计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20170452
54(10):2310-2320,2017

基于 VMFUNC 的虚拟机自省触发机制

刘维杰　王丽娜　谈　诚　徐　来
(空天信息安全与可信计算教育部重点实验室(武汉大学)　武汉　430072)
(武汉大学计算机学院　武汉　430072)
(软件工程国家重点实验室(武汉大学)　武汉　430072)
(liuweijie@whu.edu.cn)

A Virtual Machine Introspection Triggering Mechanism Based on VMFUNC

Liu Weijie, Wang Lina, Tan Cheng, and Xu Lai
(*Key Laboratory of Aerospace Information Security and Trusted Computing* (*Wuhan University*), *Ministry of Education*, *Wuhan* 430072)
(*School of Computer Science*, *Wuhan University*, *Wuhan* 430072)
(*State Key Laboratory of Software Engineering* (*Wuhan University*), *Wuhan* 430072)

Abstract　Virtualization technology as the basis of cloud computing has been widely used, while security issues of virtual machine have been attracted more and more attention. The virtual machine introspection, as an "out-of-the-box" method leveraged to monitoring virtual machine, provides a new perspective for solving the security problems. Aiming at this situation, a triggering mechanism based on VMFUNC is proposed. Taking the advantages of the CPU hardware features VM-Function and RDTSC emulation, the mechanism minimizes the overhead of VM exits. Based on the extended page table view switching through the VMFUNC, our mechanism avoids the system pause caused by VMI programs. By means of overloading VMFUNC and Xentrace, our method can trigger VMI programs actively, thus overcoming the VMI program resident consumption. In this paper, a VMI-as-a-service system is implemented and verified by experiments. The results show that the performance cost is no more than 2%, which makes VMI widely being used possible in practical cloud environment.

Key words　cloud computing security; virtual machine introspection; VMFUNC; extended page table pointer (EPTP) switching; VMI-as-a-service

摘　要　虚拟化技术作为云计算的基础得到了广泛应用,但随之而来的虚拟机安全问题日趋严重.虚拟机自省作为一种从外部监控虚拟机内部运行状态的方法,为解决虚拟机安全问题提供了新视角,但同时也引入了巨大开销,阻碍了实际应用.提出了一种基于 VMFUNC 的虚拟机自省(virtual machine introspection, VMI)触发机制.该机制借助 CPU 硬件特性 VM-Function 以及 RDTSC 指令模拟,将调用时产生 VM Exit 开销降至最低;利用 VMFUNC 的功能为目标虚拟机切换备用扩展页表,避免 VMI 程序运行时对虚拟机执行的中断;通过重载 VMFUNC 指令和 Xentrace 的功能实现高效的触发与信息

收稿日期:2017-06-02;**修回日期**:2017-07-25
基金项目:国家自然科学基金项目(61373169,61672394);国家"八六三"高技术研究发展计划基金项目(2015AA016004);国家科技支撑计划基金项目(2014BAH41B00);NSFC-通用技术基础研究联合基金项目(U1536204)
This work was supported by the National Natural Science Foundation of China (61373169, 61672394), the National High Technology Research and Development Program of China (863 Program) (2015AA016004), the National Key Technology Research and Development Program of China (2014BAH41B00), and the NSFC-General Technology Basic Research Joint Funds (U1536204).
通信作者:王丽娜(lnwang.whu@gmail.com)

传递机制,主动触发 VMI 程序运行,克服了 VMI 程序常驻带来的大量资源消耗.实现了虚拟机自省即服务系统,并进行了实验验证.结果表明:本系统带来额外性能开销不超过 2%,使 VMI 在实际云环境中的广泛应用成为了可能.

关键词 云计算安全;虚拟机自省;VMFUNC;扩展页表指针切换;虚拟机自省即服务

中图法分类号 TP309

在云计算已经得到普及和大规模应用的今天,虚拟化技术作为其基础得到了广泛应用,与此同时,虚拟机的安全问题也得到了越来越多的关注.其中,虚拟机自省(virtual machine introspection, VMI)[1]作为一种在虚拟机外部监控虚拟机内部运行状态的方法,充分利用了虚拟机管理器(virtual machine monitor, VMM 或 Hypervisor)的隔离特性以及介于硬件和虚拟化操作系统之间的新的抽象层次,为解决虚拟机安全问题提供了新视角.拥有特权的虚拟机可以通过作为安全虚拟机,借助 VMM 监控虚拟机的行为和软硬件资源使用情况(例如执行指令等)来感知虚拟机的内部状态,实现对于虚拟机透明地安全监控,以待进一步的分析与检测.

然而 VMI 技术在大规模虚拟机安全监控应用场景中仍存在疏漏.现有的 VMI 存在资源消耗高、运行时间长的困境,无法很好地适应云环境.传统的虚拟机自省方法需要由主机接管关键页面进行处理,从而实现 VMI.文献[2]指出 VMI 技术的额外开销一般在正常开销的 9.3~500 倍之间,有时甚至需要 6 s 才能遍历操作系统进程链.为了保证客户机内核(guest-OS)所维护的进程信息一致,虚拟机(virtual machine, VM)在某些情况下必须暂停.这样攻击者可能会利用这些时间延时知识来误导 VMI 的分析结果,或者延迟启动攻击以避免被检测[3].因此,实现更高效的 VMI 是保证虚拟机安全的必要前提,在云环境中不可或缺.

随着虚拟化的快速演进,软件堆栈的垂直和水平复杂度进一步增加.复杂的软件栈带来了强大的功能,同时也带来了挑战.一方面,程序员有更多的自由来利用软件层实现各种功能,包括安全性、去耦和改进管理功能;另一方面,软件层面和保护域的扩散也导致了更为复杂的跨特权级控制转换.这种不必要的开销和复杂性完全是由于灵活的跨特权级调用的缺乏导致的[4].通常硬件机制仅支持从用户级程序到操作系统的调用(Syscall)以及从虚拟机到 Hypervisor 的调用(即超级调用,Hypercall)[5].若虚拟机要将某些信息传送到 Hypervisor,必须调用超级调用,或者使用 VMREAD 和 VMWRITE 等特殊指令读写 VMCS 字段[6].然而,这些特殊指令将会引入频繁的 VM Exits,产生相当大的性能开销.想要在虚拟化环境中运用 VMI 技术保证虚拟机的安全,必须克服 VMI 具有的开销过大的缺陷.本文通过引入先进的 CPU 硬件特性 VMFUNC 解决该问题.

