Journal of Computer Applications
计算机应用,2015,35(6):1555 - 1559
ISSN 1001-9081
CODEN JYIIDU
2015-06-10
http://www.joca.cn

文章编号:1001-9081(2015)06-1555-05
doi:10.11772/j.issn.1001-9081.2015.06.1555

基于轻量操作系统的虚拟机内省与内存安全监测

马乐乐[1,2*],岳晓萌[1,2],王玉庆[1,2],杨秋松[1,2]
(1.中国科学院 软件研究所,北京 100190; 2.中国科学院 通用芯片与基础软件研究中心,上海 201210)
(*通信作者电子邮箱 lelema.cn@gmail.com)

摘 要:针对在传统特权虚拟机中利用虚拟机内省实时监测其他虚拟机内存安全的方法不利于安全模块与系统其他部分的隔离,且会拖慢虚拟平台的整体性能的问题,提出基于轻量操作系统实现虚拟机内省的安全架构,并提出基于内存完整性度量的内存安全监测方案。通过在轻量客户机中实现内存实时检测与度量,减小了安全模块的可攻击面,降低了对虚拟平台整体性能的影响。通过无干涉的内存度量和自定义的虚拟平台授权策略增强了安全模块的隔离性。基于 Xen 中的小型操作系统 Mini-OS 实现了虚拟机内省与内存检测系统原型,评估表明该方案比在特权虚拟机中实现的同等功能减少了 92% 以上的性能损耗,有效提高了虚拟机内省与实时度量的效率。

关键词:虚拟机内省;Xen Mini-OS;内存监控;完整性度量;入侵检测
中图分类号:TP309 **文献标志码**:A

Virtual machine introspection and memory security monitoring based on light-weight operating system

MA Lele[1,2*], YUE Xiaomeng[1,2], WANG Yuqing[1,2], YANG Qiusong[1,2]
(1. *Institute of Software, Chinese Academy of Sciences, Beijing* 100190, *China*;
2. *CPU and Fundamental Software Research Center, Chinese Academy of Sciences, Shanghai* 201210, *China*)

Abstract: The method of utilizing Virtual Machine Introspection (VMI) in a traditional privileged Virtual Machine (VM) to monitor the memory security of other VMs may weaken the isolation between the security module and other parts of the system, and slows down the total performance of the virtualization platform. In order to mitigate these disadvantages, a security architecture based on implementing VMI in a light-weight operating system was proposed, along with a security checking scheme based on memory integrity measurements. By monitoring and checking other VMs' runtime memory in a light-weight VM, the attack surface as well as the performance overhead was reduced. By non-intrusive checking and personalized authentication policy of the virtualization platform, the isolation of the security module was strengthened. A prototype system of VMI and memory detection was implemented based on Mini-OS of Xen. Compared with achieving the same function in privileged VM, the proposed scheme can reduce performance loss by more than 92% . It is proved that the proposed scheme can significantly improve the performance of VMI and realtime checking.

Key words: Virtual Machine Introspection (VMI); Xen Mini Operating System (Xen Mini-OS); memory monitoring; integrity checking; intrusion detection

0 引言

虚拟化技术是云平台能同时为多个用户提供服务的核心技术。在虚拟化技术下,同一物理设备可以被虚拟成多台虚拟设备,由多个用户分别使用且互不影响,从而降低计算成本。然而,虚拟化平台下的安全问题面临诸多挑战[1-2]:首先,传统计算环境下的漏洞攻击在虚拟机(Virtual Machine, VM)环境下同样有效,如 Web 漏洞 CVE-2012-0392、安全外壳协议(Secure SHell, SSH)漏洞 CVE-2010-4478、域名系统(Domain Name System,DNS)漏洞 CVE-2008-0122 等;其次,虚拟化技术使得攻击者有了新的攻击方式,如 VMware Tools 中的函数库访问漏洞 CVE-2010-1141 等。

