软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(6):1384−1401 [doi: 10.13328/j.cnki.jos.005006]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

虚拟机自省技术研究与应用进展*

李保珲[1,2,3], 徐克付[2,3], 张 鹏[2,3], 郭 莉[2,3], 胡 玥[2,3], 方滨兴[1,2,3]

[1](北京邮电大学 计算机学院,北京 100876)

[2](中国科学院 信息工程研究所,北京 100093)

[3](信息内容安全技术国家工程实验室,北京 100093)

通信作者: 徐克付, E-mail: xukefu@iie.ac.cn

摘 要: 虚拟机自省技术是备受学术界和工业界关注的安全方法,在入侵检测、内核完整性保护等多方面发挥了重要作用.该技术在实现过程中面临的核心难题之一是底层状态数据与所需高层语义之间的语义鸿沟,该难题限制了虚拟机自省技术的发展与广泛应用.为此,基于语义重构方式的不同将现有的虚拟机自省技术分为 4 类,并针对每一类自省技术中的关键问题及其相关工作进行了梳理;然后,在安全性、性能及可获取的高层语义信息量等方面对这 4 类方法进行了比较分析,结果显示,不同方法在指定比较维度上均有较大波动范围,安全研究人员需综合考虑 4 类方法的特点设计满足自身需求的虚拟机自省方案.最后,详细介绍了虚拟机自省技术在安全领域的应用情况,并指出了该技术在安全性、实用性及透明性等方面需深入研究的若干问题.

关键词: 虚拟机自省;语义鸿沟;软件结构知识;硬件架构知识;安全应用

中图法分类号: TP309

中文引用格式: 李保珲,徐克付,张鹏,郭莉,胡玥,方滨兴.虚拟机自省技术研究与应用进展.软件学报,2016,27(6):1384−1401. http://www.jos.org.cn/1000-9825/5006.htm

英文引用格式: Li BH, Xu KF, Zhang P, Guo L, Hu Y, Fang BX. Research and application progress of virtual machine introspection technology. Ruan Jian Xue Bao/Journal of Software, 2016,27(6):1384−1401 (in Chinese). http://www.jos.org.cn/1000-9825/5006.htm

Research and Application Progress of Virtual Machine Introspection Technology

LI Bao-Hui[1,2,3], XU Ke-Fu[2,3], ZHANG Peng[2,3], GUO Li[2,3], HU Yue[2,3], FANG Bin-Xing[1,2,3]

[1](School of Computer Science, Beijing University of Posts and Telecommunication, Beijing 100876, China)

[2](Institute of Information Engineering, The Chinese Academy of Sciences, Beijing 100093, China)

[3](National Engineering Laboratory for Information Security Technology, Beijing 100093, China)

Abstract: Virtual machine introspection (VMI) has received much attention from both academic and industrial community, and plays an important role in intrusion detection, kernel integrity protection and many other areas. However, the semantic gap has greatly limited the development of this technology. In this respect, this paper divides existing VMI technologies into four categories based on the methods of semantic reconstruction, followed by the problems and their corresponding researches. Analysis results reveal the difficulties in meeting all the requirements. The paper therefore details the relevant applied research in security based on VMI. Finally, it presents the future research directions that need in-depth study, such as VMI's security, availability and transparency.

Key words: virtual machine introspection; semantic gap; knowledge of software structure; knowledge of hardware architecture; security application

* 基金项目: 国家自然科学基金(61402464); 国家高技术研究发展计划(863)(2015AA016005)

Foundation item: National Natural Science Foundation of China (61402464); National High-Tech R&D Program of China (863) (2015AA016005)

收稿时间: 2015-08-15; 修改时间: 2015-10-09; 采用时间: 2015-12-05; jos 在线出版时间: 2016-01-21

CNKI 网络优先出版: 2016-01-22 11:20:07, http://www.cnki.net/kcms/detail/11.2560.TP.20160122.1120.016.html

系统虚拟化是一种资源管理技术,其将计算机的各种实体资源(如 CPU、内存)予以抽象、转换后呈现出来,改变了系统软件与底层硬件紧耦合的方式[1].系统虚拟化技术支持同时运行多个操作系统,且每个操作系统中都有多个程序运行,本文将每个虚拟化操作环境(操作系统及其上运行的应用程序)称为虚拟机(virtual machine,简称 VM);将抽象底层硬件资源以供多个虚拟机使用的虚拟化软件称为虚拟机管理器(virtual machine monitor,简称 VMM).随着软硬件技术的高速发展,以 Xen[2]、KVM[3]、VMWare ESX/GSX/Workstation(http://www.vmware.com)等为代表的系统虚拟化技术极大地推动了由多任务计算到多操作系统计算的发展,在资源管理、服务器整合、提高资源利用率等方面发挥了巨大作用,已成为云计算架构中关键的抽象层次和重要的支撑性技术.第五界虚拟化会议暨博览会议的主题便为“用虚拟化改变数据中心、虚拟化支撑云计算”.

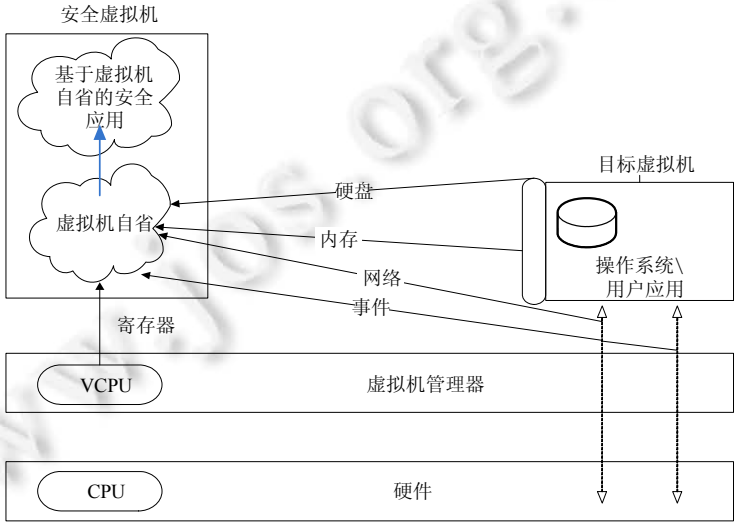

Fig.1 Architecture of VMI

图 1 虚拟机自省技术体系架构

除了上述优势外,系统虚拟化技术的隔离、更小可信计算基等特性也为开发可靠、鲁棒的安全工具提供了新思路,虚拟机自省技术(virtual machine introspection,简称 VMI)便是其中的典型代表.图 1 说明了 VMI 技术的体系架构(以 xen 为例,下文同),其通过在目标虚拟机(target virtual machine,简称 TVM)外部获取 TVM 底层状态数据(CPU 寄存器、I/O 控制器寄存器、内存、大容量存储设备以及虚拟 BIOS 等),可以在安全虚拟机(secure virtual machine,简称 SecVM)中有效地监控或干预其内部运行状态.自 2003 年 Garfinkel[4]首次提出虚拟机自省概念至今,基于 VMI 技术的安全研究越来越多,在恶意软件分析、内核完整性检测及云计算安全等方面均发挥了重要作用.比如,传统的安全软件放置在虚拟机内部,且一般与恶意软件运行于同一特权级,因此,容易成为恶意软件的首要攻击对象;然而,VMI 技术具有隔离性、自省性以及干预性[5],能够实现安全软件与恶意软件的分离,因此,其可以有效应对直接攻击问题带来的挑战.截至目前,已有商业或开源软件库实现了上述功能,如 vmsafe(http://www.vmware.com/company/news/releases/vmsafe_vmworld),Volatility(https://code.google.com/p/volatility/),LibVMI[6].但在 TVM 外部获取的底层状态数据是以二进制形式呈现的,我们无法获悉其高层语义信息,这种语义之间的差异称为语义鸿沟,由二进制形式的底层状态数据转换为高层信息的过程即为语义重构,在将底层状态数据转换为相应高层语义信息时所借助的软硬件知识便为语义知识[5].

例 1:本文由虚拟地址 ffff880039384c78 开始获取 48bit 二进制串 011001110110010101110100011101000110001110001111001,用户很难从该底层状态数据得出可用的信息.借助内核数据结构 task_struct 等知识(解析过程详见算法 1),可以获知此二进制串的高层语义是 pid 为 1068 的进程名 gettty.将这 48bit 二进制串解析为进程名 gettty 的过程即为语义重构,转换过程中所需的内核数据结构 task_struct 等知识就是语义知识.

由此可见,语义重构是VMI技术的关键步骤,其在性能、复杂度及安全性等方面对VMI技术均有较大影响,因此,深入理解不同的语义重构方式将有利于安全研究人员在特定的限制条件下设计满足其需求的虚拟机自省技术.然而,以往的研究工作一般集中在VMI技术实现或应用的某个侧面,虽然有学者对基于虚拟化的相关研究进行总结,但其或者只分析了其应用而并未阐述原理[7],或者只是针对基于内存的自省方式进行深入研究[8],缺乏全局分析.为此,本文第1节从一个更深入的视角对虚拟机自省技术进行分类分析,并着重对每类VMI技术中的语义重构方式、面临问题及相应解决方案进行分类梳理,然后,对各类自省方法进行比较总结,分析结果显示这4类方法在安全性、实用性及可获取的高层信息量等方面有较大的波动范围.鉴于其在安全领域的巨大贡献,本文第2节介绍VMI技术在安全领域的应用.第3节对后续研究工作给予展望.第4节总结全文.

1 虚拟机自省技术分析

语义重构有3个核心要素,即底层状态数据、高层语义信息及语义知识,其中,语义知识是连接底层状态数据和其相应高层语义信息的桥梁,决定了语义重构的方式和过程.本文将当前可用于语义重构的语义知识分为软件结构知识和硬件架构知识两类,其中,软件结构知识主要包括操作系统和其他软件程序中的数据结构及部分数据元素存储地址等方面的知识,其对应的底层状态数据一般以特定的数据结构(数组、链表、树等)存储在空间较大的内存、硬盘等设备中,如例1中的进程描述符task_struct,内核以双向链表的形式将该类数据元素存储于内存中,且该链表的头元素由位于System.map文件中的init_task标识;硬件架构知识主要包括硬件寄存器等设备的功能知识,其对应的底层状态数据一般从空间较小的寄存器等设备上获取(如CPU寄存器等),且该设备的功能较为固定,如cr3寄存器的功能是保存当前运行进程页目录表基地址.