通过审视现有的 Hypervisor 操作系统,催生了新一代 Intel 处理器的 VM-Function 系列指令(VMFUNC).VM-Function 技术允许用户在多个地址空间之间跨多个层级的安全、有效和灵活的跨特权级调用.VMFUNC 功能集成在英特尔的 Haswell 及其后的微体系架构中,使虚拟机能够在不退出 root 模式的情况下使用 Hypervisor 的功能.VMFUNC 被设计为通用接口,以支持由不同索引指定的多个功能.当前的 Intel 处理器仅实现了一个名为 EPTP switching 的功能,即客机虚拟机能够切换其扩展页表指针(extended page table pointer, EPTP),使用一套新的 EPT 页表(extended page table, EPT),且不触发任何 VM Exits.使用此功能的一种典型应用是将客户端内核的运行环境与用户应用程序隔离开来,以防止内核 Rootkit 使用“Return-to-user”攻击和内存扫描[7].

本文针对云环境中云租户个体的虚拟化安全要求,提出一种灵活的基于 VMFUNC[6]触发 VMI 的机制.该机制借助 CPU 硬件特性 VM-Function 以及 RDTSC 指令模拟,将调用时产生 VM Exit 开销降至最低.利用 VMFUNC 的功能为目标虚拟机切换备用扩展页表,避免 VMI 程序运行时对虚拟机执行的中断.通过重载 VMFUNC 指令和 Xentrace 的功能实现高效的触发与信息传递机制,实现了虚拟机自省即服务系统.用户可以根据需要发出 VMI 的同时,既节约了计算资源,又满足特定用户的安全需求.该触发机制有力地解决了云环境中大规模虚拟机安全监控的难题,为云计算中 VMI 的使用模式提供了新思路.

1 相关工作

自 Garfinkel 等人[1]提出虚拟机自省这一概念以来,VMI 被定义为在虚拟机之外分析虚拟机内的软件运行状况的监控手段. 由于虚拟化平台没有相关接口,SIM[8]如同 Garinkel 的实现一样,放弃了隔离性需求,即从目标虚拟机直接拉取高层数据至 Hypervisor. 甚至如 HyperNE[9]为了实现高性能监控而直接重写 Hypervisor,允许虚拟机与监控软件的直接交互. 然而坚持隔离性会带来语义鸿沟的难题,即从虚拟机获取的内存信息是最底层的二进制数据,并无任何可直接读取的数据结构以供分析. Jones 等人[10]于 2006 年提出了 AntFarm,它可以被 VMM 使用从而独立隔离地运行;Ether[11]利用了硬件虚拟化技术,迫使系统调用发生错误会陷入 VMM 的方法来监控目标虚拟机;Virtuoso[12]提供了生成自省程序的方案,但是其可靠性过低. 同时上述方法都只针对特定的目标进程,若将监控范围扩大到整个虚拟机,则其开销会变得非常巨大.

另一方面,虚拟机自省技术被用来构建一系列的安全工具,包括入侵检测系统 Graffiti[13]、内核完整性验证系统[14]、VMM 控制流完整性验证系统[15-16]等. 然而这些工具都有一个共同的特点:被动触发监控. 安全工具的触发需要通过额外的扫描和轮询,这使得安全工具无法保证在事件发生之前介入.

文献[17-18]通过生成外部语义视图(View)来监控目标虚拟机. 然而,在虚拟机外部观测并产生视图耗时较长,而内部耗时较短[19];基于 Xen 的 DRAKVUF 动态恶意软件分析[20]通过将断点指令写入到客户端的内存中来实现 VMI,利用 CPU 的硬件特性,在上下文切换过程中插入钩子,实现了监控程序的自动化调用,并且使用了额外的 EPT 页表以节省开销. DRAKVUF 通过频繁的上下文切换来陷入 Hypervisor,而在 x86 架构的 CPU 上,每次 VM Exit/VM Entry 都需消耗近 10^4 时钟周期(cycles)[7],实际上依旧引入显著的 VM Exit 开销.

为了利用基于软件结构知识的语义信息来解决语义鸿沟问题,并真正实现 VMI 程序的长期可移植性,Payne[21]于 2012 年设计实现了一个基于 XenAccess 库[22]的虚拟机自省库 LibVMI,它大大提升了 VMI 技术的性能,也创造了一个更易用的 VMI 编程环境. LibVMI 提供了应用程序编程接口,用于读取和写入虚拟机内存. 这种基于软件架构知识的独立自省技术的优势是在目标虚拟机之外获取底层状态信息和语义重构,透明性好;其可获取高层语义信息的范围几乎没有限制,信息量大. 缺点在于如果软件结构知识发生了变化,自省代码就极有可能失效. 若要保证自省代码长期有效,VMI 程序需要暂停虚拟机,因此可移植性差且资源消耗较大[19]. VMI 技术发展的主要障碍是性能问题和语义鸿沟问题,而这两者无法做到有效统一[23].

针对上述研究所存在的被动监控、开销较大等问题,本文在现有基于软件架构自省方案的基础上,结合现有的 CPU 新特性,提出能够主动触发并且实时内存读取的 VMI 机制,既能弥补语义鸿沟,又能保持极少的额外性能开销.

2 基于 VMFUNC 的 VMI 触发机制

本文借助先进的 CPU 硬件特性 VM-Function,结合 RDTSC 指令模拟,避免了 Hypercall 的频繁使用,将调用时产生 VM Exit 开销降至最低;利用 VMFUNC 的 0 号功能 EPTP switching 为目标虚拟机切换备用扩展页表,维持了内存的一致性,避免 VMI 程序运行时会中断虚拟机中程序执行的现象,同时实现了监控的透明化;使用按需触发的服务理念,通过重载 VMFUNC 指令和 Xentrace 的功能,Hypervisor 向特权域注入虚拟中断,VMI 程序的按需启动实现高效的触发与信息传递机制,最大程度地避免 VMI 程序常驻特权域(*Domain*-0, *Dom*0)而导致的资源消耗. 其总体设计如图 1 所示.

为了能够向 Hypervisor 传递一定量的信息(Guest-VM 的自省请求),本文设计使用 VMFUNC 指令所切换的 EPTP 索引作为传递信息的载体来通知 Hypervisor,进而请求自省程序的执行. 具体来说,当某个客户虚拟机(Guest-VM)用户空间中执行 VMFUNC 指令后,扩展页表指针值发生变化. 这样一来,Hypervisor 能够在 RDTSC 模拟指令代码中感知 EPTP 索引的变化,进而判断是否开启针对该 Guest-VM 的 VMI 程序.