在虚拟环境下,传统的运行在操作系统内部的安全监测程序已不能满足虚拟化环境的安全需求。而在特定的虚拟机中利用虚拟机内省技术实时监测其他虚拟机的内存[3-5],成为虚拟环境下安全防护的研究热点。例如 LibVMI[4] 和 RTKDSM[5] 可以在 Xen 的 Domain0 中对 Xen 平台上其他 VM 执行监控。HyperCoffer[6] 利用了现有的处理器硬件虚拟化扩展技术,在 Hypervisor 不可信的情况下对虚拟机进行加密与完整性保护。但这些研究均是在虚拟平台的特权虚拟机中实现虚拟机内省,该方法仍存在如下问题:1)特权虚拟机上运行 Linux 等商业化操作系统,其功能复杂,可信计算基(Trusted Computing Base,TCB)较大,不利于安全模块与其他模块的隔离,增加了安全模块的可攻击面;2)同时,特权虚拟机运行时消耗资源较多,在其中执行内省会拖慢虚拟平台的整体性能。取证虚拟机(Forensics Virtual Machine,FVM)[7-8]

收稿日期:2014-12-19;**修回日期**:2015-03-20。
基金项目:中国科学院知识创新工程重要方向性项目(KGCX2-YW-12);核高基国家重大项目(2014ZX01029101-002)。
作者简介:马乐乐(1989 -),男,山东莱芜人,硕士研究生,主要研究方向:系统安全、虚拟现实; 岳晓萌(1989 -),男,山东青州人,助理工程师,硕士,主要研究方向:系统安全、可信计算; 王玉庆(1987 -),男,河南南阳人,助理工程师,硕士,主要研究方向:信息安全、操作系统安全; 杨秋松(1977 -),男,河北泊头人,研究员,博士,CCF 会员,主要研究方向:系统安全、形式化方法。

利用 Xen 中的轻量虚拟机 Mini-OS 建立了多个专用的虚拟机监控器,每一个监控器负责搜索特定的恶意行为特征。但其多个 FVM 带来的性能损耗较大,其实验原型中也只有对目标系统进程字节级别的访问实验,不能说明大范围内存访问的性能。

为克服上述弊端,本文提出在一个安全隔离的轻量虚拟机中监测其他普通虚拟机内存的安全架构,并用于实时度量系统内存的完整性。主要贡献有:1)提出基于轻量操作系统的安全隔离与监控架构,允许在一个隔离的轻量虚拟机中监测其他客户虚拟机的内存。2)提出基于内存完整性度量的安全监测方案,并将虚拟机内省技术应用到 Xen 平台的 Mini-OS 操作系统中,实现了系统原型。3)评估原型系统性能,结果表明该机制在多达 500 页内存的连续访问中可减少 92% 以上的性能损耗,提高了虚拟机内省与实时度量的效率。

1 威胁模型

1.1 虚拟机内省

虚拟机内省技术允许一个虚拟机监控其他虚拟机的运行状态。一个拥有特殊权限的虚拟机可以通过虚拟机内省访问另一个客户虚拟机的内存,进而从中抽取出运行信息,实时监控该虚拟机状态。虚拟机内省无需改动目标虚拟机系统,可以较好地隐藏自己的监控行为。

开源的虚拟机内省工具有 XenAccess、LibVMI、Volatility 等。LibVMI[4] 源于 XenAccess[3],是在 Xen 中实现虚拟机内省的库函数包,为操作系统实现虚拟机内省提供函数库支持。Volatility 是计算机取证应用较广的开源工具,能为用户提供更为丰富易用的功能。本文借鉴 LibVMI 的工作,从虚拟平台底层开始,在微小操作系统中实现虚拟机内省技术。

本文实现虚拟机内省的轻量虚拟机称为安全代理;其他的虚拟机作为监控对象,称为目标虚拟机。安全代理可以通过虚拟机内省技术,实时读取目标虚拟机的内存,进而分析目标虚拟机的运行状态。