实际上,我们使用高级语言编程时也会面临语义鸿沟,但是,由于操作系统及运行于其上的应用软件自动完成了语义重构且将这一过程封装起来,以至于我们"日用而不知".比如,基于C语言定义变量*uint32_t a*=1便包含了3层含义:① *a*的数据类型为unsigned int,② *a*在内存中的存储地址为*&a*,③ *a*在内存中所占据空间长度为32bit,计算机基于这3项知识才能正确地将*a*的数值呈现给用户.再比如,操作系统在需要获取当前进程页目录表地址时便会读取CR3寄存器的值,但是该过程普通用户是无法感知的.部分研究工作借助TVM或SecVM的已有语义重构功能直接获取高层语义信息,从而规避了语义鸿沟问题;与之相反,部分研究工作则独立设计语义重构程序完成自省任务.基于此,依据面临语义鸿沟问题时是否采用规避策略,本文将现有VMI技术分为两类,一类是借助TVM或SecVM上已有的语义重构工具,规避语义鸿沟问题,我们称其为依赖型虚拟机自省方法;另一类则相反,其设计独立的语义重构工具,我们称为独立型虚拟机自省法.对于依赖型虚拟机自省法,本文又依据其借助目标对象的不同,将其分为基于TVM的自省方法和基于SecVM的自省方法;对于独立型虚拟机自省法,本文依据语义重构过程中所借助语义知识的不同,将其分为基于硬件架构知识的虚拟机自省法和基于软件结构知识的虚拟机自省法.

下面详细阐述这4类虚拟机自省方法的工作原理、每类方法所面临的主要问题及其相应的研究工作,最后对这4种自省方法进行比较分析.

1.1 基于目标虚拟机的依赖型自省法

1.1.1 方法概述

基于TVM的虚拟机自省法的体系架构如图2所示,首先由位于TVM内的数据捕获模块获取高层语义信息后,由数据传递模块将信息传递至SecVM中,完成自省、监控工作.由此可以看出,数据捕获模块需借助TVM完成二进制数据捕取及语义重构工作.本方法具有以下特征:

(1) 基于该方法的监控工作可以直接对接收到的数据进行分析,而无需关心语义重构等过程,因此,该方法耗时少、自省效率高;

(2) 由于该方法需借助目标虚拟机获取高层语义信息,因此其不支持分析处于停止状态的虚拟机;

(3) 由于该方法需要修改客户虚拟机,故对目标虚拟机不具有透明性,也因此限制了它的使用范围.

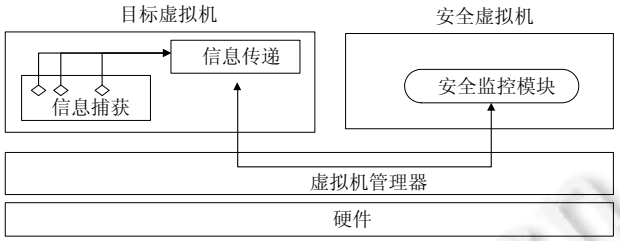

Fig.2 Architecture of VMI based on target VM

图 2 基于 TVM 的虚拟机自省法架构图

1.1.2 主要问题及相关工作

如何设计良好的数据捕获工具是该方法的关键问题之一.向 TVM 中有意义的系统执行点插入挂钩函数,将控制流导向安全工具,是常用的获取 TVM 高层语义信息的方法.在这方面,Lares[9]将钩子函数以及信息传送代码植入 TVM 中,挂钩函数被触发后会将控制流转移到安全工具以执行安全监控,SIM[10]也将钩子函数等信息捕获工具部署在 TVM 中.与上述方法类似,在 TVM 中有意义的位置插入不同于钩子函数的信息获取和传送代码,也可以实现信息捕获和传递功能.文献[11]将嵌入 TVM 中的信息获取和传送代码称为 sensor,其会在可疑或恶意行为发生时被触发,然后收集触发事件的位置、参数等信息,传递给 SecVM.针对 x86 架构而设计的 X-TIER[12]则是将信息获取功能以内核模块的形式注入 TVM 中,通过访问客户操作系统的数据结构等获取虚拟机的状态信息,最后以 Hypercall 的形式将获取的信息实时传递给 VMM.与上述方法不同,文献[13]通过将 SecVM 内监控进程的系统调用重定向至 TVM 中的伪进程(dummy process),然后由伪进程代理执行,也达到了收集 TVM 的状态信息的目的;SYRINGE[14]则是将注入上下文(injection context)植入 TVM 的伪进程中,当执行到注入上下文时,便会捕获高层语义信息并通过函数调用注入(function-call injection)的方式将捕获数据传递给位于 SecVM 中的监控软件.

由于数据获取或传递程序在 TVM 中,所以其自身容易受到 TVM 中恶意软件的直接攻击,因此,如何确保信息获取和传递代码的安全是该类方法的另一个关键问题.在这方面,Lares[9]提出了内存写保护的方法,SIM[10]则是利用处理器的临时硬件内存保护和硬件虚拟化机制设计了受控的地址空间,以保证插入代码的安全;文献[13]基于内存虚拟化技术等保护伪进程中嵌入代码的安全性:与上述方法思路不同,SYRINGE[14]基于 Localized shepherding 技术监控数据获取代码的控制流完整性.文献[15]首先在 TVM 中随机选取一个处于运行态的进程 p,向 p 植入待保护代码,然后借助进程 p 执行待保护代码,由于恶意软件无法获知待保护代码的具体位置 4,从而无法对其实施攻击.

1.2 基于安全虚拟机的依赖型自省方法

1.2.1 方法概述

基于 SecVM 的虚拟机自省方法的体系架构如图 3 所示,该方法需在 SecVM 中运行与 TVM 中待监控进程相同的模拟进程,同时将 TVM 中待监控进程的相关底层状态数据实时移入 SecVM 中,以确保 SecVM 中已有的检测工具(如 strace、ltrace 等)对模拟进程的分析结果可以同步反应待监控进程的运行状态.

本方法具有如下特征:

(1) 检测工具位于 SecVM 中,可以免受待监控进程的直接攻击等影响,安全性较高且具有良好的透明性;

(2) 该方法无法同时对 TVM 的整个状态进行监控,而只能监控某一指定进程;

(3) 该方法要求 SecVM 与 TVM 的操作系统及安全工具相同,因此其客户操作系统可移植性受到了限制.

1.2.2 主要问题及相关工作

根据自省目的,采用什么途径将 TVM 的何种数据移植入 SecVM 中,是该方法面临的首要问题.以文献[16]为例,其目的是让 SecVM 中的 ltrace 及 gdb 等工具发挥作用,鉴于该类监控工具主要监控用户态代码的执行,故其只对用户态代码需要访问的数据进行移植,如用户态的进程上下文(如寄存器内容)、内存等数据;对该类监控

工具关系不大的内核态代码等数据,并未将其列入移植范围;在移植途径方面,其为了减小了数据迁移的时间开销,采用内存数据映射,而没有选择数据拷贝的方式.

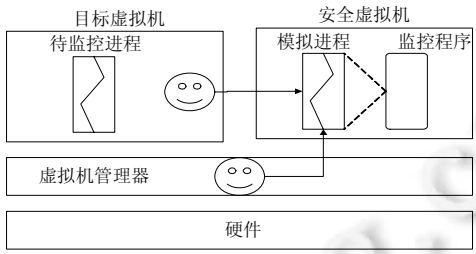

Fig.3 Architecture of VMI based on secure VM

图 3 基于 SecVM 的自省方法架构图

如何同步模拟进程与待监控进程是该方法面临的另外一个关键问题.文献[16]借助 Intel 处理器的硬件虚拟化机制在用户态时执行数据迁移,并采用系统调用重定向及页错误转发等技术保证模拟进程的运行环境、执行状态等与待检测进程相同,实现了模拟进程与待监控进程的同步,使得对模拟进程的检测结果可以实时反映待监控进程的运行状态.

1.3 基于软件结构知识的独立自省法

1.3.1 方法概述

基于软件结构知识的独立性自省方法的体系结构如图 4 所示,通过在 TVM 外部提取底层状态数据,借助软件结构知识重构出高层语义信息.该方法的关键是由软件结构知识获取两类信息:① 计算机组织、存储待重构数据元素的方式(如数组、链表、树、图等),② 待重构数据元素的三元组信息〈数据元素类型、起始地址、偏移量〉,其中,数据元素类型可以是系统定义的整型、长整型等,也可以用户自行定义良好的类型,如结构体等;起始地址表示该数据元素在物理设备的存储地址;偏移量是指该数据元素所占物理地址空间的大小.为此,本文将其划分为学习阶段和搜索阶段,其中,学习阶段负责获取待重构数据元素的初始地址、偏移量及数据类型等信息,主要的学习方法包括人工数据结构标签、源代码分析、动态学习 3 类方法[17];搜索阶段包括线性扫描或递归遍历,两个阶段交替循环执行、直至重构完成数据结构中的每个数据元素.该方法可以获取进程列表[18,19]及已加载的内核模块等信息,下面以读取内存数据、重构 TVM 中所有进程名信息为例说明如何使用该方法.

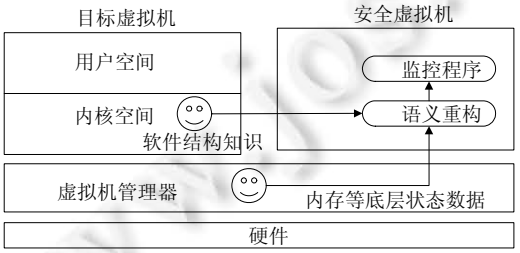

Fig.4 Architecture of VMI based on knowledge of software structure

图 4 基于软件结构知识的虚拟机自省法架构图

例 2:task_struct 结构描述了单个进程的详细信息,Linux 内核用双向循环链表将各 task_struct 组织起来(知识 1),其链表头虚拟地址存储在 System.map 文件中,由符号 init_task 标识(知识 2),相应的底层状态数据存储于内核内存中(底层状态数据来源).