在执行 VMI 程序时,该 Guest-VM 的原有页表可以被用作 VMI 程序的监控对象. 此时虚拟机上的应用程序可在 EPTP switching 之后的新 EPT 页表上可无缝地继续执行. 页表的切换仅仅只需 10^{-7} 秒级的时间开销[9],此举可极大程度地降低 VMI 程序对虚拟机运行时的影响.

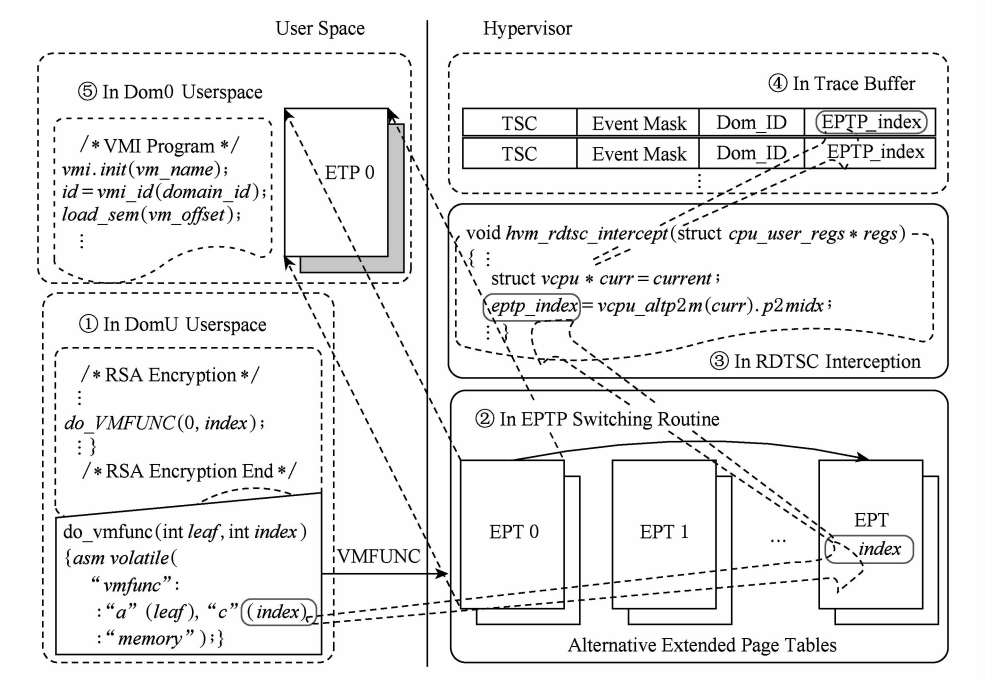


Fig. 1 System design of VMI Triggering Mechanism based on VMFUNC

图 1 基于 VMFUNC 的 VMI 触发机制设计

2.1 VMFUNC 指令利用

在使用 VMFUNC 的 0 号功能——页表指针切换(EPTP switching)之前,Hypervisor 需要为 Guest-VM 创建好备用的 EPT 页表.通过创建多个相同的扩展页表(Xen 为虚拟机创建的最大备用页表数为 10)来确保虚拟机内存访问的一致性.并且设置 VM-functions control 字段为 1,开启"EPTP switching"功能.Guest-VM 可以使用类似图 2 上半部分位于 Guest-OS 的代码示例,执行 VMFUNC 指令调用"EPTP switching"功能.

```
mov      0          %eax
mov      $index     %ecx
vmfunc                          In Guest-OS
-------------------------------------------
vmread   0x201AH    %eax        In Hypervisor
vmread   0x2024H    %edx
```

Fig. 2 VMFUNC code example

图 2 VMFUNC 代码示例

截至 2016 年底,扩展页表有 4 个级别①,它们分别是页面映射 4 级表(page map level 4 table, PML4T)、页目录指针表(page directory pointer table, PDPT)、页目录表(page directory table, PDT)和页表(page table, PT).每个 EPTP 指向一个 PML4T 结构.图 3 表示 Xen 架构中的 VMFUNC 寻址和客户/主机物理地址转换(EPT walking)的过程.

当执行用户应用(例如 RSA 加密,如图 1 步骤①所示)中的 VMFUNC 指令时,EPTP 切换例程将被执行,并且新的 EPTP 将被加载到 VMCS 中的 EPTP 字段中.此时 Hypervisor 通过新传入的 EPTP 值,来计算出当前的 EPTP 索引值.需要特别注意的是,我们通过 ECX 寄存器中传入的不同值,来区分用户的正常 VMFUNC 请求(使用页表切换功能)与本文所提出的虚拟机自省请求.我们规定:若 Hypervisor 计算得到的 EPTP 索引值小于或等于 10,则判断为正常的 VMFUNC 请求;反之,若传入 ECX 的值大于 10(不妨设为 $10+x$),则判断为虚拟机按需自省请求,并将 x 作为新的 EPTP 索引值.如图 3 下半部分所示,Hypervisor 代码中的第 1 行和第 2 行分别读取"EPTP"字段和"EPTP 列表地址(EPTP_LIST_ADDR)"字段."0x201AH"和"0x2024H"是每个 VMCS 中的"EPTP"字段和"EPTP 列表地址"字段的地址.

当采用另一个 CPU 高级硬件特性 # VE(virtual exception)[6]时,本文的方案会更为有效.通过使用 VMFUNC 和 # VE,Xen 虚拟机管理程序不必涉及处理 Guest-VM 的内部策略,从而降低 Hypervisor 的复杂性,并且可以很好地与 VM 迁移配合使用.

① Intel 于 2016 年 12 月发布了新式的 5 级页表模式.

同时,使用 VMFUNC 和 #VE 可以为 Guest 虚拟机分配更多的 CPU 周期,避免 Guest 虚拟机花费过多的时间来处理由 EPT violation 导致的 VM Exit. 因此启用 VMFUNC 和 #VE 的虚拟机将具有更好的性能. 然而,该硬件功能只集成在最新的 Intel CPU 中,因此本文仅讨论 VMFUNC 指令的利用方法.