1.2 威胁模型

保障系统的完整性是系统防护的基本目标。完整性要求信息与程序仅以特定的、已获得授权的范围内变化[9]。成功的攻击行为会引起系统部分运行状态的异常变化。若系统任一部分的完整性遭到破坏,则整个系统的完整性便被破坏。本文主要关注客户机系统受到的攻击,如:从用户接口发起的操作系统攻击;在一个客户机内的恶意代码注入攻击;篡改攻击;通过代码注入提升权限实现多个恶意客户机之间的联合攻击等。对系统完整性的实时度量,可检测客户机系统中正在发生的攻击。

威胁模型基于虚拟化平台中的如下假设:1)假设虚拟机监控器是安全的,攻击者不能利用虚拟机监控器中的漏洞控制整个虚拟机平台。2)假设启动过程是安全的,包括监控器的启动、虚拟机的启动以及虚拟机内关键应用的启动都是可信的。该状态作为后续执行实时校验的标准。3)假设客户虚拟机是不安全的,被攻击者可以利用客户操作系统或者应用程序中的漏洞攻陷客户虚拟机系统,破坏客户虚拟机系统的完整性。

安全代理运行在被监控的操作系统之外,如果被监控的操作系统被恶意程序控制,安全代理不会受到侵害,故仍然可信。然而,当安全代理所在的虚拟机被恶意程序侵染之后,通过内省技术得到的信息则是不可靠的,对此本文不作讨论。

2 系统结构

恶意程序会通过修改它所在的环境的状态来隐藏自己。这些恶意程序在设计之初就致力于绕过系统内部的防护程序。然而,如果从系统的外部实时监控系统内部的状态变化,则这些恶意软件将很难隐藏自己的行为。本文在一个微小操作系统中实现虚拟机内省技术,为运行在其他虚拟机上的商业化操作系统提供完整性度量服务,以减少性能损耗,缩小监控系统的可攻击面。

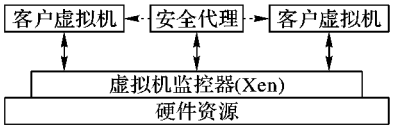

图 1 安全代理监控客户机

安全代理实施监控的整体架构如图 1 所示。安全代理中存储着其他虚拟机安全可信状态的度量值。在其他虚拟机运行过程中,安全代理可以借助虚拟机内省技术,获取内存的状态与变化,实时度量目标内存区域的完整性。比如,度量目标客户机系统关键代码是否被修改等。安全代理所监控的目标内存区域,以及监控的策略与方法,都配置在安全代理中。

2.1 设计原则

在系统设计过程中设定如下原则:轻量虚拟机、无干涉度量和虚拟机隔离与最小权限原则。下面讨论这 3 个原则以及相应的解决方案。

2.1.1 轻量虚拟机

在一个专用的虚拟机中完成单个任务,如果使用现有的商业化操作系统,如 Linux 或者 Windows 等,则虚拟机启动和运行需要消耗大量系统资源,而完整性校验的工作仅占损耗的一部分,所以会造成资源的浪费,降低运行效率。而微小操作系统可用较少资源完成相同的任务。此外,微小系统的 TCB 小,给整个系统带来的 TCB 增长较小。在其他安全条件都相同的情况下,TCB 规模越小,系统的可靠性越高[10]。

本文采取 Xen 源码树中的微小操作系统 Mini-OS[11] 作为安全代理的系统环境。Mini-OS 内核规模小,仅需 1 MB 内存就可以运行,且内核简单,没有内核空间与用户空间的分离,也没有多线程的支持。由此,本文在 Mini-OS 中实现虚拟机内省技术。相对于实现在商业化的操作系统中,这样的实现内部结构简单,且具备精简而够用的对外接口,减小安全代理的可攻击面,同时节约系统资源。

2.1.2 无干涉度量

无干涉是指不修改目标操作系统的代码与运行状态。威胁模型假设所有被监控的客户虚拟机都是不安全的,因此安全代理不能依赖于对客户虚拟机的修改,且检测过程要对目标虚拟机以及云平台上的其他虚拟机具有隐秘性。借助虚拟机内省技术,度量过程不需要对目标虚拟机的代码作任何增减,故可实现无干涉度量,隐藏安全代理的监控行为,增强安全代理与被监控对象的隔离。

