虚拟化软件栈安全研究

朱 民 涂碧波 孟 丹
(中国科学院信息工程研究所 北京 100093)
(中国科学院大学 北京 100049)

摘 要 随着云计算的兴起,虚拟化技术在 IT 产业中得到了广泛应用.政府、企业和个人已将自身的大量业务及其敏感数据转移到了云端的虚拟机中.在虚拟化软件栈中,虚拟机监控器具有最高权限和较小的可信计算基,故而能为虚拟化系统提供安全监控和保护.但同时也引入了新的软件层,增加了脆弱性,增大了攻击面.另外,多租户模式以及软硬件平台资源共享,更加剧了新软件栈的安全威胁.因此,虚拟机和虚拟机监控器的安全和隐私备受学术界和工业界关注.该文对虚拟化软件栈不同软件层的安全威胁、攻击方式和威胁机理进行了分析,并针对这些安全威胁,以可信基为视角,从基于虚拟机监控器、基于微虚拟机监控器、基于嵌套虚拟化和基于安全硬件等类别分析比较了国内外相关安全方案和技术,并指出了当前仍然存在的安全问题.最后对未来的研究方向进行了探讨和分析,并从软件和硬件两个层面给出了虚拟化软件栈的安全增强方案.

关键词 虚拟化软件栈;虚拟机安全;云安全;内存安全;计算机安全体系结构
中图法分类号 TP309 **DOI 号** 10.11897/SP.J.1016.2017.00481

The Security Research of Virtualization Software Stack

ZHU Min TU Bi-Bo MENG Dan
(*Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093)
(*University of Chinese Academy of Sciences, Beijing* 100049)

Abstract With the rise of cloud computing, virtualization has been extensively employed in the IT industry. Governments, enterprises, and private citizens have moved their business and confidential data to the virtual machines hosted in the remote cloud. In the virtualization software stack, hypervisor has the highest privilege and smaller trusted computing base, hence it can provide protection and security monitoring for a virtualization system. But virtualization also introduces a new software layer which increases the attack surfaces and vulnerabilities of the whole system. Besides, the multi-tenant model and sharing of software and hardware resources exacerbate the security threats of this new virtualization software stack. As a result, the security attacks and defenses of virtual machines and hypervisor have come under a great deal of concern in computer academia and industry. This paper analyzes the security risks, attacking mode and threat mechanism at different layers of the virtualization software stack. For these security threats, this paper analyzes and compares the state of the art of existing research findings and technologies both at home and abroad from the perspective of trusted computing base including the hypervisor-based, micro-hypervisor-based, nested virtualization-based and security hardware-based, etc. And based on the classifying of the above solutions, this paper also points out the remaining existing security problems. In the end, this paper discusses and analyzes the direction

收稿日期:2015-11-20;在线出版日期:2016-06-22.本课题得到国家重点研发计划"网络空间安全"重大专项课题(2016YFB0801002)资助.**朱 民**,男,1988 年生,博士研究生,主要研究方向为信息隐私安全、云计算安全.E-mail:zhumin@iie.ac.cn.**涂碧波**,男,1977 年生,博士,研究员,主要研究领域为网络与系统安全、操作系统.**孟 丹**,男,1965 年生,博士,研究员,主要研究领域为计算机体系结构、网络与系统安全.

of the development of future research in virtualization security and gives the enhanced security solution of the virtualization software stack in the aspects of software and hardware.

Keywords virtualization software stack; virtual machine security; cloud security; memory security; computer security architecture

1 引 言

近年来,云计算(cloud computing)得到了迅速发展,具有广阔的应用前景.目前,谷歌、IBM、微软、亚马逊、惠普、思杰、百度、腾讯和阿里巴巴等知名 IT 企业都在全力推广自身的云服务.云计算以低成本的资源租赁方式提供高可靠、高可用、规模可伸缩的个性化服务,得到了众多公司和个人的青睐[1-2].云计算与传统 IT 环境的最大区别在于其虚拟的计算环境.虚拟化是支撑云计算的技术基石,云计算中所有应用的部署环境和资源利用都依赖于虚拟平台的管理、扩展和迁移等,各种敏感操作都经由虚拟化层模拟完成.虚拟化技术通过对底层硬件(如 CPU、内存、网络)的虚拟,支持在单一服务器上并行运行多个虚拟机 VM(Virtual Machine),这提高了服务器的利用率,并为应用提供了灵活可变、动态可扩展的平台服务.