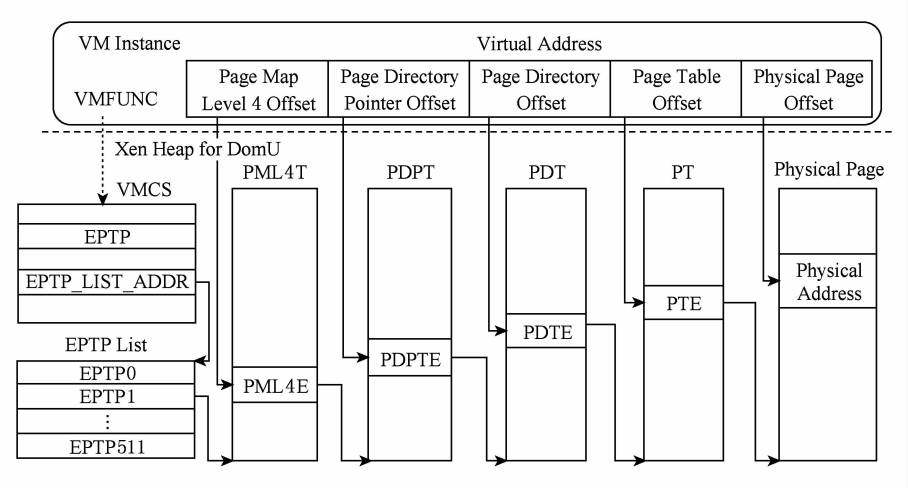


Fig. 3 EPT walking

图 3 EPT walking

2.2 RDTSC 指令模拟

为了保证实时性,本文设计了基于 RDTSC 模拟的 VMFUNC 感知机制. 该机制利用 RDTSC 模拟开销极少的特点[24],同时充分考虑到 RDTSC 请求发生的频繁性,通过将检测 VMFUNC 是否执行的功能插入到 RDTSC 截获与模拟的模块中,实现了 VMFUNC 的透明感知,并且不引入额外开销.

通过对 Xen 源代码的深入研究,发现 Xen 虚拟机管理程序为其上的虚拟机提供了多个时钟源,并且 Xen 上的 Linux 实例默认使用时间戳计数器(time stamp counter. TSC)作为其时钟源. TSC 值可以通过 RDTSC 指令访问,RDTSC 指令从 CPU 的 TSC 寄存器中将 64 b 的值取出,并将其高 32 b 装入 EDX 寄存器,低 32 b 装入 EAX 寄存器.

目前,大多数处理器芯片虚拟化功能均提供 RDTSC 模拟,即提供 Hypervisor 对 RDTSC 和 RDTSCP 指令的条件捕获与模拟. 此外,它们允许软件控制在虚拟机执行期间读取的 TSC 的值,并且在虚拟机控制结构(virtual machine control structure, VMCS)中规定额外的偏移字段[6].

根据现有 CPU 提供的上述特性,本文设置 VMCS 中的 *RDTSC EXITING* 位来截取 Guest-VM 的 RDTSC 指令,并在 Hypervisor 层模拟 RDTSC 指令的执行. 为了方便用户启动和终止 RDTSC 截取,本文修改了 Xen 的源代码,并定制了 2 个超级调用(如表 1 所示). 当 Guest 用户调用第 1 个 Hypercall 时,Hypervisor 将执行"开始截断 TSC 值"函数;当第 2 个 Hypercall——"禁用 RDTSC 拦截"被调用时,VM 的时间将被强制恢复到正常状态. 出于安全考虑,Hypercall 的调用部分已经被封装为系统调用. 通过启停 Hypervisor 对 Guest-VM 的 RDTSC 模拟,进而实现对整个 VMI 触发机制的启停.

Table 1 Capsulate Hypercalls

表 1 所封装超级调用

Hypercall	Description
int *do_enable_intercepting_rdtsc*(void)	Set *RDTSC EXITING* bit as 1, switching on RDTSC emulation.
int *do_disable_intercepting_rdtsc*(void)	Set *RDTSC EXITING* bit as 0, switching off RDTSC emulation.

本文将 VMFUNC 的使用与 RDTSC 截获功能集成在一起. 利用 RDTSC 的高频性(每秒钟执行上万次),保证监控的实时性,其开销与 Guest-VM 正常使用 RDTSC 指令所产生的开销一致.

2.3 VMI 程序触发

通过重载 Xentrace[25],本文重用了其事件记录机制和消息传递机制,设计了一种能够迅速将 Guest-VM 的 VMI 检测需求连同 VMI 程序所需的

参数一并传递至 Dom0 的机制.该机制具备通用性,且不增加过多的源代码,对系统的性能影响极小,最大程度上维护了系统的安全性和可移植性.

Xentrace 是在 Xen 源代码中自带的一个工具,在编译时就会默认生成,在进行测试或调优时或需要进行底层细节分析时可以使用 Xentrace 工具来辅助进行.Xen Hypervisor 在关键位置有许多跟踪点,允许开发人员查看系统内部发生的情况.启用这些跟踪点后,Xen 会将跟踪信息写入 Xen 中的每个 CPU 对应的缓冲区(如图 1 步骤④),然后在 Dom0 中的 Daemon 程序设置并启用跟踪,并定期读取这些缓冲区并将其写入磁盘.Xentrace 生成的数据是二进制的格式,不能直接理解.因此本文为记录 VMFUNC 所传递的参数设计了专门的事件格式与事件解析器来存储解析其数据.

除了 Domain-U 的标识(Dom_ID)与时间戳,Hypervisor 还需要提供 EPTP 索引,用以告知 VMI 程序 Guest-VM 所使用的 EPT 页表基址.缓冲区中的每条记录格式如图 4 所示:

TSC	Event Mask	Dom_ID	EPTP_index

Fig. 4 VMFUNC event record format

图 4 VMFUNC 事件记录格式

运行 VMFUNC 指令后,本文将当前的程序运行环境切换到了另外一套 EPT 页表中,这样一来,原来程序运行的那一套 EPT 页表就能够单独被 Dom0 分析而不中断被监控 VM 应用程序的运行(如图 1 步骤⑤).检测程序和虚拟机应用程序并发执行,因此 VMI 程序的透明性能够得到保障.之后,Hypervisor 将以虚拟中断的形式通知 Dom0 开启自省程序,对有需求的目标 Guest-VM 进行针对性的恶意进程检测.

3 VMI-as-a-Service 系统实现

本文借鉴云服务的理念,在 Xen 虚拟化平台[5]上实现了 VMI-as-a-service 系统①,其具体思想是将 VMI 作为一种服务按需提供给云租户.