尽管虚拟机内省不需要改动目标虚拟机系统的任何代码与数据,但一些特定的恶意程序仍然可以检测到客户机是否正在运行在一个虚拟机环境中[12]。恶意程序还可能利用共享虚拟 CPU(virtual CPU,vCPU)的缓存的隐蔽信道检测到是

否有虚拟机在监控它的恶意行为[13]。

2.1.3　最小权限限制与隔离

隔离的运行环境可提高安全代理的可信性。但在普通虚拟机架构中,虚拟机之间可相互通信、共享内存。为保证安全代理与外界的隔离,需要切断安全代理所在的虚拟机与其他的虚拟机之间不必要的内存共享与通信方式。此外,安全代理作为虚拟机监控者,具有访问其他虚拟机内存的权限。为防止安全代理影响其他虚拟机的正常运行,限定安全代理具有最小内存访问权是另一个必要措施。

由此,安全代理在设计过程中需遵循以下限制:

1)安全代理仅与虚拟机监控器直接通信。安全代理对外暴露出的接口要尽可能少。本文假设其他虚拟机是不安全的,因此为减小安全代理的可攻击面,禁止与其他虚拟机的通信接口是有效的办法。安全代理仅保留与虚拟机监控器的通信接口,实现对其他虚拟机的内存读取与实时度量。

2)禁止安全代理与其他 VM 之间的内存共享机制。虚拟机监控器提供内存共享机制,允许 VM 之间访问对方的内存。例如,Xen 中的授权表[14]等。由于安全代理可以借助虚拟机监控器的内存管理接口获取其他 VM 的内存访问权,所以对于其他内存共享方式,需加以禁止。此外,需要禁止被监控 VM 对安全代理的内存访问。比如,在 Xen 中,禁止被监控 VM 通过监控器内存管理接口、授权表等方式访问安全代理的内存页。

3)安全代理对其他 VM 的内存的访问权限为只读权限。如果允许安全代理改变其他 VM 的内存,则目标虚拟机的完整性或安全性很可能会被安全代理破坏。比如,如果安全代理被攻陷,则攻击者就可以利用该代理控制其他虚拟机。

在 Xen 中采用强制访问控制工具 Xen Security Modules (XSM)[15]可实现虚拟机环境的强制隔离及特殊的内存授权。XSM 可以控制虚拟机与虚拟机之间、虚拟机与监控器之间,以及相关资源如内存和设备之间的交互权限。使用 XSM 既可以禁止不必要的虚拟机通信与内存共享,也可以在无需修改目标虚拟机的前提下授予安全代理虚拟机对其他虚拟机的内存只读权限。

2.2　结构设计

如图 2 所示,完整性校验系统结构主要包括 3 个模块:主模块、可信状态库和虚拟机内省模块。

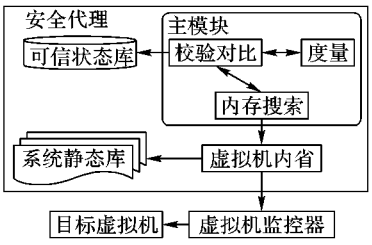

图 2　完整性校验系统结构

2.2.1　主模块

主模块负责对整个安全代理的管理,借助其他功能模块提供给主模块的接口,实现安全代理的基本功能,主要包括内存搜索、度量和校验对比。

内存搜索部分可以根据内省模块获取的内存页,对监控目标进行语义抽取和搜索定位。度量部分计算监控目标的度量值,允许实现多种度量算法。本文以 SHA1 算法为例实现原型系统。主模块的校验对比部分可以访问安全状态数据库中存储的可信度量值,与实时度量值进行比较分析。

2.2.2　可信状态数据库

可信状态数据库用于存储虚拟机可信度量值,可以包含所有有助于校验内存运行状态可信性的度量值。比如:用于校验静态代码的程序哈希值,用于校验程序的控制流完整性的程序流控制图[16]等。