算法 1. 进程名提取过程.

输入:*System.map task_struct init_task*;

输出:*all process name*.

```
1. name_off←parse_name_off(task_struct);
2. name_len←parse_name_len(task_struct);
3. pointer_off←parse_pointer_off(task_struct);
4. pointer_len←parse_pointer_len(task_struct);
5. vm_addr←get_vm_addr(System.map,init_task);
6. list_head←vm_addr;
7. phy_addr←vddr2paddr(vm_addr);
8. memory←get_memory(vmID,phy_addr);
9. DO{
10.   p_name←parse_process(memory,name_off,name_len);
11.    Printf(p_name);
12.    next_entry_addr←next_entry_addr(memory,port_off,port_len);
13.    phy_addr←vddr2paddr(next_entry_addr);
14.    memory←get_memory(vmID,phy_addr);
15. }WHILE(next_entry_addr !=list_head);
```

算法 1 描述了利用该方法提取内核中所有进程名的过程,其中,第 1 行~第 4 行依据 task_struct 进程结构体获取进程名以及 next 指针的获取相对于结构体起始位置的偏移量、长度信息,第 5 行查找 System.map 文件中 init_task 对应的虚拟机地址;第 7 行完成 3 级地址转换(虚拟机虚拟地址、虚拟机物理地址、物理机机器地址),第 8 行依据机器地址 phy_addr 读取内存,第 10 行、第 11 行分析内存得出当前进程名并输出,第 12 行、第 13 行完成 next 值的读取和地址转换,然后读取下一项对应的机器内存,如此循环,直至遍历结束.

经上述分析,可以得出本方法具有以下特征:

(1) 由于在 TVM 外完成底层状态数据获取及语义重构等工作,因此该方法具有较好的透明性;

(2) 该方法可以获取的高层语义信息范围较大,包括进程列表、已加载内核模块及开启的 socket 和文件信息等;

(3) 该方法一般基于特定的软件结构知识设计自省程序,因此,软件架构知识的变化会导致自省程序失效,故该方法的客户操作可移植性也较差;

(4) 攻击者通过改变软件结构的方式,可以使得自省程序失效,即该方法易受攻击、鲁棒性不高[20].

1.3.2 主要问题及其相关工作

搜索阶段一般是从具有存储地址标识符(见算法 2 中的 init_task)的数据元素出发,沿着指针或引用的方向寻找下一数据元素,直至所有待重构数据元素分析完毕.由此可以看出,指针或引用在分析过程中扮演了承上启下的重要作用,但是内核中存在大量的通用指针(void*)、联合体以及动态数组等不易被识别的指针类型,加之搜索具有递归特性,因此,容易造成错误的积累和传播.以算法 2 为例,若在分析进程结点 *n* 的 *next* 值时发生错误,则取进程结点(n+1)的内存数据时出错,最终后续的所有结点($n+i$,i=2,3,…)均会出现分析错误,因此,如何正确辨析指针类型是该方法需解决的首要问题.为此,文献[21]设计了内核对象指针 KOP 系统,该方法首先利用域敏感、上下文敏感的 points-to[22]方法生成 points-to 有向图,从而得到通用指针的候选目标类型;然后构建包含类型定义、全局变量等信息的有向拓展类型图;最后,KOP 基于有向拓展类型图、内核对象大小、特定类型存储特征以及内核的内存池界限等来解决类型模糊问题.文献[23]利用 used-as 分析方法来揭示结构体、联合体域以及全局变量的实际用途与其声明类型之间的关系,其通过检测赋值以及从左到右的映射操作初始化 points-to 图,对每次映射操作的解引用层次进行记录,得到 points-to 图的传递闭包;然后在复制、指针类型转换及函数参数赋值时,将全局变量或者结构体域的实际应用类型和其声明类型进行比较,以建立变量和类型上下文敏感的

used-as 关系.MAS[24]将指针分析形式化为上下文无关的语言可达性问题,利用新型的需求驱动指针分析方法,快速、准确的识别出通用指针的候选类型,同时也对通用指针、数据类型之间关系的识别有很好效果.

该方法面临的第 2 个问题是要求自省程序的输入参数应与软件结构知识相匹配,例如,由于不同内核版本的 init_task 值可能不同,若算法 2 作用于不同版本的 Linux 内核时输入参数没有随之改变,但是由于操作系统版本变更或者打补丁等因素,经常导致自省工作无法顺利完成,本文称此类造成自省程序失效的问题为输入参数问题.针对输入参数问题,文献[25]基于开源内存分析软件 Volatility 提供的丰富操作系统分析接口,开发了实时的内核数据结构监控系统,可以有效应对待检测数据结构不断变化带来的不便.文献[26]的方法与其不同,在虚拟机启动之前,其首先基于对硬盘数据的自省结果识别操作系统的版本号,然后定位操作系统内核文件,最后由微软公共符号服务器(Microsoft's public symbol server)获取正确的操作系统数据结构定义信息.

另外,自省程序本身的控制逻辑也应与软件结构知识相对应,如内核对打开端口与进程的管理结构不同(打开端口以哈希表的形式组织在一起),因此,算法 2 不能分析内核中打开端口的情况,需要重新设计自省程序,本文称此类造成自省程序失效的问题为控制逻辑问题.应对该类问题,需要大量人工参与,耗时且耗力,因此,如何自动生成自省代码、减少人工干预,是该方法面临的一个重要课题.针对控制逻辑问题,Virtuoso[27]将待检测指令,如 ps、netstat,在虚拟机内执行多次,只抽取和运算有关的指令并且生成指令路径;然后采用动态分片法抽取生成自省代码所需路径片段;最后将路径片段融合、转译为可以在虚拟机外执行语义重构的代码.针对 virtuoso 需要反复训练并且不能完全自动化的缺点,VMST[28]利用了内核重定向技术将数据访问重定向到目标操作系统的内存中来生成自省程序,其主要原理是:一个程序通常由代码 P 和数据 x 组成,P 在不同的机器里面通常是相同的,但运行时用到的 x 通常不同,因此,如果可以在机器 A 上运行代码 P 并使用机器 B 上的数据 x,就可以得到自省程序 P',使得 $P'(x)=P(y)$.但是 virtuoso、VMST 性能损耗较大,为此,文献[29]提出将执行和训练记忆解耦,性能有了较大的提高.

最后,该方法对利用特征变换等技术躲避检测的内核对象不具有较好的识别效果,如文献[30]指出恶意攻击者通过对进程特征做较少的改动便可成功躲避该自省方法的检测.但是,恶意软件经常会采用特征变换的方式达到免检测的目的,因此,有效识别内核中大量存在的隐藏的内核对象具有重要的意义.针对特征变换引起的问题,文献[31]首先监测系统的执行过程并标记读写的目标对象,然后分析出可被恶意攻击者更改却不影响正常运行的部分,最后生成了具有高鲁棒性的特征.然而,该方法只利用结构体中的不变量来生成规则集,没有利用结构体之间的关系,为此,文献[32]利用 Points-to 关系生成结构化签名的有向图,提高了内核数据结构实体查找的精确性.

1.4 基于硬件架构知识的独立型自省法

1.4.1 方法概述

在虚拟化系统中,TVM 与 VMM 分别运行于非特权态和特权态,当 TVM 访问敏感资源时会陷入 VMM 中,VMM 便介入了客户虚拟机与硬件资源之间的交互过程,可获取 CPU 寄存器等设备中的相关底层状态数据.基于硬件架构知识的独立型自省方法的架构如图 5 所示,VMM 通过在 TVM 外获取特定硬件设备的状态数据,然后借助硬件架构知识,推理出高层语义信息.基于硬件架构知识的独立型自省方法可以重构进程[33]、系统调用[34]等相关信息,下面以监控进程创建为例说明该方法的工作过程.

例 3:Linux 进程均有各自的虚拟地址空间,并由 CR3 寄存器存放进程页表基地址,因此,对 CR3 寄存器的改写表明新的页目录被载入.由于 CR3 是特权寄存器,TVM 对其的写事件会自动陷入到 VMM 中,从而被 VMM 主动捕获(硬件架构知识).由上述语义知识可以得出,需要的数据资源包括 CR3 寄存器内容等(底层状态数据来源).若 VMM 捕获 TVM 对 CR3 特权寄存器的写事件且写入内容为新的地址空间标示符,便可推理产生了新进程[33](推理分析).

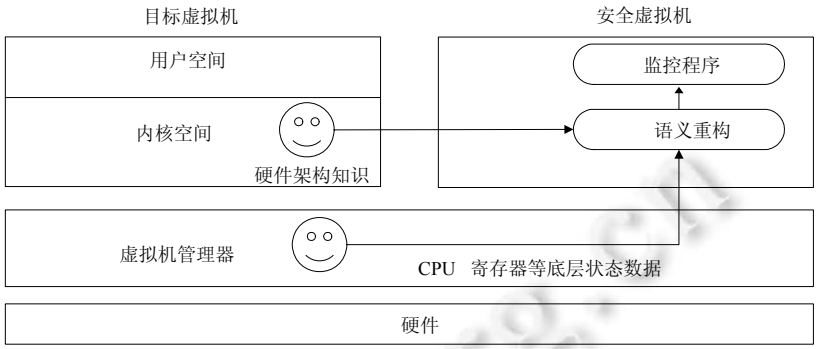

Fig.5 Architecture of VMI based on knowledge of hardware architecture

图 5 基于硬件架构知识的虚拟机自省法架构图

算法 2. 监控进程创建过程.

输入:*NULL*;

输出:是否生成新进程.

```
1. WHILE(1){
2.    IF (cap_write_cr3()==TRUE){
3.       asid←get_cr3_content();
4.       IF (exist(asid_list,asid)==TRUE){
5.           Process_switch;
6.       }ELSE{
7.           Insert(asid_list,asid);
8.          New_process;
9.     }END IF
10.    }END IF
11. }END WHILE
```

算法 2 描述了该方法监控进程创建的过程,其中,第 1 行表示持续监听写 CR3 寄存器事件,第 2 行~第 10 行表示若检测到此事件,则读取写入 CR3 寄存器的内容 asid,第 4 行表示检测记录链表 asid_list 是否已经存在当前的 asid 值,若存在则说明发生了进程切换,否则生成了新进程,并将当前 asid 插入记录链表 asid_list 中.