然而,云计算促进了虚拟化在 IT 产业中的应用,但同时也使得虚拟化软件栈成为重点攻击目标.首先,虚拟化扩增了传统服务器的软件栈.众所周知,软件栈越大、越复杂,攻击面和脆弱性就越多,安全性则更难以保障.根据 NVD 统计,2015 年 Xen 曝出 21 个安全漏洞,而近两年 KVM 拥有 24 个以上的安全漏洞①.如果具有最高特权的虚拟机监控器(hypervisor)被控制,其上运行的所有虚拟机的安全将难以保证.其次,在云计算平台中,物理资源通过虚拟化技术供多个租户共享,这导致攻击者和其他用户可能共处同一台物理机,而虚拟化技术提供的隔离性并不强,这降低了攻击难度②.最后,排除软件漏洞,攻击者利用侧信道攻击(side-channel attack)也可窃取其它虚拟机的敏感数据[3].不断发生的攻击事件充分说明了虚拟化软件栈安全的重要性.例如,2007 年云服务提供商 Salesforce.com 遭受攻击,导致大量用户的敏感数据泄露丢失;2009 年 Google 发生大批用户文件泄漏事件;2010 年和 2011 年微软和 Google 的邮箱服务分别出现了数据丢失事件;2014 年 iCloud 遭受攻击,导致大量好莱坞明星的私照被泄露③.

除了黑客的主动攻击之外,云计算还存在信任缺失问题[4].在云计算中,用户把计算等资源外包给云提供商,失去了对计算和数据的完全控制.用户完全不能确定自己的数据是否受到保护、应用程序是否被正确执行.为保护用户隐私,云提供商的内部人员应具有最小权限,但事实并非如此.内部人员的失职、好奇等都会导致数据破坏、隐私泄漏.例如 2010 年 Google 的两名员工长期窥探用户的 Google Voice 和 Gtalk 信息④.此外,内部人员可以绕过部分安全机制,更易实施攻击.如今,在用户不能对云平台执行环境进行管控和安全确认的情况下,云提供商未能提供有效的证据证明用户数据的安全.

以上论述很好地解释了为何很多公司的 CIO(Chief Information Officer)迟迟不将其核心业务部署到云平台.为消除用户的顾虑,让企业和个人大规模地使用云平台和服务,数据安全是云提供商目前亟待解决的问题.为此,产业界和学术界都在不断地提出相应的安全机制和解决方案.当前虚拟化的研究主要集中在对 hypervisor 的保护、对虚拟机的隔离以及对 VM 的内部系统、应用的保护,甚至将虚拟化从可信计算基(Trusted Computing Base,TCB)中剔除,以此来增强虚拟化软件栈的安全.

本文目的在于对虚拟化软件栈存在的安全威胁、研究现状及解决方案进行一个全面的论述,并通过分析、比较找出未来虚拟化软件栈安全的研究方向.本文首先在第 2 节对云计算框架和虚拟化技术进行简要的介绍.然后,在第 3 节对虚拟化软件栈的安全威胁进行分析和分类.随之,第 4 节结合第 3 节中的安全威胁,从不同层次阐述近年来学术界为增强虚拟化软件栈的安全在各方面所做的尝试.最后,

① National Vulnerability Database. https://web.nvd.nist.gov/2016,6,9

② Common Virtualization Vulnerabilities and How to Mitigate Risks. https://pentestlab.wordpress.com/2013/02/25/common-virtualization-vulnerabilities-and-how-to-mitigate-risks/2013,2,25

③ Lessons from the Apple iCloud Data Leak (Acellion). http://www.accellion.com/blog/lessons-apple-icloud-data-leak, 2014,9,23

④ Google fired employees for breaching user privacy(Techspot). http://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html, 2010,9,15

在第 5 节对未来虚拟化软件栈的安全目标和发展方向进行展望，并在第 6 节对全文进行总结.