3.1 层次结构及模块划分

VMI-as-Service 系统主要分为 3 个模块:VMFUNC 感知模块、参数传递模块和 VMI 启动模块,如图 5 所示.其中,VMFUNC 感知模块存在于 Hypervisor,当目标主机在用户层或者内核层执行 VMFUNC 指令时,若执行成功,则会切换页表.此时,VMFUNC 感知模块在每一个 RDTSC 截获过程中,会判断是否存在 VMFUNC 的执行,即是否有页表基地址被修改.当检测到目标主机 VMFUNC 所传入参数大于 10 时,会通知处于 VMM 的参数传递模块.当参数传递模块被调用时,VMI 启动模块负责开启基于 LibVMI 的自省程序来对目标主机的内存进行分析.

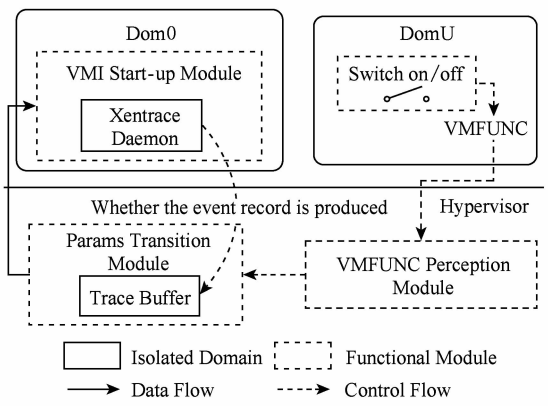


Fig. 5 System architecture and module partition

图 5 系统层次结构及模块划分

3.2 VMFUNC 感知模块

VMFUNC 感知模块负责感知 EPTP switching 的执行和 VMCS 中 EPTP 字段的改变.VMFUNC 感知模块主要实现 3 项功能:

1)读取和保存当前的 EPTP.本文在实现 VMI-as-a-Service 时,为了保证充分的可用性,对原始的 VMFUNC 功能做了保留,并在其上加入了新的功能.当用户想要使用原有 VMFUNC 功能时,其在 ECX 中放置的值为正常值(小于或等于 10).而当 VMFUNC 感知模块发现 ECX 寄存器中的值大于 10,则读取当前的 EPTP,并比对上次保存的 EPTP 值,查看该 EPTP 是否变化,从而感知目标虚拟机 Guest-VM 发出了使用 VMI-as-a-Service 的请求,并进行进一步处理.

2)RDTSC 截获与模拟.Hypervisor 对 VMFUNC 指令感知操作均在 RDTSC 截获之后完成.由于 VMFUNC 感知模块的核心功能即为随时能够对用户发起的 VMFUNC 调用进行感知.于是在本系统中,利用了 RDTSC 指令每秒能够被 VMM 截获上千次的特点,主动地拦截系统获取 CPU 中 TSC 的值.在感知模块运行即将结束时,将

① https://github.com/lpjlwj/VMI-as-a-Service

当前的 TSC 值重新写入寄存器,以保证 TSC 值的正确性与一致性.

3) 计算 EPTP 与 EPT index 的映射,并写入 EPTP 到 VMCS. 当 VMI 感知模块已经感知到用户的 EPTP 字段被修改后,参数传递模块将读入当前的 EPTP 和保存的原 EPTP,接着计算得出当前用户想要切换到的目的 EPT index. 当 VMFUNC 成功执行时,需要将新的 EPTP 存入对应的 VMCS,并且在下一次 VM Entry 时应用到用户虚拟机中.

在执行 VMFUNC 之前,需要为虚拟机创建额外的页表. 在 Xen 的设计中,p2m 表是管理从客户机物理地址到机器地址转换的内存管理层. 在 Xen 中,使用硬件辅助分页(hardware assisted paging, HAP)来具象化 p2m,利用 Intel 扩展页表机制来创建 p2m 表. 而为了创建多个备用 EPT 页表,本文使用 Xen altp2m 机制[26],它允许 Xen 为每个客户端创建不止一个 EPT 页表(EPT View).

当 Guest 虚拟机需要进行 VMI 监控时,Xen 无需重新配置页表的访问权限来限制应用程序的运行[9],只需切换副本 EPT View 即可. 由于每个 vCPU(virtual CPU)都有自己的 VMCS 执行此切换,因此无需暂停任何其他 vCPU 或者对访问进行模拟. Guest-VM 访问其物理内存的方式不变,但实际上它正在使用副本 EPT 页表. Guest-VM 无法感知此过程,从而实现透明性. 其具体步骤设计如下:

① VMFUNC 在用户虚拟机内执行;

② 判断模块是否开启,若开启则继续;若未开启,则执行步骤⑥;

③ 截获 TSC 值并将其保存;

④ 读取当前 EPTP 值,判断当前 EPTP 值与上一次保存的 EPTP 值是否一样,若相同则返回执行步骤①,若不同则调用参数传递模块;

⑤ 将模拟的 TSC 值送入 EAX,EDX 寄存器;

⑥ 结束并返回到虚拟机执行环境.

3.3 参数传递模块

参数传递模块负责将 VMFUNC 感知模块所获得的参数记录在缓存区中,并通知 Dom0 中的 VMI 启动模块读取缓存区. 主要实现 2 个功能:

1) 通过重载 Xentrace 相关函数接收 VMI 调用参数,并将其记录在 trace buffer 中.

2) 在特权域 Dom0 协助调用基于 LibVMI 虚拟机自省程序. 向 VMI 启动模块传入虚拟域标志,新的 EPTP 字段等用于用户启动 VMI 的参数.

具体步骤设计如下:

① 在 Dom0 中启动 Xentrace;

② 当 VMI 启动模块被调用后,将接下来需要传入 VMI 程序的参数(Domain ID、新的 EPTP)写入到 Xentrace 的核心函数*TRACE_ND*()中;

③ 当 *TRACE_ND*()第 1 次执行时,将需要传入的参数写入到 Xentrace 在 Xen 堆上的 trace buffer 中;

④ 向 Dom0 注入虚拟中断 VIRQ_TBUF.

3.4 VMI 启动模块

该模块的主要功能主要是接收 VMI 调用参数并启动位于 Dom0 中的 VMI 程序. 具体步骤设计如下:

① 在 Dom0 中启动 Xentrace Daemon;

② Dom0 中的 Daemon 程序检测是否有新的虚拟中断注入,若发现有新的参数传入,则解析传入的参数,并通过所提供的内存页读取接口,读取目标虚拟机的内存;

③ 调用 Dom0 中的虚拟机自省程序.

4 实验与分析

4.1 实验环境

实验平台搭建于 CPU 为 Intel Core i7-6700K * 4 的物理主机上(已集成 VMFUNC 指令),其主频为 4 GHz,内存为 16 GB. 本文选择 Linux 3.14.60 作为物理机的操作系统,在 Xen Hypervisor 4.6.0 实现了本系统,同时为虚拟机分配单 vCPU,2 GB 内存.