经上述分析,可以得出该类方法具有以下特征:

(1) 其在 TVM 外完成底层状态数据获取及语义重构等工作,因此,该方法具有良好的透明性;

(2) 由于硬件架构知识变动较少,如 linux 的不同版本均利用 CR3 寄存器存储进程页表物理基地址,因此,该方法的客户操作系统可移植性(guest OS portability)较好[30];

(3) 由于恶意软件较难更改客户操作系统的硬件架构,因此该方法的安全性较强;

(4) 由于可用的硬件架构知识及其可用的硬件资源有限,与基于软件结构知识的自省方法相比,该方法可获取的高层语义信息较少.

1.4.2 主要问题及其相关工作

该方法的一个主要问题是可利用的硬件架构知识及基于其可获取地高层语义信息均有限.文献[35]从 X86 架构固有特性等方面总结了部分可用于该方法的事件和硬件信息,并给出了它们和软件结构知识相结合以扩展其自省能力的方法.表 1 列出了可用的陷入事件和硬件信息,其中,虚拟机状态信息列是指可以直接访问且对 VMI 有支撑意义的设备信息,虚拟化扩展信息列是指可以利用的 TVM 与 Hypervisor 之间的交互事件,系统中断使能列可以用来迫使无法自动陷入的事件陷入 Hypervisor.

Table 1 Hardware and events of X86 architecture for VMI

表 1 X86 架构可用于虚拟机自省的硬件、事件信息

VM state access	Virtualization extensions	System interrupt-enable mechanisms
CR3	Context switch trapping	Avoiding countermeasures
IDTR and system call MSRs	VM entry trapping	Page-Fault exception
GDTR	System interrupt trapping	General protection exception
LDTR	User interrupt trapping	Invalid opcode exception
Virtual I/O		Debug exception

系统调用被广泛用于推断、观察和理解进程行为,因此,从 TVM 外部获取其的系统调用信息,对于行为监控具有重要意义.但是,流行的指令集架构(如 Intel IA-32、Intel 64)不支持系统调用自动陷入 VMM[36],因此,如何迫使其陷入 VMM 是该方法面临的另一个难题.针对此问题的一般思路是利用产生异常的方式创造自动陷入 VMM 的条件,实现系统调用的主动捕获.文献[37,38]提出了基于 x86 架构的快速系统调用进入(fast system call entry)机制创造陷入条件的方法.以 Ether[37]为例,其首先将 SYSENTER_EIP_MSR 寄存器设为不合法的数值,并将正确数值存储于 Ether 的内存中,迫使系统调用发生时出现页错误而陷入 Hypervisor 中.与文献[37,38]不同,Nitro[36]的陷入方法分为基于中断的系统调用陷入、基于 SYSCALL 的系统调用陷入、基于 SYSENTER 的系统调用陷入 3 类(以 Intel 为例),文献[39]的思路与 Nitro 类似,利用通用保护、UD 异常等,使 VMM 截获无法自动陷入的指令,然后利用有关 VCPU 的异常陷入信息上下文进行识别.

1.5 比较与总结

表 2 从安全性、可移植性、所用语义知识等多方面对上述 4 种方法进行了比较分析.

Table 2 Comparison of four methods

表 2 4 种虚拟机自省方法的比较

	依赖型虚拟机自省方法		独立型虚拟机自省方法	
	基于目标虚拟机	基于安全虚拟机	基于软件结构知识	基于硬件架构知识
高层语义信息量	不定	不定	较多	较少
底层状态信息	不全	全	全	全
安全性	低	高	较低	高
耗时	少	较少	多	较少
客户 OS 可移植性	差	差	较差	较好
透明性	差	较好	好	好
语义知识	不定	不定	软件结构知识	硬件架构知识
隔离性	差	较好	好	好

表 2 中的比较内容在本节前 4 部分均有分析,不再赘述.由表 2 可以看出,由于其在实现原理及实现方式的不同,不存在一种语义重构方式可以在所有特性均处于优势.本文又选取可以覆盖 4 类自省方式的 5 种已有实现方法,从支持操作系统类型、可靠性等 4 个维度对其进行了对比分析,分析结果见表 3.

Table 3 Comparison of five methods

表 3 5 种实现方法的比较

方法	支持的 OS 类型	可靠性	开发监控工具	访问 TVM
VMI-Based IDS[5]	Linux	高	难	需要
LibVMI[6]	Windows,Linux	高	容易	需要
Process out-grafting[15]	Linux,Windows	较低	容易	需要
Virtuoso[24]	Windows,Linux	较低	较难	不需要
Antfarm[30]	Windows,Linux	高	一般	不需要

表 3 的分析结果再一次证明不存在一种实现方式可以同时满足所有的设计需求.比如,文献[5]最早实现了基于 VMI 的入侵检测系统,具有较高的可靠性,但很难基于其再次开发监控工具;LibVMI 可以支持多样的操作系统,也方便了监控工具的开发,但其需要访问 TVM 运行脚本以获取部分数据结构偏移量信息;Process out-grafting 方便了监控工具的开发,但是需要频繁访问 TVM;Virtuso 生成自省程序较为简单,但是其可靠性并不高;Antfarm 虽然具有良好操作系统可移植性,但其获取的高层信息有限,因此难以基于其开发监控工具.

由此得出,在设计 VMI 的实现方式时需要根据安全应用的实际需要综合考虑各实现方法的优缺点,以达到

整体效果的最优.本文认为应根据 TVM 的入侵容忍程度、性能、可移植性、可靠性以及一致性等方面的不同需求设计 VMI 的实现方式,特别的,本文总结了在选取实现方式时应注意的两个方面,以期引起大家的注意.

(1) 受其自身特性的限制,特定 VMI 实现方式存在不能满足某种应用需求的可能,如因为没有硬盘文件系统结构的辅助不可能精确抽取特定硬盘扇区的数据,因此,基于硬件架构知识的方法将难以检测到硬盘文件的变化情况.因此,应根据应用场景以及安全需求的不同,综合考虑高层语义信息、底层状态资源以及语义知识等因素,恰当选择语义重构的某一种方式或者组合用之.

(2) 组合使用时需考虑各重构方法之间的相互关系:① 某种方式的优点不会弥补另外一种方式的缺点,如将方法 1 和方法 4 结合使用时,由于方法 1 的存在,会导致整个方案的可移植性较差(短板效应);② 某方式的缺点也可能会较好地兼容其他方式的优点,如,虽然方法 1 具有安全性不高,但其与方法 3 结合就能达到检测是否存在隐藏进程的目的(详细分析见第 3.1 节).

2 虚拟机自省技术的应用进展

由于 VMI 技术具有隔离性以及可干预性[5],基于其重构出的高层语义信息可以进行多方面的安全应用研究.本文依据应用目标的不同,从内核完整性检测、内核完整性保护、恶意代码行为分析、入侵检测及数字取证 5 个方面,说明基于 VMI 技术的安全作用.近年的工作倾向于将 VMI 技术应用于具体的应用场景,本节最后给出了此方面的代表性工作.

2.1 目标虚拟机内核完整性检测

虚拟机的内核,一旦被入侵,包括安全软件在内的所有软件均面临很大的风险.目前已有多种内核攻击技术,可以入侵内核并开启后门[40],如系统调用劫持技术、内核对象劫持技术及直接操纵内核对象技术(DLKM)等,因此,内核完整性检测的重要性越来越突出.由于 VMI 技术可以有效监控 TVM 内部的事件及运行状态,基于其获取的高层语义信息可以有效检测 TVM 的内核完整性是否遭受破坏.

TVM 的部分内核状态在运行期间始终不变,检测其是否发生改变即可判断内核完整性是否遭受破坏.这方面基于 VMI 的代表性工作有文献[41−45].其中,SBCFI[41]在系统运行期间对 TVM 中可用的函数指针及其指向目标代码进行垃圾回收式的遍历、检测;文献[42,43]则对 TVM 内核的数据结构不变量进行检测.然而,由于 TVM 内核也有部分状态在运行时经常发生变化,比如创建进程,造成前述方法的检测效果较差.为此,HIMA[44]实时捕获 TVM 中发生的事件和输入输出信息,进而判断事件对虚拟机状态的影响,同时依据变化及时调整测量方法以适应待检测状态的不断变化.

特别地,隐藏进程检测是内核完整性检测的一个重要方面.文献[45,46]利用 VMI 技术生成外部语义视图后,与 TVM 内产生的视图进行比较,可较好地识别恶意隐藏进程.但是,在虚拟机外部观测并产生视图耗时较长,而内部视图的生成耗时较短,基于此,二者存在一定的时间差,造成内外视图比较法的检测结果存在一定的误差.为此,Lycosid[47]提出了一种与内外视图比较法不同的检测思路,其利用假设检验方法探测 TVM 是否含有隐藏进程,并利用最小二乘回归分析法对 CPU 的运行时间等进行分析.

2.2 目标虚拟机内核完整性保护

本文将对 TVM 内核的攻击分为两类:① 通过篡改内核代码、内核函数指针和跳转指令等更改控制流,② 篡改内核动态数据,两者都可以破坏内核数据的完整性.因此,本文从 TVM 内核的数据完整性保护和控制流完整性保护两方面,说明 VMI 技术在内核完整性保护方面的作用.

内核数据在位置、内容以及数量等方面具有动态性,且与部分驱动程序同处于位置较为固定的内核内存空间,因此,攻击者可以较为容易的发现并篡改内核中的关键数据.为此,Sentry[48]将整个 TVM 内核的地址空间划分为不同区域,并且根据各区域内数据敏感性的不同,设置不同的访问控制策略,VMM 截获向敏感区域写数据事件后,判断该事件是否符合安全策略,有效保护了内核的完整性.刘宇涛等基于硬件提供的事务内存机制,从主动和被动两个方面对 TVM 的内核实体进行保护[49].