本文假设度量对象的启动状态可信。可信度量值在目标对象的启动过程中初始化到数据库中。在目标对象的实时度量过程中,这些可信状态被用来校验目标对象的完整性。

2.2.3　虚拟机内省模块

安全代理实时度量内存中的运行对象,需要能够访问目标虚拟机的内存页,并从中抽取出内存状态信息。虚拟机内省模块借助开源工具 LibVMI 获得目标虚拟机的内存信息。目标虚拟机内存页的实时获取由该模块负责。该模块是安全代理与虚拟机监控器直接通信的唯一接口。该模块通过向虚拟机监控器发出请求,获取目标虚拟机特定的内存页内容以及基本语义信息。比如,通过内省模块可以读取目标虚拟机的虚拟 CPU 寄存器状态,获取内核页表、用户进程页表和特定内存页等。

3　系统原型实现

基于以上设计,本文在 Xen 的微小操作系统 Mini-OS 中实现了原型系统 TinyVMI,对其他客户虚拟机的完整性执行无干涉的实时内存度量。下面从库函数、XSM 授权、实时校验步骤 3 个方面介绍实现中的一些关键技术细节。

3.1　库函数

安全代理的编译需要 C 库以及 Glib 库的支持,而在 Xen 原生的 Mini-OS 中并没有这些库函数。最新的 Xen 源码树中已经提供对 C 库的交叉编译环境,该交叉编译环境使用 Newlib 作为 C 库。Newlib 是在嵌入式系统开发中经常用到的 C 函数库,其中实现了标准 C 库最小集合以及一些基本的数学库函数,如输入输出函数、字符串操作、时间戳函数等。

Glib 中定义的一些常用数据结构及其算法需要在安全代理中使用。但是由于 Glib 与 Xen 的 Mini-OS 环境不能兼容,且多数复杂库函数在安全代理中没有用到,因此将依赖于 Glib 的常用函数完成自己的实现。用替代后的 Glib 函数的数据结构与函数示例如下。

用于记录最近使用的物理页的链表 tiny_list 以及对应的维护函数定义如下:

```
typedef int64_t * mem_key_t;
struct tiny_list_node{
        struct tiny_list_node * next;
        mem_key_t data;};
struct tiny_list{
        int size;
        tiny_list_node_t head;
        tiny_list_node_t tail;};
status_t tiny_list_foreach(tiny_list_t list, void * remove_entry,
  mem_cache_t mem_c);
status_t tiny_list_free(tiny_list_t list);
tiny_list_node_t tiny_list_find_custom
  (tiny_list_t lru, mem_key_t paddr);
tiny_list_t tiny_list_remove_link(tiny_list_t lru,
  tiny_list_ node_t last);
```

```
tiny_list_t tiny_list_remove(tiny_list_t list, mem_key_t key);
tiny_list_t tiny_list_prepend(tiny_list_t list, mem_key_t key);
```

下面是利用上述库函数的例子——对虚拟机内省物理页的缓存操作函数：

```
status_t mem_cache_remove(mem_cache_t mem_cache, mem_key_t
  key);
mem_cache_entry_t mem_cache_lookup
   (mem_cache_t mem_cache, mem_key_t key);
int mem_cache_size(mem_cache_t mem_cache);
status_t mem_cache_insert(mem_cache_t mem_cache,
  mem_key_t key, mem_cache_entry_t entry);
void memory_cache_init(vmi_instance_t vmi,
  unsigned long age_limit);
void * memory_cache_insert(vmi_instance_t vmi, addr_t paddr);
void memory_cache_destroy(vmi_instance_t vmi);
```

3.2 XSM 授权

在 Xen 中，除了 Domain0 之外，客户虚拟机无法直接访问其他客户虚拟机的内存。安全代理需要在隔离的微小虚拟机中获取其他虚拟机的内存状态。因此，借助 Xen 中的 XSM 机制可以赋予小型虚拟机必需的内存访问权限。