首先编译并安装修改后的 Xen,包括 Xen,Xen Tools;然后安装 LibVMI. 对 LibVMI 进行适当修改后编译并安装监控服务. 加载编译好的驱动 VMI 偏移计算模块到 DomU 内核中. 为保证正常使用 Xen altp2m 机制,虚拟机中需配置"*altp2mhvm*=1",启用 Xen 硬件虚拟化的 altp2m 功能并设置虚拟机的影子内存"*shadow_memory*=16".

4.2 功能测试

考虑到多应用程序并行调用的情况,本文测量了单 vCPU 虚拟机能够达到的最大触发次数. 每次实验一共调用 1 000 次,共计 50 次实验,统计 VMI 的成功触发次数. 在高负荷切换的情况下,触发速率为 1 000 次/秒,触发率达到 100%. 因此可以认为该机制是可靠的.

4.3 性能测试

4.3.1 CPU 密集型程序性能影响

为了验证方案的正确性和测试本系统的性能,

本文实现了 2 套基于 LibVMI 的 VMI 程序,分别为 cMonitor[27]与 CAPT. cMonitor 通过在 Hypervisor 对 DomU 的监控,实现对网络数据包进行漏洞触发规则透明校验和过滤,提升云平台针对已知漏洞的快速免疫能力;CAPT 的功能则是持续透明地记录目标主机内的系统事件和网络事件,用于事后的攻击起源追踪. 在进行性能测试之前,本文对 VMI 的操作时间进行了测量,如表 2 所示,为后续实验分析提供量化参考.

Table 2 Average Execution Time of four VMI Programs
表 2 4 种 VMI 程序的平均操作时间

VMI Operation	Execution Time/ms
process-list	36.1
module-list	27.1
cMonitor	55.7
CAPT	74.8

本文选用 PARSEC 测试集[28]中的 blackscholes, canneal, dedup 和 streamcluster 作为基准测试程序. 同时考虑到实际应用情况,选用对安全需求敏感的 AES CBC-128 模式文件加密程序,RSA 1 024 b 字符串加密程序同样作为基准测试. 我们在 PARSEC 中的基准测试程序的源代码前插入了 VMFUNC 指令;在 OpenSSL 1.0.2g 中的 AES 和 RSA 加密程序进行了 VMFUNC 指令插桩. 对于 AES 加密程序,VMFUNC 指令被插入在每一个区块 block 的第 1 轮加密的开始位置. 这样每加密 1 KB 文本,有 32 条 VMFUNC 指令被执行;对于 RSA 加密程序,VMFUNC 被插入在函数 *bn_mod_mul_montgomery*() 的开始. 该函数是用于蒙哥马利模乘算法的核心函数. 同时,AES 加密程序密钥由伪随机数发生算法生成,加密对象为 1 KB 大小的二进制全"0"的明文文件. RSA 加密程序明文为 117 B 的全"0"字符串.

通过对 6 个不同的基准测试程序的 20 次重复实验,得到不同情况下程序的标准化运行时间. 我们对比了使用 LibVMI 的 SHM-snapshot[21]实现方案和传统的将虚拟机暂停的 VMI 实现方案. 当使用 CAPT 作为自省程序时,具体开销如图 6 所示:

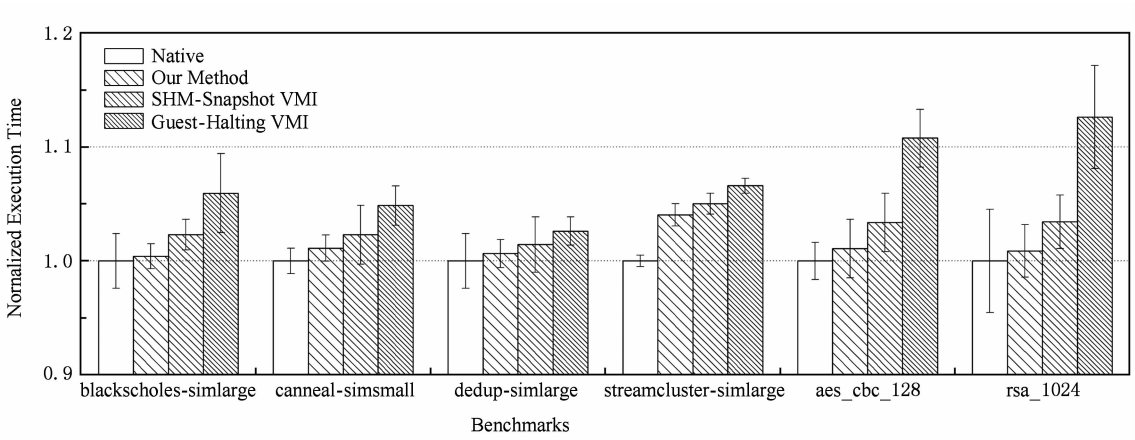


Fig. 6 Comparison of different benchmarks

图 6 基准测试程序耗时对比

测试结果表明:传统的将虚拟机暂停的 VMI 实现方案相对于无 VMI 情况下,系统消耗大大增加,远高于其他 3 种方案. 使用共享页面的 SHM-snapshot 方案采用了 Dom0 共享 DomU 页面的方式,可以将性能开销降至 5%以内. 而本方案由于 VMFUNC 的触发机制实现了 VMI 的按需调用,因此 VMI 的开销被有效抑制,相比于无 VMI 情况下,仅增加了不超过 2%的调用和切换耗时.

4.3.2 I/O 密集型程序性能影响

本文以云环境中应用范围最广的 Web-server 作为测试基准,测试了本系统对其性能影响. 我们采用 Apache+PHP 作为服务器的基础运行环境,在被监控虚拟机中部署 Web-server,并使用修改过的 OpenSSL 运行库,重新编译 Apache,生成已插桩的 SSL 模块.

使用包测试工具 Siege[29]进行压力测试,结果如图 7 和图 8 所示. 其中,每台服务器被 Siege 持续测试 30 min. 我们测试了标准化的单线程时 HTTPS 包响应时间和标准化的最大包处理能力.

4.4 现有 VMI 方法比较

本文所实现的 VMI 触发机制本质上属于改进的 Live memory reads 方法[30]. 我们从运行速度、资源消耗、虚拟机性能影响、EPT 页表切换连续性和兼容性 5 个角度来评估现有的 VMI 方法,如表 3 所示.