控制流的完整性要求代码的执行路径正确并且完整,禁止执行不合法的代码或按照不合法的顺序执行代码.限制只有经过验证的代码才可以具有运行权限,是确保内核完整性的常用方法,基于VMI的代表性工作有文献[50−52].其中,Secvisor[50]要求用户对将在TVM内核模式下执行的代码进行验证;Nickle[51]执行内核代码验证以确保只有经过验证的内核代码才可以存储在影子内存中,并在TVM运行时将指令路由至影子内存中获取执行代码,从而保护TVM内核的完整性;Patagonix[52]通过验证可执行代码的加密哈希值的方式对TVM内核实施保护.

特别地,TVM 内核中的钩子函数和指针是易受攻击的对象,是控制流完整性防护的一个重要方面,部分工作集中在此.HookLocator[53]利用VMI技术监控对内核函数指针的更改操作,其基于开源的LibVMI库,首先从物理内存中识别内核函数指针并建立一个指针列表,然后扫搜内核中引用列表函数指针的数据,执行较差验证,如果函数指针指向了内核代码之外的地址范围则认为其被恶意篡改.HookMap[54]利用后向数据切片技术定位可以被 Rootkit 用来挂钩的内存地址,并对其实施保护,Hypersafe[55]将钩子函数重定位到页对齐的内存空间,并利用基于硬件的页层级的保护机制对 hook 的访问进行控制.另外,也有部分研究工作对函数的返回地址等进行保护,以防止被劫持或误用,如 StackGuard[56],StackShield(http://www.angelfire.com/sk/stackshield/info.html)等.

2.3 基于VMI技术的恶意代码行为分析

在形式和功能上不断进化的恶意软件是信息化时代最大的安全威胁之一,常用的特征码扫描检测技术无法避免特征码滞后等缺陷[57].近来,由于 VMI 技术可以从虚拟机外详尽分析恶意代码,获取恶意软件的行为,如进程创建、系统调用及文件操作等[58],因此,越来越多的研究工作围绕基于 VMI 的恶意代码行为分析展开.

分析恶意软件攻击流程是近年的一个研究重点.文献[59]将 VMI 和 TCG 指令级的污点分析整合起来,并提供了事件驱动的编程接口,可以实现恶意软件行为的实时监控;文献[60]设计了基于 VMI 的混合型蜜罐系统进行恶意软件诱捕,并分析恶意代码的入侵和感染过程.

基于 VMI 技术分析恶意软件是否操纵敏感内核数据,是另外一个研究重点.PoKeR[61]是 kernel-Rootkit 分析器,具有挂钩行为分析、易受操纵内核目标识别、用户模式影响评估以及内核 Rootkit 代码抽取等 4 项功能;Rkprofiler[62]能够探测 TVM 中内核恶意代码的执行、捕获内核恶意代码的函数调用并构建函数调用图,同时利用 AMT(aggressive memory tagging)技术对内核恶意代码访问的动态实体进行跟踪,以发现 Rootkit 调用的内核函数及 Rootkit 访问的内核对象.

最后,防止恶意代码感知到检测软件的存在是基于 VMI 分析过程中的重要一环.由于硬件辅助虚拟化没有打乱原始的指令流程和异常处理句柄,因此其最适合做透明化监控[37].Ether[37]提出了一种对恶意代码实施透明跟踪的方法,当恶意代码通过查找 EFLAGS 寄存器中的单步调试位来判断其是否被调试时,Ether 通过设置 VMCS,截获对 EFLAGS 的访问,从而使系统运行态转到 VMX root 下进行处理,向恶意代码返回无调试存在(EFLAGS.TF=0)的结果.Holography[63]不依赖 TVM 操作系统内部的驱动来记录恶意代码的行为,而是依赖 Spy Satellite 监控 CPU 的指令、寄存器以及内存数据等硬件层次信息来截获恶意代码的行为,从而避免被恶意代码感知.

2.4 基于VMI技术的入侵检测

入侵检测系统通过检测主机或者网络信息,如内部状态、事件以及 I/O 活动等,来判断是否存在入侵行为.根据获取信息来源的不同,入侵检测系统可分为两类,以网络数据包为主要信息来源的基于网络的入侵检测系统以及以进程、网络连接、系统调用和日志等系统活动为主要信息来源的基于主机的入侵检测系统.基于网络的入侵检测系统具有较强的抗攻击的能力,但是对于主机内部的信息获取有限;另一方面,基于主机的入侵检测系统可以较好的获取主机内部的信息,但是其抗攻击的能力不强,容易被恶意软件屏蔽,使其难以发挥作用.然而,基于 VMI 的入侵检测系统可以将两者的优势整合在一起,使得新型入侵检测的信息来源更加丰富.

文献[4]最先利用上述思想设计了基于 VMI 的入侵检测系统 Livewire,Livewire 提供了用来监测 TVM 内

存、寄存器以及网络设备标志等状态信息的功能,可以监控特定范围的主机和网络状态变化,并可对异常事件予以干预.之后,也有部分工作基于此思路将其在检测及防范网络攻击等方面进行了研究.文献[64]设计了可以部署在云服务器集群内部的协同入侵检测系统,可以有效抵御“云滴冻结”等来自云环境内部的分布式拒绝服务攻击;基于 xen 实现的入侵检测系统 VNIDA[65]通过虚拟机管理器层的 event sensor 对 DomU 的状态进行观测,生成调用序列等信息以供检测使用;文献[66]则基于开源 VirtualBox 设计了基于 VMI 的入侵检测系统,实验结果显示,该入侵检测系统可以有效检测 TVM 中的 zero-day 恶意软件.

2.5 基于VMI技术的数字取证

静态的硬盘分析等计算机取证技术由于无法获取进程列表、开启的网络端口以及加载的内核模块等易消失数据,不能完备地获取所需信息.然而,VMI 和数字取证之间存在密切的联系,两者均是从相同的内存视图开始,不同的是 VMI 技术是在运行时检测安全事件的发生,而取证内存分析是在安全事件发生之后,获取物理内存镜像执行离线分析[67],因此,VMI 技术可以有效地弥补静态取证分析的不足.该领域也因此取得了快速发展,执行取证的数据获取、分析工具越来越丰富[68].比如,文献[69−71]尝试将基于 VMI 的内存分析技术整合到数字取证领域,取得了显著效果;基于硬件虚拟化技术的 HyperSleuth[38]实现了对遭受破坏的系统执行取证分析;基于 VMI 开发的监控语言 VMI-PL[72]可以灵活地对虚拟机的内存、数据流进行取证分析.

2.6 最新的应用进展

近年来,云计算迅速发展,其弹性可扩展的特性离不开系统虚拟技术的支撑,如亚马逊EC2、阿里云 ECS、IBM SoftLayer、Linode 及 Rackspace Cloud 等主流厂商均采用了 Xen 作为底层基础架构.云中虚拟机作为一种提供虚拟资源的主要方式,便成为了攻击者入侵云计算中心的首选目标.因此,对云中虚拟机的监控、取证等显得越来越重要.

首先从云服务提供商角度来分析现有基于 VMI 技术监控 TVM、数据保护等方面的工作.在监控 TVM 方面,文献[73]设计了一种基于 KVM 的云中入侵检测系统,其将每个 VM 视为一个独立的进程,利用 VMI 技术收集各 VM 系统调用信息,然后利用分类器分析系统调用信息集识别 TVM 的异常行为;CloudSec[74]则是一种基于 VMWare ESXi 的云中虚拟化安全解决方法,其利用 VMI 技术重构 TVM 的内存状态、监控内存操作事件,可有效检测云中虚拟机中是否存在 DKOM 等攻击.但是,将 VMI 技术应用于公有云环境时存在不能与已有监控系统良好兼容以及打破了多租户之间的隔离边界等缺陷.为此,文献[75]设计了弹性的具有加密功能的 CryptVMI,其向云租户提供其虚拟服务器的完整状态信息的同时,保证云服务提供商无法获悉.除此之外,VMI 技术在保护云中数据及提供高可靠服务等方面也发挥了重要作用.在这方面,TrustDraw[76]基于 VMI 技术的云中关键数据保护系统,文献[77]设计了融合 VMI 的强制访问控制方法;文献[78]则基于 VMI 设计了适应云计算环境的监测点/重启机制,缓解了该机制延迟高及监测点文件大小难以确定等问题,可以有效支撑云提供高可靠的服务.

在公有云环境下,由于云租户不具备访问管理主机(如 Dom0)的权限,因此,其无法直接使用 VMI 技术.为此,文献[79]设计了 CloudVMI,使得云租户可以借助转储内存信息服务,自主完成语义重构、安全分析,提升了云租户的自我安全能力.文献[80]针对云中安全事件取证的难题,设计了 LiveCloudInspector,通过基于 VMI 技术实现了远程内存快照、远程网络取证,使得云服务提供商可以将安全取证作为一项透明服务向租户提供.与前述工作不同,文献[81]通过赋予云租户更大、更灵活的权限,来应对该问题.其提出并设计了一种新型的特权模型,该模型一方面限制了 Dom0 的部分权限,另一方面赋予 DomU 灵活的控制权限,使得云租户可以按需配置 VMI 等服务.

在其他安全领域,越来越多的工作围绕特定的应用场景展开 VMI 技术的应用研究,并呈现出多种自省方法综合使用等特点,下面仅举两例说明之.信用卡、借记卡等数据的存储、处理及传输需遵循支付卡行业数据安全标准(PCI-DSS),以确保处理该类数据的基础架构免受攻击者的非法访问.由于该类处理架构通常是运行于多个虚拟机上的分布式处理系统,为此,文献[82]开发了基于 VMI 技术的监测系统,自动地检测支付卡数据处理等模块,确保贸易商遵循 PCI-DSS.也有部分工作围绕浏览器、邮件等位于用户空间的特定应用程序展开研究.文

献[83]通过 VMI 技术劫持特定应用程序的内存访问事件,形成包含三元组信息的窥探点,然后基于对窥探点的分析成功地实现了对浏览器等特定应用程序的监控.

3 展 望

研究人员已经从多个侧面对 VMI 技术进行了研究,并基于其设计了不同类型的安全工具,在内核完整性检测与保护、恶意代码行为分析、入侵检测以及数字取证等领域发挥了重要作用.然而,该技术在安全性、实用性及透明性等方面仍面临严峻的技术挑战.下面阐述我们对 VMI 技术的若干思考,以抛砖引玉.