通过 XSM 的授权限制主要有两方面：1）授予安全代理足够的内存访问权限，使其拥有对其他虚拟机内存的只读权限；2）除此之外，禁止安全代理所在的轻量级虚拟机与其他虚拟机之间其他形式的通信与内存共享机制，保证该轻量虚拟机与外界的通信接口最少。

下面是利用 XSM 为安全代理定义的权限策略示例，安全代理被标记的策略类型为 DomT_t 类型。下面的策略组合允许安全代理以调用 Hypercall 的方式与 Xen Hypervisor 通信，并获取对其他 HVM 类型客户虚拟机内存的只读权限。其 FLASK 策略定义如下：

```
allow domT_t xen_t: xen {kexec readapic writeapic mtrr_read mtrr_
  add mtrr_del scheduler physinfo heap quirk readconsole writeconsole
  settime getcpuinfo microcode cpupool_op sched_op pm_op};
allow domT_t domHVM_t: mmu {memorymap map_read pageinfo
  pagelist};
allow domT_t domHVM_t: domain {setvcpucontext pause unpause
  resume max_vcpus setvcpuaffinity getvcpuaffinity getdomaininfo
  getvcpuinfo getvcpucontext setdomainmaxmem setdomainhandle
  setdebugging hypercall settime set_target setaddrsize getaddrsize
  trigger getextvcpucontext setextvcpucontext getvcpuextstate
  setvcpuextstate getpodtarget setpodtarget set_misc_info set_virq_
  handler};
allow domT_t domHVM_t: hvm {sethvmc gethvmc mem_sharing};
```

3.3 实时校验过程

安全代理执行实时完整性校验的过程如图 3 所示，关键环节有启动记录可信度量、安全代理搜索度量目标和度量与验证三部分。

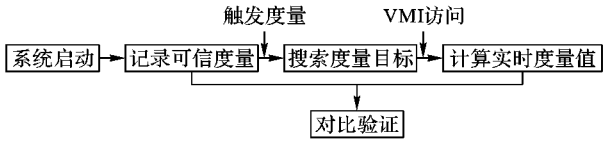

图 3 实时检验过程

3.3.1 启动时记录可信度量

可信状态数据库的度量值是在对象的启动过程中初始化到数据库中。比如对于键盘驱动程序，安全代理会在目标虚拟机的中断向量表初始化完成后，根据目标虚拟机的中断向量表搜索键盘驱动程序在目标虚拟机内存中的位置，获取相关的内存页，然后计算键盘驱动程序中的静态代码段与数据段，进而计算度量值并保存到可信状态数据库中。

3.3.2 安全代理搜索度量目标

实时完整性度量由安全代理主动触发执行，首先根据度量目标的虚拟机名称、目标虚拟地址确定目标物理地址，然后将物理地址的内存页映射到安全代理的地址空间中。这样，在安全代理中就可以访问目标虚拟机中度量对象的物理页面。比如，对于键盘驱动程序，首先根据目标虚拟机名称以及中断向量表偏移，经一系列地址转换与搜索，最终确定中断处理程序的物理页面。之后将内存页面映射到安全代理的内核空间中，然后根据程序结构，获取键盘服务程序的代码段、数据段等内容。

3.3.3 度量与验证

获取度量目标的内存信息之后，安全代理可以根据度量目标的定义计算目标的度量值。度量值的计算采用 SHA1 算法。在键盘驱动的校验过程中，被度量区域可信状态的初始化和运行时的度量使用相同度量算法。如果同一区域运行时度量的哈希值与存储的可信状态的哈希值不相同，说明该区域的状态与启动时的状态不一致，则认为键盘程序完整性遭到破坏。