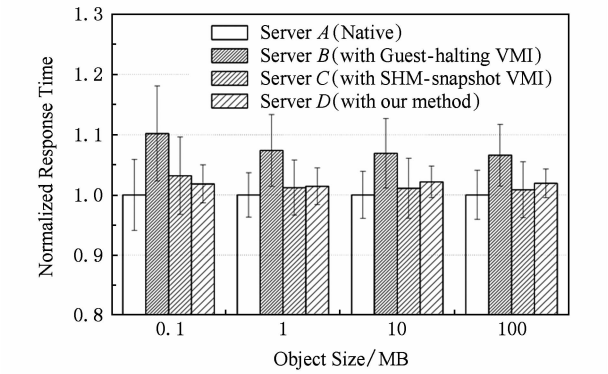


Fig. 7 Normalized response time in different object sizes

图 7 标准化包响应时间

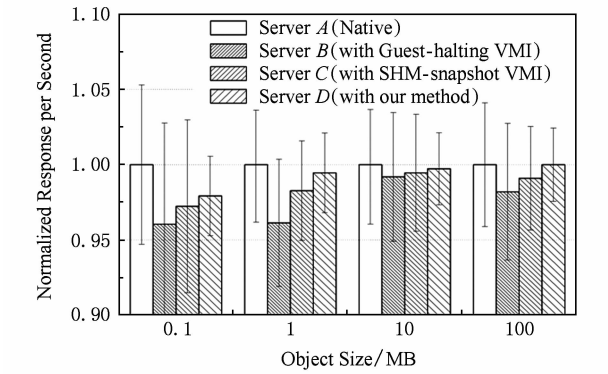


Fig. 8 Normalized response per second in different object sizes

图 8 标准化最大包处理吞吐率

Table 3 Comparison of the Existing VMI Methods

表 3 现有 VMI 方法比较

VMI Categories	Speed	Resource Cost	VM Performance Impact	EPT View Consistency	Compatibility
Guest-halting	Very low	Very high	Low	√	Default
Guest cooperation/ agent assisted access	Medium	Medium	Low	√	VM /dev/mem support or module installation
Halt snap	Low	High	High	√	Default
Live snap	Medium	Low	Low	√	Hypervisor modifications
Live memory mapping	High	Low	Very low		Hypervisor+Guest-OS modifications
Live Memory Reads	High	Low	Very low		Library+Hypervisor modifications
Our method	High	Low	Very low	√	Hypervisor modifications

其中,Guest-halting 方法效率最低,且无法做到实时自省. SHM-snapshot 方案即属于 Halt snap 方法. 作为 Live snap 方法的代表,文献[31]提出使用实时内部的内存快照作为自省程序的分析视图,能够为 KVM 创建不停止 Guest-VM 的内存快照,不用暂停虚拟机即可实现监控,一定程度上减少了开销. 但由于这些方法修改了 Hypervisor 的内存管理部分,因此兼容性并不好. 文献[32]为 QEMU 提供了从虚拟机到特权域的内存映射接口,加快了监控效率. 然而上述方案与 Hypervisor 的实现结合紧密,实现复杂,不具有较好的通用性.

Live memory mapping 的典型例子是 Xen 提供的内存页面共享机制,该方法的运行速度非常快,且对虚拟机影响很小;然而这种方法仅适用于 PV (Para-virtualization)虚拟机,兼容性差. 本文方法同样可以实时自省,且不会扰乱 Guest-VM 的执行,不过我们不通过基于文件描述符的方式来访问虚拟机的状态,而是提供一套副本 EPT 视图供 Dom0 审查. 这样既保证了运行速度,且无需修改 Guest-OS,在 Xen Hypervisor 中仅加入了近 60 代码行 SLOC (source lines of code),具有很好的透明性和可移植性.

5 结束语

在云计算大规模普及的今天,虚拟机技术越来越多地应用到了各种计算平台中. 然而由于缺乏高效而又可靠的虚拟机内存存取机制,传统虚拟机自省技术的实现一般开销过大. 本文从 Intel 处理器硬件新特性 VMFUNC 中获得灵感,将 VMFUNC 与虚拟机自省技术结合起来,提出了一种基于 CPU 硬件特性的虚拟机自省触发机制. 相比于其他 VMI 实现机制,本方案具有极小的时间开销和较小的空间开销,相较于正常程序的执行,仅引入了低于 2%性能开销,使 VMI 在实际环境中的广泛应用成为了可能.

下一步,我们会将 # VE 特性集成到本方案中,使系统更加高效并增加其稳定性. 同时寻求更高效的虚拟机自省方法来为云平台的安全运行提供保障.

参 考 文 献

[1] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection [C] //Proc of Network and Distributed System Security Symp (NDSS'03). Reston, VA: ISOC, 2003: 191-206

[2] Wu Rui, Chen Ping, Liu Peng, et al. System call redirection: A practical approach to meeting real-world virtual machine introspection needs [C] //Proc of Dependable Systems and Networks (DSN'14). Piscataway, NJ: IEEE, 2014: 574-585

[3] Nance K, Bishop M, Hay B. Investigating the implications of virtual machine introspection for digital forensics [C] // Proc of ARES'09. Reston, VA: ISOC, 2009: 1024-1029

[4] Li Wenhao, Xia Yubin, Chen Haibo, et al. Reducing world switches in virtualized environment with flexible cross-world calls [C] //Proc of the 42nd Int Symp on Computer Architecture (ISCA'15). Piscataway, NJ: IEEE, 2015: 375-387

[5] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization [J]. ACM SIGOPS Operating Systems Review, 2003, 37(5):164-177

[6] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual[M/OL]. Santa Clara, California: Intel Corporation, [2017-07-21]. https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-manual-325462.html

[7] Liu Yutao, Zhou Tianyu, Chen Kexin, et al. Thwarting memory disclosure with efficient hypervisor-enforced intra-domain isolation [C] //Proc of the ACM SIGSAC Conf on Computer and Communications Security (CCS'15). New York: ACM, 2015: 1607-1619

[8] Sharif M, Lee W, Cui Weidong, et al. Secure in-VM monitoring using hardware virtualization [C] //Proc of the ACM SIGSAC Conf on Computer and Communications Security (CCS'09). New York: ACM, 2009: 477-487

[9] Huang Xiao, Deng Liang, Sun Hao, et al. Secure and efficient kernel monitoring model based on hardware virtualization [J]. Journal of Software, 2016, 27(2): 481-494 (in Chinese)
(黄啸, 邓良, 孙浩, 等. 基于硬件虚拟化的安全高效内核监控模型[J]. 软件学报, 2016, 27(2): 481-494)