VMI 技术有较强的工作前提,如其假设虚拟机管理器或目标虚拟机自身的完整性不受破坏,但是,实际情况却难以满足其需求,使得虚拟机自省技术难以持续发挥作用.为了提升 VMI 技术的安全性,我们认为后续工作需关注如下方面的研究.

(1) 提高虚拟机管理器的安全性

现有 VMI 的研究工作多数是在虚拟机管理器安全可靠的前提下开展的.但是,虚拟机管理器是管理资源分配等的系统软件,近年来,随着其功能的增多,代码量也随之加大,导致其安全漏洞频繁出现(http://www.cvedetails.com/vulnerability-list/vendor_id-6276/XEN.html).由于部分 VMI 技术的底层状态数据获取及传递等工作需借助虚拟机管理器才可以顺利完成,因此,应通过尽量简化虚拟机管理器的设计等方式增强虚拟机管理器的安全性,使得 VMI 技术可以稳定地发挥作用.

(2) 从语法和语义两方面提升基于软件结构知识的虚拟机自省方法的安全性

由于基于软件结构知识的虚拟机自省方法可以获取丰富的高层语义信息,因此,其具有较为广泛的使用范围,然而,该方法在重构语义时依赖的软件结构可被攻击者轻易改变,因此,提升该方法的鲁棒性显得尤为迫切.文献[20]从改变底层数据结构语法和语义的角度分析了针对该类自省方法的可能攻击途径:在语法方面,内核中存在某些未被使用的数据结构,因此,篡改此类数据结构可以在不影响系统运行的前提下,达到破坏语义恢复的目的;在语义方面,篡改底层数据结构语义,一方面使得语义对于操作系统透明,另一方面使得自省程序由于输入参数问题而失效.因此,提高基于软件结构知识的虚拟机自省法的健壮性是亟待解决的问题,需从语法与语义两个方面进行该方法的安全性提升方法研究.

虽然现有的语义重构方法已经能够有效的还原出大量安全所需的高层语义信息,但是,其在性能、可移植性及一致性等方面存在不同程度的不足,不能适应部分实际应用场景的需求.这方面在云计算环境中显得尤为突出,下面以云环境为例分析亟待深入研究的 VMI 技术实用性问题.

(1) 研究降低虚拟机自省技术资源消耗的方法

文献[13]指出部分 VMI 技术的开销一般在 9.3~500X 之间,如 VMST 在重构 ps 命令时会有 20X 的开销,Virtuoso 需要 6s 才可以遍历整个进程链表,由此可以看出,VMI 技术会有较大的时间消耗.由于普通计算机会有一定的空闲期,将该空闲期用于安全分析,用户是可以接受的;但是云中的计算能力需用来服务更多的用户,一方面,云服务提供商不愿因大量的资源消耗而导致收入减少,另一方面,云租户也不愿意为此付出更多的费用.因此,降低 VMI 技术的资源消耗、减少时间开销是将此技术应用于云计算需首要解决的问题.

(2) 增强虚拟机自省技术对不同客户操作系统的可移植性

云计算中心运行着数以万计的虚拟服务器,这在客观上要求基于 VMI 的安全工具具备同时监控多个虚拟服务器的功能.但是,云中运行于同一物理主机的虚拟机可能属于不同的租户,即便属于同一云租户,其所使用的操作系统也会不同.如果需同时运行多个安全工具才能满足上述要求,则会消耗云中的大量资源.因此,需研究同一 VMI 程序同时支撑多样操作系统,以增强 VMI 技术对不同客户操作系统的可移植性.

(3) 研究虚拟机状态一致性保障方法

VMI 技术的一个关键问题是虚拟机状态的一致性问题[8],若不采取恰当的一致性保障方法,则会导致待自省程序访问不存在或错误的底层状态数据.当前的方法通常是暂停-自省(pause-and-introspect,简称 PAI),但是该方法具有较强的入侵性,会因停机时间较长而影响某些服务的正常执行,影响了云服务的可用性.另一方面,PAI

方法只对外在不一致性(extrinsic inconsistency)问题有效,而对操作系统自身的数据结构处于不一致状态而引起的内在不一致性(intrinsic inconsistency)问题,PAI 方法也难以确保两者之间的一致性.因此,我们需研究新型一致性保障方法,在虚拟机状态一致性的前提下保障云服务的可用性不会因 VMI 技术的引入而受到影响.

较独立型自省方法,偏重状态可理解性的依赖型虚拟机自省方法在性能等方面具有一定优势,然而, VMI 技术对比传统的基于主机安全应用的优势恰恰来源于对受保护系统的透明性.若利用高效的依赖型自省方法获取高层语义信息,则势必会影响 VMI 技术的透明性优势,我们认为,后续工作应更加注重 VMI 技术透明性方面的研究,使 VMI 技术在透明性方面的优势得以充分体现.

(1) 研究免语义重构的虚拟机自省安全方法

应用 VMI 技术进行安全研究的习惯性思路是将二进制信息重构为高层语义信息,然后实施监控.但是,某些安全应用的目的只是简单的区分虚拟机某部分状态是否安全,因此可以考虑不经语义重构而直接对特定的二进制状态分析的安全方法,如此便可在保证性能损耗较低的前提下,保障了 VMI 技术的高透明性.虽然,已有小部分工作进行了尝试性的研究,但下面两个问题仍没有得到很好的解决:① 如何在二进制状态数据中自动发现可以用于分类的良好特征;② 如何设计适应状态数据分类的算法.

(2) 注重研究基于硬件架构知识的虚拟机自省及其应用方法

由表 2 可知,基于硬件知识的虚拟机自省方法除了在性能及安全性等方面有一定优势外,其透明性的优势也较为突出,基本不需要目标虚拟机的支持.但是,当前基于该方法的应用研究还比较缺乏,部分硬件架构知识的价值还未得到充分挖掘,因此,本文认为应重视硬件架构知识在语义重构方面不可替代的作用,以提高虚拟机自省方法的透明性.

4 结束语

近年来,随着云计算的兴起,基于虚拟化的安全研究引起了越来越多的关注.本文首先梳理了语义重构的四种方式,并分别对其面临的关键问题及相关工作进行了分析;然后分别从内核完整性检测、内核完整性保护、恶意代码分析、入侵检测、数字取证以及最近的应用进展等方面介绍了基于 VMI 的安全研究工作.最后,在总结上述工作的基础上,本文对未来虚拟机自行技术进行了展望,以期引起大家的关注.

致谢 在此,我们向对本文的工作给予支持和建议的同行,尤其是向信息内容安全国家工程实验室的老师和同学表示衷心感谢.

References:

[1] Jin H. Computing System Virtualization: Principles and Applications. Beijing: Tsinghua University Press, 2008. 3−51 (in Chinese).

[2] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt L, Warfield A. Xen and the art of virtualization. In: Scott ML, ed. Proc. of the 19th ACM Symp. on Operating Systems Principles. New York: ACM Press, 2003. 164−177. [doi: 10.1145/945445.945462]

[3] Kivity A, Kamay Y, Laor D, Lublin U, Anthony L. KVM: The Linux virtual machine monitor. In: Proc. of the Linux Symp. New York: ACM Press, 2007. 225−230.

[4] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection. In: Neuman C, ed. Proc. of the Network and Distributed Systems Security Symp. Washington: Internet Society, 2003. 191−206.

[5] Pfoh J, Schneider C, Eckert C. A formal model for virtual machine introspection. In: Nieh J, Stavrou A, eds. Proc. of the 1st ACM Workshop on Virtual Machine Security. New York: ACM Press, 2009. 1−10. [doi: 10.1145/1655148.1655150]

[6] Payne BD. Simplifying virtual machine introspection using Libvmi. Vol1.1, Sandia Report, Sandia National Laboratories, 2012. 1−20.

[7] Xiang GF, Jin H, Zeng DQ. Virtualization-Based security monitoring. Ruan Jian Xue Bao/Journal of Software, 2012,23(8): 2173−2187 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4219.htm [doi: 10.3724/SP.J.1001.2012.04219]

[8] Suneja S, Isci C, de Lara E, Bala V. Exploring VM introspection: Techniques and trade-offs. In: Gavrilovska A, Brown AD, Steensgaard B, eds. Proc. of the 11th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. New York: ACM Press, 2015. 133−146. [doi: 10.1145/2731186.2731196]

[9] Payne BD, Carbone M, Sharif M, Lee W. Lares: An architecture for secure active monitoring using virtualization. In: ISO Securityprivacy, ed. Proc. of the 29th IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society, 2008. 233−247. [doi: 10.1109/SP.2008.24]

[10] Sharif M, Lee W, Cui W, Lanzi A. Secure in-VM monitoring using hardware virtualization. In: Proc. of the 16th ACM Conf. on Computer and Communications Security. New York: ACM, 2009. 477−487. [doi: 10.1145/1653662.1653720]

[11] Asrigo K, Litty L, Lie D. Using VMM-based sensors to monitor honeypots. In: Boehm H, ed. Proc. of the 2nd ACM Int'l Conf. on Virtual Execution Environments. New York: ACM Press, 2006. 13−23. [doi: 10.1145/1134760.1134765]

[12] Vogl S, Kilic F, Schneider C, Eckert C. X-TIER: Kernel module injection. In: Lopez J, Huang XY, Sandhu R, eds. Proc. of the Network and System Security. LNCS 7873, 2013. 192−205. [doi: 10.1007/978-3-642-38631-2_15]

[13] Wu R, Chen P, Liu P, Mao B. System call redirection: A practical approach to meeting real-world virtual machine introspection needs. In: Blough D, ed. Proc. of the 44th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN). Washington: IEEE Computer Society, 2014. 574−585. [doi: 10.1109/DSN.2014.59]

[14] Carbone M, Conover M, Montague B, Lee W. Secure and robust monitoring of virtual machines through guest-assisted introspection. LNCS 7462, 2012. 22−41. [doi: 10.1007/978- 3-642-33338-5_2]