2　云计算与虚拟化

云计算是继分布式计算、网格计算和对等计算之后的一种新型计算模式[5]. 狭义的云计算是指通过虚拟化技术和分布式计算构建的数据中心. 广义的云计算指厂商利用虚拟化技术构建虚拟集群，动态地组织异构计算资源，隔离具体硬件体系结构和多样化软件平台差异，灵活构建满足不同应用需求的计算环境，提高计算资源的使用效率，实现弹性计算. 用户只需利用终端通过网络即可访问云计算提供的在线软件服务、计算分析、硬件租用等不同类型的服务，从而实现资源管理、维护与使用的解耦. 从横向来看，云计算框架如图 1 所示.

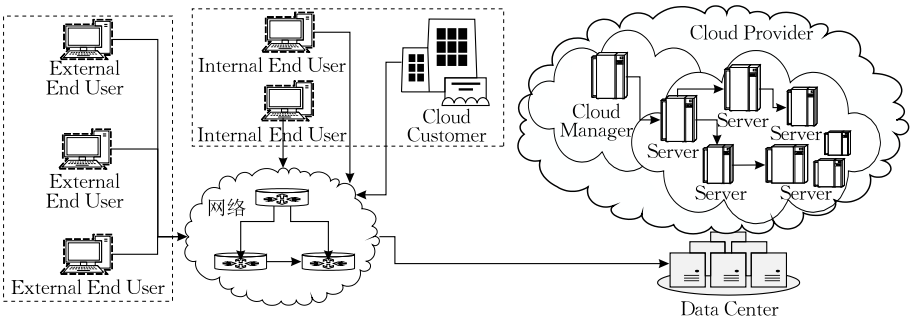

图 1　云计算概要图

从图 1 中可知，典型的云计算主要包括三部分：云提供商、云客户和终端用户. 云客户从云提供商租用资源，然后部署自己的服务，供终端用户使用. 终端用户利用终端通过网络就可访问相关服务. 云客户不需要购买软、硬件资源，只需根据业务需求动态地从云提供商租赁即可. 云提供商以虚拟机的方式向客户提供计算资源. IT 领域按需服务、弹性扩展的理念在云计算中得到了真正的体现.

未来云计算将形成一个以云基础设施为核心、涵盖云基础软件、平台服务与云应用服务等多层次的 IT 模式. 从图 1 可知，云基础设施的物理实体是数据中心. 数据中心内的物理硬件通过虚拟化的抽象、简化，形成了弹性虚拟计算资源池. 虚拟化角度下的云计算架构如图 2 所示. 从上层看，云计算是一种资源整合和利用的模型，而下层的技术支撑则是虚拟化技术. 利用虚拟化技术，数据中心形成了一个服务导向的、可伸缩的 IT 基础架构.