4 性能分析

为评估 TinyVMI 的内存访问对整个系统性能的影响，本文在基于 Xen Domain0 的 LibVMI 和基于 Xen Mini-OS 的 TinyVMI 中，分别实现了两组功能相同的实例程序：虚拟地址转换和内核模块遍历程序，并分别对比了两组程序在 LibVMI 和 TinyVMI 上运行的性能。硬件平台：CPU 为 Intel Pentium Dual-Core，主频 2.80 GHz，内存 2 GB。软件平台：虚拟机监控器 Xen-4.4.0，LibVMI-0.10.1；Domain0 为 Ubuntu 12.04（64 bit），分配内存 900 MB；目标虚拟机为硬件虚拟机（Hardware Virtual Machine，HVM）模式的 Linux Mint 15（32 bit），内存 700 MB；基于 Mini-OS 的原型系统分配内存 32 MB。

下面分别介绍在原型系统上实现的两组实例程序及其性能表现。

第一个实例是虚拟地址转换。程序以虚拟机名称和虚拟机地址作为输入，查找目标系统的内核页表，并通过获取的各级内核页表计算出最终的目标物理地址。整个过程需要多级页表的访问，因此会发生多次循环调用地址转换接口。

为对比基于 Domain0 的 LibVMI 和基于 Mini-OS 的 TinyVMI 两者的运行性能，实验统计了 TinyVMI 和 LibVMI 在同一目标虚拟机的相同度量对象上完成 30～500 个虚拟地址页首地址的转换与映射时间，统计结果如图 4 所示。随着访问页数的增加，TinyVMI 的增长速度远远小于 LibVMI 的增长。此外，对于 500 页内存访问的平均时间进行统计，使用 LibVMI 的平均时间为 1 139 μs，而使用 TinyVMI 仅为 43.95 μs。

上述结果表明，在 Mini-OS 中实现的地址转换操作比在 Domain0 中实现的相同功能减少了 96% 的时间消耗。由此可见，对于同一个目标虚拟机的相同页地址的转换与映射，在 TinyVMI

中实现的性能损耗大大低于在 LibVMI 中的性能损耗。

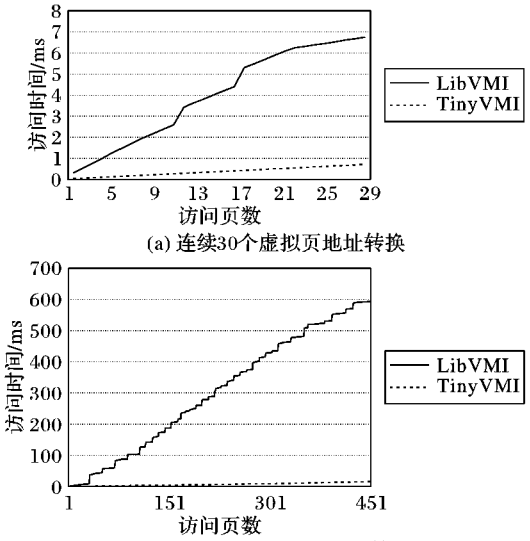

图 4　内存页访问时间统计

另一个实例是内核模块的访问程序。本实例程序以虚拟机名称作为输入,可以搜索目标虚拟机的内核空间并获取已被加载到内核的内核模块名称,测试 TinyVMI 和 LibVMI 在遍历 Linux 客户机内核中所有模块名称所需要的时间。实验结果显示,使用 LibVMI 遍历一个客户机所有内核模块名称的平均时间为 30.0 ms,而 TinyVMI 仅为 2.2 ms。结果表明 TinyVMI 可以节省 LibVMI 所需时间的 92%,因此,TinyVMI 原型系统在内存分析上同样具有更好的性能。

综上所述,在微小操作系统中实现虚拟机内省并实时监测目标虚拟机的内存,相对于在 Domain0 中实现该技术,可以节省 92% 以上的内存访问时间,有效减小了内存监测对整体性能的影响,提高系统的运行效率。

5　结语

在商业化操作系统实现虚拟机内省来监控其他系统内存的方法对平台的性能影响很大,而且显著增大了系统可信代码基。本文提出了一种在微小操作系统中实现虚拟机内省,并用来监控商业化目标系统的系统完整性的体系架构,实现了原型系统,并对原型系统的内存监控行为作了评估,取得了显著的性能提升。