[15] Gu ZS, Deng Z, Xu DY, Jiang XX. Process implanting: A new active introspection framework for virtualization. In: Proc. of the 30th IEEE Symp. on Reliable Distributed Systems (SRDS). Washington: IEEE Computer Society, 2011. 147−156. [doi: 10.1109/SRDS.2011.26]

[16] Srinivasan D, Wang Z, Jiang X, Xu DY. Process out-grafting: An efficient "out-of-VM" approach for fine-grained process execution monitoring. In: Proc. of the 18th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2011. 363−374. [doi: 10.1145/2046707.2046751]

[17] Jain B, Baig MB, Zhang DL, Porter DE, Sion R. SOK: Introspections on trust and the semantic gap. In: Proc. of the 2014 IEEE Symp. on Security and Privacy (SP). Washington: IEEE Computer Society, 2014. 605−620. [doi: 10.1109/SP.2014.45]

[18] Payne BD, Martim DP, de Carbone A, Lee W. Secure and flexible monitoring of virtual machines. In: Proc. of the Computer Security Applications Conf. Washington: IEEE Computer Society, 2007. 385−397. [doi: 10.1109/ACSAC.2007.10]

[19] Kourai K, Nakamura K. Efficient VM introspection in KVM and Performance comparison with Xen. In: Proc. of the IEEE 20th Pacific Rim Int'l Symp. on Dependable Computing (PRDC). Washington: IEEE Computer Society, 2014. 192−202. [doi: 10.1109/PRDC.2014.33]

[20] Bahram S, Jiang X, Wang Z, Grace M, Li J, Xu D. Dksm: Subverting virtual machine introspection for fun and profit. In: Proc. of the 29th IEEE Symp. on Reliable Distributed Systems. Washington: IEEE Computer Society, 2010. 82−91. [doi: 10.1109/SRDS.2010.39]

[21] Carbone M, Cui WD, Lu L, Lee W, Peinado M, Jiang XX. Mapping kernel objects to enable systematic integrity checking. In: Proc. of the 16th ACM Conf. on Computer and Communications Security (CCS 2009). New York: ACM Press, 2009. 555−565. [doi: 10.1145/1653662.1653729]

[22] Andersen LO. Program analysis and specialization for the C programming language [Ph.D. Thesis]. University of Copenhagen, 1994.

[23] Schnerder C, Pfoh J, Echert C. Bridging the semantic gap through static code analysis. In: Proc. of the 2012 European Workshop on System Security (EuroSec 2012). New York: ACM Press, 2012. 1−6.

[24] Cui WD, Peinado M, Xu ZL, Chan E. Tracking rootkit footprints with a practical memory analysis system. In: Proc. of the USENIX Security Symp. (USENIX). New York: ACM Press, 2012. 601−615.

[25] Hizver J, Chiueh T. Real-Time deep virtual machine introspection and its applications. In: Proc. of the 10th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. New York: ACM Press, 2014. 3−14. [doi: 10.1145/2576195.2576196]

[26] Roberts A, McClatchey R, Liaquat S, Edwards N, Wray M. POSTER: Introducing pathogen: A real-time virtual machine introspection framework. In: Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. New York: ACM Press, 2013. 1429−1432. [doi: 10.1145/2508859.2512518]

[27] Dolan-Gavitt B, Leek T, Zhivich M, Giffin J, Lee W. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In: Proc. of the 32nd IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society, 2011. 297−312. [doi: 10.1109/SP.2011.11]

[28] Fu FC, Lin ZQ. Space traveling across VM: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection. In: Proc. of the IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society, 2012. 586−600. [doi: 10.1109/SP.2012.40]

[29] Saberi A, Fu Y, Lin Z. Hybrid-Bridge: Efficiently bridging the semantic gap in virtual machine introspection via decoupled execution and training memorization. In: Proc. of the 21st Annual Network and Distributed System Security Symp. Washington: Internet Society, 2014. 1−15.

[30] Walters A, Petroni NL. Volatools: Integrating volatile memory forensics into the digital investigation process. 2007. http://www.blackhat.com/presentations/bh-dc-07/Walters/Paper/bh-dc-07-Walters-WP.pdf

[31] Dolan-Gavitt B, Srivastava A, Traynor P, Giffin J. Robust signatures for kernel data structures. In: Proc. of the 16th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2009. 566−577. [doi: 10.1145/1653662.1653730]

[32] Lin ZQ, Rhee J, Zhang XY, Xu DY, Jiang XX. Siggraph: Brute force scanning of kernel data structure instances using graph-based signatures. In: Proc. of the 18th Annual Network and Distributed System Security Symp. Washington: Internet Society, 2011. 1−18.

[33] Jones ST, Arpaci-Susseau AC, Arpaci-Dusseau RH. Antfarm: Tracking processes in a virtual machine environment. In: Proc. of the USENIX Annual Technical Conf. New York: ACM Press, 2006. 1−14.

[34] Rhee J, Riley R, Xu DY, Jiang XX. Kernel malware analysis with un-tampered and temporal views of dynamic kernel memory. In: Proc. of the 13th Int'l Symp. of Recent Advances in Intrusion Detection. Berlin: Springer-Verlag, 2010. 178−197. [doi: 10.1007/978-3-642-15512-3_10]

[35] Pfoh J, Schneider C, Eckert C. Exploiting the x86 architecture to derive virtual machine state information. In: Proc of the 4th Int'l Conf. on Emerging Security Information, Systems and Technologies. Washington: IEEE Computer Society, 2010. 166−175. [doi: 10.1109/SECURWARE.2010.35]

[36] Pfoh J, Schneider C, Eckert C. Nitro: Hardware-Based system call tracing for virtual machines. In: Tetsu I, Nishigaki M, eds. Proc. of the 6th Int'l Conf. on Advances in Information and Computer Security. Washington: IEEE Computer Society, 2011. 96−112. [doi: 10.1007/978-3-642-25141-2_7]

[37] Dinaburg A, Royal P, Sharif M, Lee W. Ether: Malware analysis via hardware virtualization extensions. In: Ning P, ed. Proc. of the 15th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2008. 51−62. [doi: 10.1145/1455770.1455779]

[38] Martignoni L, Fattori A, Paleari R, Cavallaro L. Live and trustworthy forensic analysis of commodity production systems. In: Somesh J, Robin S, Christian K, eds. Proc. of Recent Advances in Intrusion Detection. Berlin: Springer-Verlag, 2010. 297−316. [doi: 10.1007/978-3-642-15512-3_16]

[39] Xiong HQ, Liu ZY. The architectural based interception and identification of system call instruction within VMM. In: Datta A, ed. Proc. of the the 1st Int'l Workshop on Cloud Computing and Information Security. Paris: Atlantis Press, 2013. 73−76.

[40] Leff A, Rayfield JT. Web-Application development using the model-view-controller design pattern. In: Titsworth FM, ed. Proc. of the 5th IEEE Enterprise Distributed Object Computing Conf. Washington: IEEE Computer Society, 2001. 118−124. [doi: 10.1109/EDOC.2001.950428]

[41] Petroni NL, Hicks JRM. Automated detection of persistent kernel control-flow attacks. In: Ning P, ed. Proc. of the 14th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2007. 103−115. [doi: 10.1145/1315245.1315260]

[42] Baliga A, Ganapathy V, Iftode L. Automatic inference and enforcement of kernel data structure invariants. In: Proc. of the 24th Annual Computer Security Applications Conf. New York: ACM Press, 2008. 77−86. [doi: 10.1109/ACSAC.2008.29]

[43] Hofmann O, Dunn A, Kim S, Roy I, Witchel E. Ensuring operating system kernel integrity with OSCK. In: Gupta R, ed. Proc. of the 16th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2011. 279−290. [doi: 10.1145/1950365.1950398]

[44] Azab AM, Ning P, Sezer EC, Zhang XL. HIMA: A hypervisor-based integrity measurement agent. In: Gates C, ed. Proc. of the 25th Annual Computer Security Applications Conf. New York: ACM Press, 2009. 461−470. [doi: 10.1109/ACSAC.2009.50]

[45] Jiang X, Wang X, Xu I. Stealthy malware detection through VMM-based "out-of-the-box" semantic view reconstruction. In: Ning P, ed. Proc. of the 14th ACM Conf. on Computer and Comm unications Security. New York, 2007. 128−138. [doi: 10.1145/ 1315245.1315262]

[46] Wang L, Gao HJ, Liu W, Peng Y. Detecting and managing hidden process via hypervisor. Journal of Computer Research and Development, 2011,48(8):1534−1541 (in Chinese with English abstract).

[47] Jones ST, Arpaci-Dusseau AC, Arpaci-Dusseau RH. VMM-Based hidden process detection and identification using Lycosid. In: Gavrilovska A, ed. Proc. of the 4th Int'l Conf. on Virtual Execution Environments. New York: ACM Press, 2008. 91−100. [doi: 10.1145/1346256.1346269]

[48] Srivastava A, Giffin J. Efficient protection of kernel data structures via object partitioning. In: Zakon RH, ed. Proc. of the Annual Computer Security Applications Conf. New York: ACM Press, 2012. 429−438. [doi: 10.1145/2420950.2421012]

[49] Liu YT, Xia YB, Guan HB, Zang BY, Chen HB. Concurrent and consistent virtual machine introspection with hardware transactional memoty. In: Jerger NE, ed. Proc. of the 20th IEEE Int'l Symp. On High Performance Computer Architecture. Washington: IEEE Computer Society, 2014. 416−427. [doi: 10.1109/HPCA.2014.6835951]

[50] Seshadri A, Luk M, Qu N, Perrig A. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In: Bressoud TC, ed. Proc. of the ACM Symp. on Operating System Principles. New York: ACM Press, 2007. 335−350. [doi: 10.1145/1294261.1294294]

[51] Riley R, Jiang X, Xu D. Guest-Transparent prevention of kernel rootkits with VMM-based memory shadowing. In: Lippmann R, ed. Proc. of the 11th Recent Advances in Intrusion Detection. Berlin: Springer-Verlag, 2008. 1−20. [doi: 10.1007/978-3-540- 87403-4_1]

[52] Litty L, Lagar-Cavilla HA, Lie D. Hypervisor support for identifying covertly executing binaries. In: Van Oorschot P, ed. Proc. of the 17th Conf. on Security Symp. New York: ACM Press, 2008. 243−258.