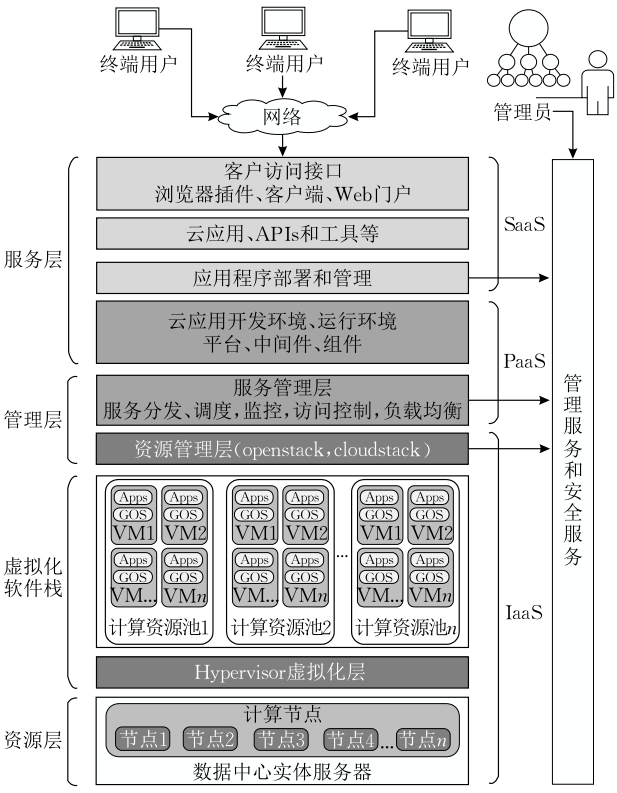

图 2　虚拟化角度下的云计算

如图 2 所示，虚拟化在单一物理服务器上构建了一个包含虚拟机监控器、客户操作系统(Guest Operating System，GOS)和应用(Applications，Apps)的虚拟化软件栈，而这可以看作是云计算的雏形和缩影.

Hypervisor 作为虚拟化的核心部件，是一种位于虚拟机和底层硬件之间的特殊操作系统，目的是提高资源利用率，降低能耗，以及提高资源配置的灵活性和弹性服务. Hypervisor 主要负责虚拟机的管理和通信、硬件资源的抽象和分配以及 I/O 设备仿真. 目前，Hypervisor 90%以上的市场份额被商业的 VMware ESXi①、Microsoft Hyper-V② 和开源的

① VMWare-esxi. https://www.vmware.com/products/vsphere-hypervisor

② Microsoft Hyper-V. https://technet.microsoft.com/en-us/windowsserver/dd448604.aspx

KVM①(Kernel Virtual Machine),Xen② 占据.

根据虚拟化层不同的实现方式,Hypervisor 可以分为独立监控模式(Type 1)、宿主模式(Type 2)和混合模式,见图 3.云计算主要使用独立模式和混合模式.在独立模式中 Hypervisor 直接在裸机上运行,例如 VMware ESXi.虚拟机对真实硬件的访问都要经由 Hypervisor 提供的硬件驱动程序.而宿主模式中,Hypervisor 是作为一个应用程序运行在宿主 OS(Operating System)中,例如 VMware Workstation.这样 Hypervisor 可以充分利用宿主 OS 提供的设备驱动和底层服务来进行资源管理.但虚拟机对硬件的访问需要经过 Hypervisor 和宿主 OS 两层软件.混合模式,例如 Xen,兼具了独立模式和宿主模式的优点.在结构上与独立模式类似,但 I/O 设备的仿真借用了宿主模式,交由特权虚拟机处理.然而,KVM 是一种区别于这 3 种模式的特殊 Hypervisor,这是因为 KVM 与宿主机位于同一层次.

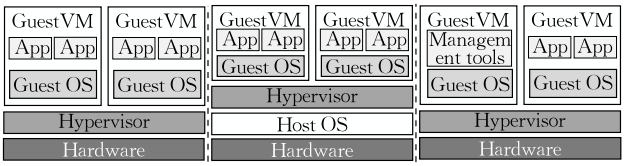

图 3 Hypervisor 的 3 种类型

根据是否需要修改 GOS,Hypervisor 又可分为全虚拟化和半虚拟化.全虚拟化下,Hypervisor 模拟与真实硬件完全相同的环境,不需要修改 GOS,具有很好的透明性.半虚拟化需要 GOS 调用 Hypervisor 提供的超级调用(hypercalls)完成对敏感指令和特权指令的模拟.虽然半虚拟化不具有透明性,但是降低了 Hypervisor 的复杂度.而硬件辅助虚拟化是全虚拟化的硬件实现,例如 Intel 的 VT[6]、AMD-V[7] 和 ARM 的 VE(Virtualization Extension)[8].GOS 只能在受控模式下运行,当 GOS 需执行特权操作时,CPU 自动切换模式.在完成操作后,Hypervisor 通知 CPU 切回 GOS 继续执行.硬件虚拟化已广泛应用于服务器平台.

3 虚拟化软件栈安全威胁

如图 2 所示,云框架包含很多层,每一层都含有不同的安全问题[9],但主要威胁来自基层的虚拟化[10].为了解决虚拟化中的安全问题,我们首先分析虚拟化软件栈中的攻击来源.然后,确定这些攻击来源可能导致的安全问题.在这一节中,我们将对攻击来源、安全威胁和攻击方式进行阐述.

