESP寄存器指向内核栈顶。Windows针对进程的操作有两个步骤：打开进程、操作进程。首先要通过类似OpenProcess函数打开指定进程获取一个句柄，然后根据此句柄对进程进行特定的操作，进程句柄会作为进程相关系统调用的首个参数被传递。Windows中的参数传递根据stdcall参数传递规范[19]，遵循从右至左压栈的顺序。基于此，在进入VMM的陷入处理流程后，可以通过ESP寄存器定位虚拟机内核栈顶，进而获取相关参数。

2) 进程句柄解析

Windows中进程通过句柄表管理自己打开的所有对象，句柄值便是对应对象在句柄表中的索引。在VMM中通过FS寄存器可以获取当前处理器结构KPCR，进而获取当前线程Kthread，Kthread中保存有所属进程对象Eprocess的指针。Eprocess中保存有当前进程句柄表的指针ObjectTable是一个指向HADLE_TABLE结构的指针，HADLE_TABLE确定句柄表的级数和基地址，具体的句柄表表项是HADLE_TABLE_ENTRY结构，其中保存有对应的对象头OBJECT_HEAD的地址。根据句柄表结构，利用句柄值索引句柄表得到具体的对象头(OBJECT_HEAD)，对象体位于对象头的0x18偏移处。2层句柄表解析架构如图7所示(1层和3层类似)。

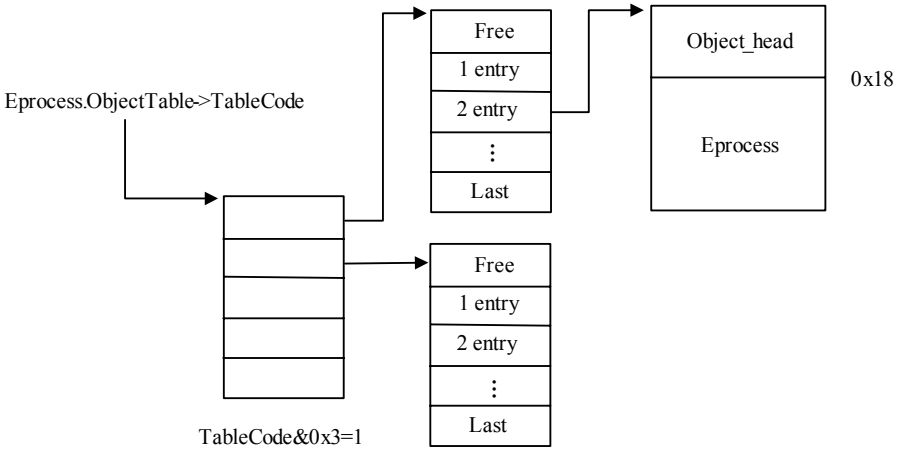

图7　句柄表解析架构

3) 进程处理

根据最新一期的微软安全报告[4]，一个进程可以通过以下行为攻击进程或者系统。主要有恶意终止、代码注入、内存读取、调试攻击等。同时，为了阻断虚拟机内部获取系统调用函数的路径，打开文件操作被加入敏感行为列表。基于此，AVMPPS系统挂钩了以上行为对应的核心系统调用，分别是NTTerminateProcess、NTReadVirtualMemory、NTWriteVirtualMemory、NTCreateProcess、NTDebugActiveProcess和NTCreateFile函数。这样在上述行为发生时，会触发SSDT中的钩子进而陷入到VMM中。

在VMM中维护一条受保护进程链表，在虚拟机中进程敏感行为被捕获而陷入到VMM后，若是针对进程的调用，则根据此链表对当前进程进行分析，若操作对象是受保护进程，且源头进程是第三方进程，则禁止本次调用，否则就允许调用。对于通过打开程序可执行文件进行调试的陷入，需要分析文件名和源头进程。对于打开系统镜像文件的调用陷入，直接拒绝。针对非法操作，AVMPPS系统通过VMCS结构获取当前指令的长度intr_length，令当前虚拟机指令寄存器(EIP)后移intr_length个字节，从而

跳过当前的call指令，并向EAX注入错误码，通知客户机系统调用执行失败。而对于普通操作，则根据VMM中保存的完整SSDT，向EIP注入正确的系统调用地址，表示允许当前调用。

3 系统实验结果及分析

为了验证AVMPPS系统的防护效果并且了解AVMPPS系统对虚拟机性能的影响，针对AVMPPS系统的功能和性能分别作了测试。Windows 7系统自发布以来稳定性和用户量均位于Windows各个版本前列，因此选取Windows 7系统进行测试。