[53] Ahmed I, Richard III GG, Zoranic A, Roussev V. Integrity checking of function pointers in kernel pools via virtual machine introspection. In: Thuraisingham B, ed. Proc. of the 16th Information Security Conf. Berlin: Springer-Verlag, 2013. 1−16. [doi: 10.1007/978-3-319-27659-5_1]

[54] Wang Z, Jiang X, Cui W, Wang XY. Countering persistent kernel rootkits through systematic hook discovery. In: Lippman R, ed. Proc. of the Int'l Symp. on Recent Advances in Intrusion Detection. Berlin: Springer-Verlag, 2008. 21−38. [doi: 10.1007/978-3-540-87403-4_2]

[55] Wang Z, Jiang X. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In: Lindqvist U, ed. Proc. of the 31st IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society, 2010. 380−395. [doi: 10.1109/ SP. 2010.30]

[56] Cowan C, Pu C, Maier D, Hintony H, Walpole J, Bakke P, Beattie S, Grier A, Wagle P, Zhang Q. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In: Rubin A, ed. Proc. of the 7th USENIX Security Symp. New York: ACM Press, 1998. 63−78.

[57] Zolkipli MF, Jantan A. Malware behavior analysis: Learning and understanding current malware threats. In: Setar A, ed. Proc. of the 2nd Int'l Conf. on Network Applications, Protocols and Services. Washington: IEEE Computer Society, 2010. 218−221. [doi: 10.1109/NETAPPS.2010.46]

[58] Lengyel TK, Maresca S, Payne BD, George D, Vogl S, Kiayias A. Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system. In: Kevin B, ed. Proc. of the 30th Annual Computer Security Applications Conf. New York: ACM Press, 2014. 386−395. [doi: 10.1145/2664243.2664252]

[59] Henderson A, Prakash, Yan Lk, Hu XC, Wang X, Zhou RD, Yin H. Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform. In: Dwyer M, ed. Proc. of the 2014 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2014. 248−258. [doi: 10.1145/2610384.2610407]

[60] Lengyel TK, Neumann J, Maresca S, Payne BD, Kiayias A. Virtual machine introspection in a hybrid honeypot architecture. In: Sean P, ed. Proc. of the 5th USENIX Conf. on Cyber Security Experimentation and Test. New York: ACM Press, 2012. 1−8.

[61] Riley R, Jiang X, Xu D. Multi-Asspect profiling of kernel rootkit behavior. In: Wilkes J, ed. Proc. of the 4th ACM SIGOPS EuroSys Conf. New York: ACM Press, 2009. 47−60. [doi: 10.1145/1519065.1519072]

[62] Xuan C, Copeland J, Beyah R. Toward revealing kernel malware behavior in virtual execution environments. In: Engin K, ed. Proc. of the 12th Int'l Symp. on Recent Advances in Intrusion Detection. New York: ACM Press, 2009. 304−325. [doi: 10.1007/ 978-3-642-04342-0_16]

[63] Dai SY, Fyodor Y, Wu JS, Lin CH, Huang Y, Kuo SY. Holography: A hardware virtualization tool for malware analysis. In: Hua T, Xu SY, Gao JH, eds. Proc. of the 15th IEEE Pacific Rim Int'l Symp. on Dependable Computing, Washington: IEEE Computer Society, 2009. 263−268. [doi: 10.1109/PRDC.2009.48]

[64] Wang YC, Ma JF,Lu D, Zhang LM, Meng XJ. Game optimization for Internet DDoS attack detection in cloud computing. Journal of Computer Research and Development, 2015,1873−1882 (in Chinese with English abstract).

[65] Zhang X, Li Q, Qing S, Zhang H. VNIDA: Building an IDS architecture using VMM-based non-intrusive approach. In: Macskassy S, ed. Proc. of the 14th Int'l Workshop on Knowledge Discovery and Data Mining. New York: ACM Press, 2008. 594−600. [doi: 10.1109/WKDD.2008.135]

[66] Azmandian F, Moffie M, Alshawabkeh M, Jennifer Dy, Aslam J, Kaeli D. Virtual machine monitor-based lightweight intrusion detection. ACM SIGOPS Operating Systems Review, 2011,45(2):38−53. [doi: 10.1145/2007183.2007189]

[67] Nance K, Hay B, Bishop M. Investigating the implications of virtual machine introspection for digital forensics. In: Josang A, ed. Proc. of the Int'l Conf. on Availability, Reliability and Security. New York: ACM Press, 2009. 1024−1029. [doi: 10.1109/ ARES. 2009.173]

[68] Garcia GL. Forensic physical memory analysis: An overview of tools and techniques. In: TKK T-110.5290 Seminar on Network Security. 2007. 305−320.

[69] Petroni NL, Jr. Walters A, Fraser T, Arbaugh WA. FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory. Digital Investigation Journal, 2006,3(4):197−210. [doi: 10.1016/j.diin.2006.10.001]

[70] Walters AA, Petroni NL. Volatools: Integrating volatile memory into the digital investigation process. 2007.

[71] Hay B, Nance K. Forensics examination of volatile system data using virtual introspection. ACM SIGOPS Operating System Review, 2008,42:74−82. [doi: 10.1145/1368506.1368517]

[72] Westphal F, Axelsson S, Neuhaus C, Polze A. VMI-PL: A monitoring language for virtual platforms using virtual machine introspection. Digital Investagation, 2014,11:S85−S94. [doi: 10.1016/j.diin.2014.05.016]

[73] Alarifi SS, Wolthusen SD. Detecting anomalies in IaaS environments through virtual machine host system call analysis. In: Warwick K, ed. Proc. of the Int'l Conf. for Internet Technology and Secured Transactions. Washington: IEEE Computer Society, 2012. 211−218.

[74] Ibrahim AS, Hamlyn-Harris J, Grundy J, Almorsy M. CloudSec: A security monitoring appliance for virtual machines in the IaaS cloud model. In: Qiu MK, ed. Proc. of the 5th Int'l Conf. on Network and System Security. Washington: IEEE Computer Society, 2011. 113−120. [doi: 10.1109/ICNSS.2011.6059967]

[75] Yao F, Sprabery R, Campbell RH. CryptVMI: A flexible and encrypted virtual machine introspection system in the cloud. In: Robert D, ed. Proc. of the 2nd Int'l Workshop on Security in Cloud Computing. New York: ACM Press, 2014. 11−18. [doi: 10.1145/2600075.2600078]

[76] Biedermann S, Katzenbeisser S. POSTER: Event-Based isolation of critical data in the cloud. In: Sadeghi AR, ed. Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. New York: ACM Press, 2013. 1383−1386. [doi: 10.1145/2508859.2512506]

[77] Win TY, Tianfield H, Mair Q. Virtualization security combining mandatory access control and virtual machine introspection. In: Bahsoon R, ed. Proc. of the 2014 IEEE/ACM 7th Int'l Conf. on Utility and Cloud Computing. Washington: IEEE Computer Society, 2014. 1004−1009. [doi: 10.1109/UCC.2014.165]

[78] Ferrol A, Scott HF, Stephen L, Naughton T. Efficient checkpointing of virtual machines using virtual machine introspection. In: Kondo D, ed. Proc. of the 14th IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing. Washington: IEEE Computer Society, 2014. 414−423. [doi: 10.1109/CCGrid.2014.72]

[79] Baek HW, Srivastava A, Van der Merwe J. CloudVMI: Virtual machine introspection as a cloud service. In: Proc. of the 2014 IEEE Int'l Conf. on Cloud Engineering. Washington: IEEE Computer Society, 2014. 153−158. [doi: 10.1109/IC2E.2014.82]

[80] Zach J, Reiser HP. Live cloud inspector: Towards integrated IaaS forensics in the cloud. In: Distributed Applications and Interoperable Systems. Springer-Verlag, 2015. 207−220.

[81] Butt S, Lagar-Cavilla HA, Srivastava A, Ganapathy V. Self-Service cloud computing. In: Yu T, ed. Proc. of the 2012 ACM Conf. on Computer and Communications Security. New York: ACM Press, 2012. 253−264. [doi: 10.1145/2382196.2382226]

[82] Hizver J, Chiueh T. Tracking payment card data flow using virtual machine state introspection. In: Zakon RH, ed. Proc. of the 27th Annual Computer Security Applications Conf. (ACSAC 2011). New York: ACM Press, 2011. 277−285. [doi: 10.1145/2076732.2076771]

[83] Dolan-Gavitt B, Leek T, Hodosh J, Lee W. Tappan zee (north) bridge: Mining memory accesses for introspection. In: Sadeghi AR, ed. Proc. of the 2013 ACM SIGSAC Conf. on Computer & Communications Security. New York: ACM Press, 2013. 839−850. [doi: 10.1145/2508859.2516697]

附中文参考文献:

[1] 金海.计算系统虚拟化:原理与应用.北京:清华大学出版社,2008.

[7] 项国富,金海,邹德清,陈学广.基于虚拟化的安全监控.软件学报,2012,23(8):2173−2187. http://www.jos.org.cn/1000-9825/4219.htm [doi: 10.3724/SP.J.1001.2012.04219]

[47] 王丽娜,高汉军,刘炜,彭洋.利用虚拟机监视器检测及管理隐藏进程.计算机研究与发展,2011,48(8):1534−1541.

[64] 王一川,马建峰,卢笛,张留美,孟宪佳.面向云环境内部 DDoS 攻击检测的博弈论优化.计算机研究与发展,2015,52(8):1873−1882.

李保珲(1986－),男,山东东阿人,博士生,主要研究领域为云计算及其安全,网络安全.

徐克付(1977－),男,博士,副研究员,主要研究领域为网络安全,云计算及其安全.

张鹏(1984－),男,博士,副研究员,主要研究领域为分布式系统,数据挖掘.

郭莉(1969－),女,正研级高级工程师,博士生导师,主要研究领域为信息安全.

胡玥(1963－),女,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为文本分类,社交网络,网络安全.

方滨兴(1960－),男,博士,博士生导师,中国工程院院士,主要研究领域为网络安全,信息内容安全,分布式计算.