3.1 攻击来源和攻击目标

图 4 描述了多租户模式下虚拟化软件栈的 3 层架构.按攻击层次可以将攻击对象分为 Hypervisor、GOS 和 Apps,其中 Hypervisor 和 GOS 是主要的攻击目标.在虚拟化平台中,客户虚拟机由平台的内部管理人员利用管理工具或软件通过 Hypervisor 提供的接口进行管控.这样内部人员可以通过管理工具对虚拟机进行恶意操作,例如虚拟机转存(dump)、快照(snapshot)和迁移(livemigration)[11],甚至虚拟机

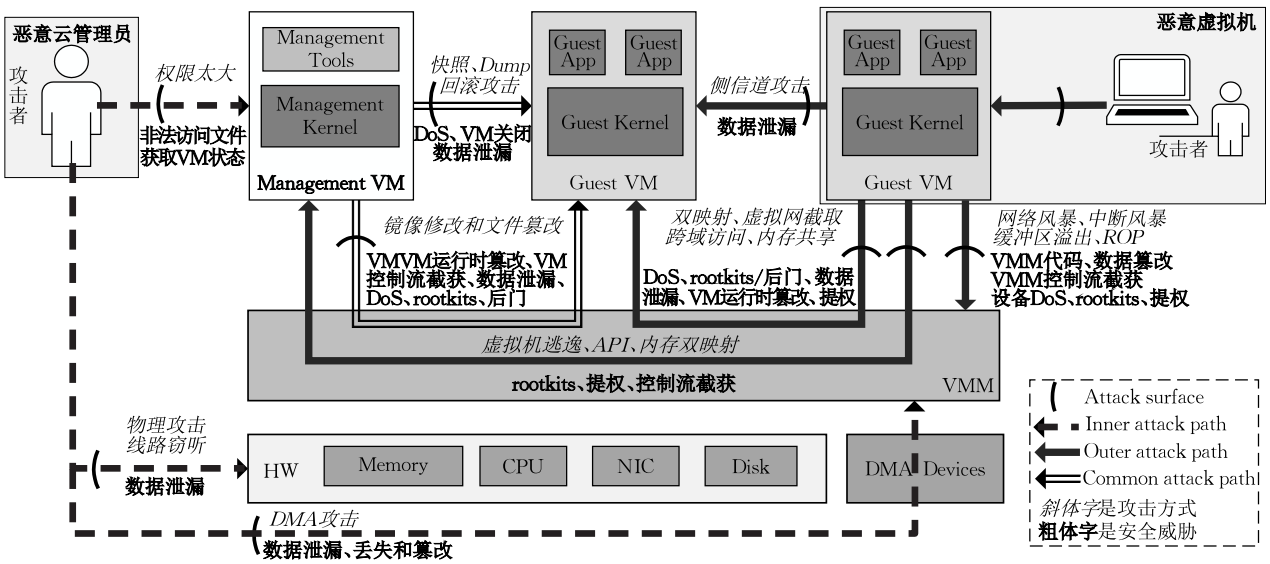

图 4 攻击面和攻击源

① KVM. http://www.linux-kvm.org/page/Main_Page
② Xen. http://www.xenproject.org/

镜像备份.而来自外部的网络攻击者则可以利用虚拟化软件栈的软件漏洞、脆弱性进行攻击,甚至可以租用同一台物理服务器上的虚拟机对其它虚拟机进行攻击.基于此,我们可以将攻击分为内部攻击和外部攻击.在图 4 中内部攻击路径和外部攻击路径分别用虚线箭头和实线箭头表示,而双线箭头是内部和外部攻击的公共路径.两者相比,内部攻击比外部攻击更易实施、成功率更大,而且不易被发现,带来的威胁和灾难危害更难控制.