实验环境：宿主机：Intel Core i5四核处理器、64位CentOS 7操作系统3.10.1版本内核；虚拟机：双核2 G内存、32位windows 7操作系统；VMM：KVM 3.10.1；Qemu：1.7.1；Libvirt：1.2.17。

3.1 功能测试

为了验证AVMPPS系统的防护效果，采取多个样本分别针对进程运行防护、系统调用执行流防护、SSDT防护做测试，样本原理介绍如表1所示。同时为了凸显本系统的优势，和已有的优秀方案做对比，测试结果表明，AVMPPS可以有效地对指定进程进行保护，同时保护SSDT自身和系统调用执行流的完整性。如表2所示。

表1 AVMPPS防护功能测试

恶意样本	操作
Shutdown.sys/fin.exe	终止特定进程
Windbg/OD	恶意调试/恶意附加
Cheategine/inject.exe	读/写进程内存
360/腾讯管家	HOOK入口函数
SafeCrack	内联HOOK SSDT
FileFilter	隐藏恶意文件
Sysenter.sys	修改系统调用入口
GetSSDT.sys	通过导出结构，获取SSDT
HidePro	直接挂钩SSDT，隐藏自身
PCHunter	获取原始SSDT函数地址

表2 AVMPPS防护功能测试

恶意样本	方法		
	文献[7]	文献[12]	AVMPPS
Shutdown.sys/fin.exe	失败	失败	失败
Windbg/OD	成功	失败	失败
Cheategine/inject.exe	失败	失败	失败
360/腾讯管家	失败	失败	失败
SafeCrack	成功	失败	失败
FileFilter	失败	失败	失败
Sysenter.sys	成功	失败	失败
GetSSDT.sys	成功	失败	失败
HidePro	失败	成功	失败
PCHunter	成功	失败	失败

3.2 安全性测试

作为安全防护系统，自身安全尤为重要，如果系统运行期间代理驱动被卸载，防护功能就荡然无存。因此借助于PCHunter和Kmdmanager工具针对已有的基于代理驱动的方案和本方案进行安全性对比试验，测试结果如表3所示。

表3 AVMPPS安全性测试

方法	项目			
	文献[7]	文献[8]	文献[12]	AVMPPS
代理检测	成功	成功	失败	失败
代理卸载	成功	成功	成功	失败

通过表3可以发现，文献[7-8]的方法未对代理驱动作任何保护，随时可以被检测和卸载。文献[12]对驱动作了隐藏设置，可以躲避检测，但是如果事先知道驱动名称，还是可以强制卸载；AVMPPS未使用任何代理驱动，因此不可能被检测到，且AVMPPS基于影子内存，类似于系统预留的内存，任何驱动和检测软件都无法获取其有效地址，因此AVMPPS的安全性可以得到保证。

3.3 性能测试

AVMPPS系统在虚拟机系统调用机制启用时部署，一旦部署成功，之后的性能损耗基本在于进程特定敏感行为调用的陷入，由于AVMPPS系统对无关调用不作陷入设置，最大程度上减少了模式切换的次数，所以对虚拟机性能的影响极其微小，基本可以忽略。以下利用鲁大师从CPU、内存、I/O三方面分别测试10次取平均值来测试AVMPPS系统对虚拟机的影响，并和其他无代理方案进行对比。测试结果表明，AVMPPS系统部署成功后对CPU的影响在2%左右，而对内存和I/O的影响都低于0.5%，性能损耗方面远远低于其他的无代理方案，如表4所示。

表4 AVMPPS性能测试展示

方法	项目			
	CPU/%	内存/%	I/O/%	平均/%
AVMPPS	2.1	0.1	0.3	0.83
文献[6]	5.3	2.5	3.1	3.9
文献[11]	6.2	3.5	2.6	4.1

4 结 束 语

综上所述，为了解决当前虚拟机进程防护方案有代理、高能耗的问题，本文提出基于影子内存的无代理虚拟机进程防护方案。通过透明构建的影子内存保证SSDT、系统调用执行流和核心代码的完整

性。利用硬件虚拟化的陷入机制过滤进程敏感行为，为受保护进程提供运行时防护。由于本文系统不对无关行为作陷入设置，保证进程运行安全的同时，大幅减少了虚拟机陷入次数，降低了虚拟机性能损耗。目前AVMPPS系统在分配虚拟机内存时，对具体系统版本有一定的依赖，后续要研究一种较为普适的方法获取虚拟机内存，这样就在较大程度上减少了对虚拟机系统版本的依赖。

参 考 文 献

[1] 林闯, 苏文博, 孟坤, 等. 云计算安全: 架构、机制与模型评价[J]. 计算机学报, 2013, 36(9): 1765-1784.
LIN Chuang, SU Wen-bo, MENG Kun, et al. Cloud computing security: Architecture, mechanism and medeling [J]. Chinese Journal Of Computers, 2013, 36(9): 1765-1784.