虽然当前工作实现了在微小系统中对大型商业化操作系统的实时内存度量,但该平台的安全功能还有很大的扩展空间,在设计架构上同样有很大的扩展余地。比如,安全代理所监控的目标内存区域,以及监控的策略与方法,都定义并配置在安全代理中。这类工作可以借助安全规则来实现,所以后续需提出具有足够描述能力的规则定义与实施模型,以满足更多的内存监控与安全防护需求,进一步优化平台的功能与效率。

参考文献:

[1] PÉK G, BUTTYÁN L, BENCSÁTH B. A survey of security issues in hardware virtualization [J]. ACM Computing Survey, 2013, 45(3): Article 40.

[2] CHAWLA S, NIGAM A, DOKE P, *et al.* A survey of virtualization on mobiles [C]// ACC 2011: Proceedings of the First International Conference on Advances in Computing and Communications, Communications in Computer and Information Science, LNCS 191. Berlin: Springer, 2011: 430 – 441.

[3] PAYNE B D, de CARBONE M D P, LEE W. Secure and flexible monitoring of virtual machines [C]// ACSAC 2007: Proceedings of the Twenty-Third Annual Computer Security Applications Conference. Piscataway: IEEE, 2007: 385 – 397.

[4] PAYNE B D. Simplifying virtual machine introspection using LibVMI, SAND2012-7818 [R/OL]. [2014 – 12 – 01]. http://prod.sandia.gov/techlib/access-control.cgi/2012/127818.pdf.

[5] HIZVER J, CHIUEH T. Real-time deep virtual machine introspection and its applications [C]// VEE 2014: Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York: ACM, 2014: 3 – 14.

[6] XIA Y, LIU Y, CHEN H. Architecture support for guest-transparent VM protection from untrusted hypervisor and physical attacks [C]// HPCA 2013: Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture. Piscataway: IEEE, 2013: 246 – 257.

[7] SHAW A L, BORDBAR B, SAXON J, *et al.* Forensic virtual machines: dynamic defence in the cloud via introspection [C]// IC2E 2014: Proceedings of the 2014 IEEE International Conference on Cloud Engineering. Piscataway: IEEE, 2014: 303 – 310.

[8] HARRISON K, BORDBAR B, ALI S T T, *et al.* A framework for detecting malware in cloud by identifying symptoms [C]// EDOC 2012: Proceedings of the 2012 IEEE 16th International Enterprise Distributed Object Computing Conference. Piscataway: IEEE, 2012: 164 – 172.

[9] GUTTMAN B, ROBACK E A. An introduction to computer security: the NIST handbook [R]. Gaithersburg: National Institute of Standards and Technology, 1995.

[10] SINGARAVELU L, PU C, HARTIG H, *et al.* Reducing TCB complexity for security-sensitive applications: Three case studies [J]. ACM SIGOPS Operating Systems Review, 2006, 40(4): 161 – 174.

[11] POPURI S. A tour of the Mini-OS kernel [EB/OL]. [2014 – 12 – 01]. http://www.cs.uic.edu/~spopuri/minios.html.

[12] PALEARI R, MARTIGNONI L, ROGLIA G F, *et al.* A fistful of red-pills: how to automatically generate procedures to detect CPU emulators [C]// WOOT 2009: Proceedings of the 3rd USENIX Conference on Offensive Technologies. Berkeley: USENIX Association, 2009: 2 – 2.

[13] RISTENPART T, TROMER E, SHACHAM H, *et al.* Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds [C]// CCS 2009: Proceedings of the 16th ACM Conference on Computer and Communications Security. New York: ACM, 2009: 199 – 212.

[14] CHISNALL D. The definitive guide to the Xen hypervisor [M]. Upper Saddle River: Pearson Education, 2008: 59 – 74.

[15] COKER G. Xen security modules (XSM) [EB/OL]. [2014 – 12 – 01]. http://mail.xen.org/files/summit_3/coker-xsm-summit-090706.pdf.

[16] WANG Z, JIANG X. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity [C]// SP 2010: Proceedings of the 2010 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2010: 380 – 395.