虚拟化中存在众多安全威胁,而这些安全威胁来源于不同方式的攻击.图 4 对这些安全威胁和攻击方式进行了分析和展现.以数据泄漏为例,攻击者可以通过侧信道和虚拟机逃逸等攻击方式窃取其它虚拟机的数据.在此需要强调的是这些安全威胁不是独立存在的,是可以相互转化相互依托的.在 3.2 节和 3.3 节中将会对这些安全威胁和虚拟化中主要的攻击方式进行详细分析和讨论.

3.2　安全威胁

虚拟化中的安全威胁主要分为数据泄露和丢失、拒绝服务、权限提升、运行时(Hypervisor/GOS)代码和数据篡改以及控制流截获、rootkits 和后门遗留等.而数据泄露和丢失、运行时数据篡改以及控制流截获和 rootkits 是虚拟化威胁的主要表现形式.

威胁 1.　数据泄漏和丢失.

在传统系统中每个用户都使用自己独立的计算机处理数据.但是,在多租户的虚拟化环境下,计算资源是物理共享的,其隔离是软件层面的:CPU 的隔离是基于时间片轮转的;内存的隔离是基于 Hypervisor 维护的逻辑页表.虚拟化平台的多租户资源共享和动态迁移等特性使得数据泄露和丢失威胁在该框架中更为突出[12].在虚拟化平台中,数据泄漏表现为两种形式:外部数据泄露给虚拟机内部[13];虚拟机自身数据泄漏给外部软件(其它虚拟机或程序),后者是主要的安全威胁.DMA(Direct Memory Access)攻击、侧信道攻击、虚拟机跨域访问等方式都可以导致严重的信息泄露,给个人或企业带来不可估量的灾难.数据丢失与数据泄漏不同,数据丢失是指数据被删除、销毁或者毁坏.攻击者的目的并不仅仅是窃取用户的数据,而是对用户的数据进行破坏.近几年数据窃取和丢失事件在云提供商中不断出现,具体参考第 1 节中的示例.此类攻击的主要防护方式是数据加密、访问控制和隔离机制.

威胁 2.　控制流截获以及后门、rootkits.

控制流截获是攻击者利用系统漏洞或脆弱性使得指令的执行流程按攻击者的意图执行.控制流截获主要是对系统内部的控制结构(中断表、系统调用表)、跳转结构(控制指针、空指针)进行修改,或者利用系统原有指令组合执行(ret2lib 和 return-oriented programming).Rootkits 是一类特殊的恶意代码,攻击者若想实现 rootkits 攻击,首先要将其代码嵌入到系统中,然后通过截获控制流,使其代码执行.从而隐藏攻击者的行为或遗留后门,为后续攻击提供便利.上述攻击的主要目的是控制 Hypervisor 或 GOS,然后控制数据中心的网络,或者窃取用户数据,甚至可以通过控制众多客户虚拟机实施 DDoS (Distributed Denial of Service)攻击.这些攻击可以通过代码完整性、控制流完整性和影子备份等机制进行防护.

威胁 3.　拒绝服务(Deny of Service,DoS).

虚拟化层的 DoS 威胁在云服务中出现的概率很大,主要表现为用户不能够对其数据和服务程序进行访问和控制.虚拟化环境下的 DoS 威胁甚至包括对硬件资源(如内存、CPU 和网卡等)、以及虚拟网络等资源的访问不响应[14].有很多方式可以实现这种类型的攻击,通过网络风暴、中断风暴以及挂起硬件或控制服务,从而导致其它虚拟机产生资源饥饿、服务不响应.但是,虚拟化平台也为攻击者实施 DDoS 攻击提供了便利,在该环境下攻击者可以方便地批量租用虚拟机,或者利用虚拟平台的漏洞批量控制该平台上的虚拟机实施 DDoS 攻击.对于 DoS 威胁,主要的防护方式是对资源进行监控或者对吞吐量进行限制.

威胁 4.　虚拟机镜像威胁.

在虚拟平台中,虚拟机的启动和容灾恢复都是利用虚拟机镜像[15-16].虽然可以直接对虚拟机镜像打补丁防范虚拟机中的漏洞,但这也为内部攻击者提供了可乘之机.如果攻击者事先对虚拟机的镜像文件进行了修改、甚至替换,则启动后的虚拟机完全受攻击者控制.对于虚拟机镜像的防护主要是利用完整性验证方案,在系统启动之前对镜像文件进行完整性检测.