[10] Jones S, Arpacidusseau A C, Arpacidusseau R H, et al. AntFarm: Tracking processes in a virtual machine environment [C] //Proc of USENIX Annual Technical Conf (USENIX ATC'06). Berkeley, CA: USENIX Association, 2006: 1-14

[11] Dinaburg A, Royal P, Sharif M, et al. Ether: Malware analysis via hardware virtualization extensions [C] //Proc of the ACM SIGSAC Conf on Computer and Communications Security (CCS'08). New York: ACM, 2008: 51-62

[12] Dolangavitt B, Leek T, Zhivich M, et al. Virtuoso: Narrowing the semantic gap in virtual machine introspection [C] //Proc of IEEE Symp on Security and Privacy (Oakland'11). Piscataway, NJ: IEEE, 2011: 297-312

[13] Cristalli S, Pagnozzi M, Graziano M, et al. Micro-virtualization memory tracing to detect and prevent spraying attacks [C] //Proc of USENIX Security Symp (USENIX Security'16). Berkeley, CA: USENIX Association, 2016: 431-446

[14] Ahmed I, Richard Ⅲ GG, Zoranic A, et al. Integrity checking of function pointers in kernel pools via virtual machine introspection [C] //Proc of the Information Security Conf (ISC'13). Berlin: Springer, 2013: 1-16

[15] Wang Zhi, Jiang Xuxian. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity [C] //Proc of IEEE Symp on Security and Privacy (Oakland'10). Piscataway, NJ: IEEE, 2010: 380-395

[16] Vasudevan A, Chaki S, Maniatis P, et al. üBERSPARK: Enforcing verifiable object abstractions for automated compositional security analysis of a hypervisor [C] //Proc of USENIX Security Symp (USENIX Security'16). Berkeley, CA: USENIX Association, 2016: 87-104

[17] Jiang Xuxian, Wang Xinyuan, Xu Dongyan, et al. Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction [C] //Proc of the ACM SIGSAC Conf on Computer and Communications Security (CCS'07). New York: ACM, 2007: 128-138

[18] Wang Lina, Gao Hanjun, Liu Wei, et al. Detecting and managing hidden process via hypervisor [J]. Journal of Computer Research and Development, 2011, 48(8): 1534-1541 (in Chinese)
(王丽娜, 高汉军, 刘炜, 等. 利用虚拟机监视器检测及管理隐藏进程[J]. 计算机研究与发展, 2011, 48(8): 1534-1541)

[19] Li Baohui, Xu Kefu, Zhang Peng, et al. Research and application progress of virtual machineintrospection technology [J]. Journal of Software, 2016,27(6): 1384-401 (in Chinese)
(李保珲, 徐克付, 张鹏, 等. 虚拟机自省技术研究与应用进展[J]. 软件学报, 2016, 27(6): 1384-1401)

[20] Lengyel T K, Maresca S, Payne B D, et al. Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system [C] //Proc of Annual Computer Security Applications Conf (ACSAC'14). Piscataway, NJ: IEEE, 2014: 386-395

[21] Payne B D. Simplifying virtual machine introspection using LibVMI [EB/OL]. 2012 [2017-07-21]. https://prod.sandia.gov/techlib/access-control.cgi/2012/127818.pdf

[22] Payne B D, Martim D P A, Lee W. Secure and flexible monitoring of virtual machines [C] //Proc of Annual Computer Security Applications Conf (ACSAC'07). Piscataway, NJ: IEEE, 2007: 385-397

[23] Pfoh J, Schneider C, Eckert C. A formal model for virtual machine introspection [C] //Proc of the ACM Workshop on Virtual Machine Security (VMSec'09). New York: ACM, 2009: 1-10

[24] Vattikonda B C, Das S, Shacham H, et al. Eliminating fine grained timers in Xen [C] //Proc of the 3rd Cloud Computing Security Workshop. New York: ACM, 2011: 41-46

[25] Faggioli D. Tracing with Xentrace and Xenalyze [EB/OL]. (2012-09-27) [2017-07-21]. https://blog.xenproject.org/2012/09/27/tracing-with-xentrace-and-xenalyze

[26] Lengyel T. Stealthy monitoring with Xen altp2m [EB/OL]. (2016-04-13) [2017-07-21]. https://blog.xenproject.org/2016/04/13/stealthy-monitoring-with-xen-altp2m

[27] Zhang Hao, Zhao Lei, Xu Lai, et al. cMonitor: VMI-based fine-grained monitoring mechanism in cloud [J]. Wuhan University Journal of Natural Sciences, 2014, 19(5): 393-397

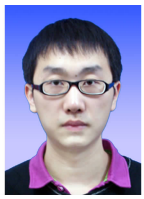
[28] Bienia C, Kumar S, Singh J P, et al. The PARSEC benchmark suite: Characterization and architectural implications [C] //Proc of Int Conf on Parallel Architectures and Compilation Techniques (PACT'08). Piscataway, NJ: IEEE, 2008: 72-81

[29] Siege. Siege-http load testing and benchmarking utility [EB/OL]. (2012-01-30) [2017-07-21]. https://www.joedog.org/siege-home

[30] Suneja S, Isci C, De Lara E, et al. Exploring VM introspection: Techniques and trade-offs [C] //Proc of ACM SIGPLAN/SIGOPS Int Conf on Virtual Execution Environment (VEE'15). New York: ACM, 2015: 133-146

[31] Cui Lei, Li Bo, Zhang Yangyang, et al. HotSnap: A hot distributed snapshot system for virtual machine cluster [C] // Proc of the USENIX Large Installation Systems Administration Conf (LISA'13). Berkeley, CA: USENIX Association, 2013: 59-73

[32] Roberts A, McClatchey R, Liaquat S, et al. Poster: Introducing pathogen: A real-time virtual machine introspection framework [C] //Proc of the ACM SIGSAC Conf on Computer and Communications Security (CCS'13). New York: ACM, 2013: 1429-1432



Liu Weijie, born in 1991. PhD candidate. His main research interests include virtualization security and cloud security.

Wang Lina, born in 1964. Professor and PhD supervisor. Senior member of CCF. Her main research interests include multimedia security and cloud computing security.

Tan Cheng, born in 1989. PhD candidate of Wuhan University. Student member of CCF. His main research interests include network security and cloud security.

Xu Lai, born in 1990. PhD. His main research interests include virtualization security and covert channel analysis.