[2] SCHWARZKOPF R, SCHMIDT M, STRACK C, et al. Increasing virtual machine security in cloud environments[J]. Journal of Cloud Computing, 2012, 1(1): 1-12.

[3] Imperva. Man in the cloud (MITC) attacks[EB/OL]. [2015-09-06]. https://www.imperva.com/docs/HII_Man_In_The_Cloud_Attacks.pdf.

[4] ERIC A, VOLV G, CARMEN L, et al. Microsoft security intelligence report[EB/OL]. [2017-03-17]. https://www.microsoft.com/en-us/security/intelligence-report.

[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] 崔竞松, 向浩, 郭迟, 等. 基于 KVM 的 windows 客户机进程查杀技术[J]. 四川大学学报: 工程科学版, 2014, 46(6): 7-13.
CUI Jing-song, XIANG Hao, GUO Chi, et al. Online anti-virus technology of process running on Windows VM based on KVM[J]. Journal of SiChuan University (Engineering Science Edition), 2014, 46(6): 7-13.

[7] 彭春洪. 基于KVM虚拟机的恶意行为检测系统设计与实现[D]. 成都: 电子科技大学, 2015.
PENG Chun-hong. The design and implementation of the malicious behavior detecting system based on KVM virtual machine[D]. Chengdu: University of Electronic Science and Technology of China, 2015.

[8] 张文静. 基于 Xen 的 Rootkit 检测技术研究[D]. 西安: 西安电子科技大学, 2014.
ZHANG Wen-jing. Research on rootkit detection techonlogy based on Xen[D]. Xi'an: XiDian University, 2014.

[9] 张贵民. 基于 Intel VT 的内核完整性监控技术研究[D]. 郑州: 解放军信息工程大学, 2014.
ZHANG Gui-min. Research on kernel intergrity monitoring technology based on Intel VT[D]. Zhengzhou: PLA Information Engineering University, 2014.

[10] 黄啸. 基于虚拟化的内核监控模型研究与实现[D]. 南京: 南京大学, 2015.
HUANG Xiao. Research and implementation of kernel monitoring model based on virtualization[D]. Nanjing: Nanjing University, 2015.

[11] 李佳瑶. 基于虚拟化的 Wndows 进程内存保护研究[D]. 南京: 南京大学, 2014.
LI Jia-yao. Research on memory protection of Windows process based on virtualization[D]. Nanjing: Nanjing University, 2014.

[12] 陈兴蜀, 赵成, 陶术松. 基于KVM的windows虚拟机用户进程防护[J]. 电子科技大学学报, 2016, 46(6): 950-957.
CHEN Xing-shu, ZHAO Cheng, TAO Shu-song. KVM-based Windows virtual machine user process protection[J]. Journal of University of Electronic Science and Technology of China, 2016, 46(6): 950-957.

[13] DUNLAP G W, KING S T, CINAR S, et al. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay[J]. Acm Sigops Operating Systems Review, 2002, 36(SI): 211-224.

[14] Intel. 64-ia-32-architectures-software-developer-manual-325462[EB/OL]. [2015-03-20]. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

[15] MANDT T. Kernel pool exploitation on Windows 7[EB/OL]. [2017-03-17]. https://media.blackhat.com/bh-dc-11/Mandt/BlackHat_DC_2011_Mandt_kernelpool-wp.pdf.

[16] 潘爱民. Windows 内核原理与实现[M]. 北京: 电子工业出版社, 2013.
PAN Ai-min. Understanding the Windows kernel[M]. Beijing: Publishing House of Electronics Industry, 2013.

[17] JIA X, ZHU M. Research on significance of VCPU scheduling for SR-IOV on NUMA platform[C]//2014 International Conference on Cloud Computing and Internet of Things (CCIOT). [S.l.]: IEEE, 2015: 57-60.

[18] WANG R B, LU K, LU X C. Aware conflict detection of non-uniform memory access system and prevention for transactional memory[J]. Journal of Central South University, 2012, 19(8): 2266-2271.

[19] Microsoft. MSDN library for visual studio 2017[EB/OL]. [2016-05-09]. https://msdn.microsoft.com/en-us/library/984x0h58.aspx.

编 辑 蒋 晓