威胁 5.　运行时代码、数据篡改.

在启动过程中通过可信启动等技术能够保证系统和软件启动过程中的完整性,但是在运行时该类方案并不可取.代码篡改是一个非常严重的安全问题,攻击者通过缓冲区溢出、库函数映射等实现对 Hypervisor、GOS 等代码的注入和修改.这都可导致控制流截获、安全机制被关闭或绕过、提权、隐藏

攻击行为、遗留后门等.

威胁 6. 权限提升.

权限提升在虚拟化系统中尤为重要,包括两部分权限提升,从用户层到内核层,从内核层到虚拟化层.权限提升可以使得攻击者获得更高的权限,运行的代码级别更高,危害更大.在虚拟化中权限提升的主要表现形式是虚拟机逃逸,该威胁表现为客户虚拟机利用 Hypervisor 的脆弱性漏洞使得 Hypervisor 与客户虚拟机之间的隔离被破坏,从而导致客户虚拟机的代码运行在 Hypervisor 特权级,因此可以直接执行特权指令.

威胁 7. 不可信的云内部人员.

在通常情况下,云提供商员工被认为是可信的.但是 Google 员工窃听用户数据事件和棱镜事件足以证明云内部人员并非完全可信.用户将数据放在遥远的云端,对数据失去了控制,并不知晓云内部人员对用户自身数据所做的操作.此外,云内部人员可能拥有过高的访问权限,而且他们的行为不受防火墙和入侵检测系统限制.基于此种状况,内部人员在利益驱动下很容易侵犯用户的隐私,窃取用户的数据,甚至将用户的个人数据提供给第三方.以 Xen 为例,管理员可以对用户的虚拟机做快照和 dump 备份,甚至可以监听用户的网络.这类威胁的防护难度很大,需要从两方面入手:让虚拟机的管理过程对用户可见;提供安全有效的硬件机制保护客户的数据安全,具体的机制在 4.3 节会进行详细论述.

3.3 攻击方式

上面介绍的 7 类安全威胁需要通过一定的攻击方式才能得以实施.下面我们将对 5 类主要的攻击进行介绍.

攻击 1. DMA 攻击.

DMA 的初衷是允许外围设备绕过 MMU,直接对物理内存进行读写操作,从而提高 I/O 效率.在 Intel VT-d 提出之前,具有 DMA 功能的外设可以对物理内存进行任意访问,但 VT-d 的提出使得这一问题得到了缓解.DMA 攻击[17]主要分为 3 步:首先,对外设进行改造,嵌入恶意代码;然后,将外设部署到目标主机中;最后,利用恶意代码发送 DMA 请求,实现恶意攻击.DMA 攻击的难点是定位需要访问的数据结构或代码的地址,如此才能精确地实现有目的的攻击.不然,只能利用 DMA 进行粗粒度的数据窃取.在虚拟化场景下,内部攻击者则可以通过 DMA 设备,对物理内存中的代码、数据进行篡改或窃取,从而实现代码注入、控制流劫持和数据泄漏等.当前的主要解决方案是结合 IOMMU 对 DMA 的读写操作进行限制.

攻击 2. 多重映射和虚拟机跨域访问.

跨域访问是指客户虚拟机不仅能够访问自身的地址空间,同时还能够访问到其它虚拟机或 Hypervisor 地址空间中的数据.在 IaaS 模型中,每个虚拟机都有独立的 EPT(Extend Page Table)或 SPT(Shadow Page Table),并且 Hypervisor 拥有单独的地址空间.然而,攻击者利用一些软件漏洞、DMA 攻击、VLAN 跳跃攻击和 Cache 变更等实现虚拟机跨域访问.例如,攻击者利用 Hypervisor 漏洞或者已控制的 Hypervisor 对客户虚拟机(VM1)的页表进行修改,使其映射到另一客户虚拟机(VM2)的地址空间中,从而实现跨域访问.跨域访问能够窃取或篡改其他用户的数据或建立隐蔽信道.防止这类攻击的主要方式是对不同虚拟机之间进行隔离,并且剥夺 Hypervisor 更新 EPT 页表的能力.

攻击 3. 跨虚拟机的 Cache 攻击.

随着计算模式从独占计算硬件到云计算模式的迁移,基于共享 Cache 的侧信道攻击变得越发严重.基于 Cache 的侧信道攻击和隐蔽信道攻击[18]使得攻击者能够在数秒或数分钟内从当前流行的加密方法(RSA、AES 和 DES)中获取到受害者的密钥信息.基于 Cache 的侧信道攻击不需要获取 Hypervisor 等特权和利用其漏洞,而只需通过对时间损耗、电源损耗以及电磁辐射等特性的监测、统计即可获取到其它客户虚拟机的数据.侧信道攻击可以分为 3 种方式:基于时间驱动(time driven)[19]、基于轨迹驱动(trace driven)[20-21]和基于访问驱动(access driven)[22-23].基于时间驱动的攻击是攻击者重复地检测被攻击者的加密操作所使用的时间,然后通过差分分析等技术推断出密钥等信息.基于轨迹驱动的攻击通过持续地对设备的电能损耗、电磁发射等情况进行监控,获取到其敏感信息,但是这类侧信道攻击需要攻击者能够物理接近攻击目标.基于访问驱动的攻击是攻击者在执行加密操作的系统中运行一个应用,这个应用用于监控共享 Cache 的使用情况从而获取密钥信息.基于访问驱动攻击的优势是不需要攻击者得到受害者精确的时间信息.

攻击 4. 快照、内存转存威胁.

虚拟机快照(snapshot)是 Hypervisor 提供给

管理者的 API，用于容错和虚拟机维护. 云提供商的内部管理员可以利用管理工具对运行中的虚拟机进行快照，这为内部攻击者提供了便利之门. 这样可以在用户不知情的情况下，就可将虚拟机回滚（roll back）[24] 到特定阶段，从而绕过一些安全机制的更新. 内部攻击者甚至可以利用内存转存（memory dumping）工具对用户的内存进行转储，然后进行线下分析、窃取用户数据. 通常这类攻击的防护是利用密码学机制防护，或者禁用管理员的快照和转存操作.

攻击 5.　物理攻击和线路窃听.

物理攻击是指攻击者能够物理接近攻击目标所在的物理服务器. 虽然数据中心有专门的安全防护措施（比如录像监控和审计机制），但是数据中心的机房每天都有维修人员、清洁人员和管理人员出入，这就给安全带来了潜在的隐患. 冷启动攻击就是很好的例子.

通道或线路窃听可认为是另一种形式的物理攻击，攻击者通过一些特殊的方式监听受害者的通道和线路，包括外部网络、虚拟机之间的虚拟网络和内部总线等，从中窃取来自或流向虚拟机和 Hypervisor 的数据.

3.4　虚拟化软件栈威胁机理分析

以上所述攻击，主要发生在内存中. 这是因为系统启动之后，所有的操作和数据都是在内存中完成. 在现今的虚拟化方案中，内存的隔离是通过虚实地址映射实现的. 但虚拟化中的内存管理和传统 Linux OS 不同. 这是因为给虚拟机展现的内存并不是真实的物理内存，而是由 Hypervisor 虚拟出的连续的抽象物理内存. 正因为如此，虚拟机内存映射分两步完成：虚拟机内部的内存映射以及虚拟机到物理机的内存映射. 以 Intel 为例，每个 CPU 核心有两个寄存器分别存储 GPT（Guest Page Table）和 EPT（Extended Page Table）的基地址. 如图 5 所示，首先 GOS 将进程的 GVA（Guest Virtual Address）通过 GPT 转化为 Hypervisor 提供给它的抽象 GPA（Guest Physical Address），然后 Hypervisor 根据 EPT（per-VM）将 GPA 转换成 HPA（Hardware Physical Address）. 整个过程由 MMU 自动完成. 在硬件不支持 EPT 的情况下，MMU 完成地址转换是由 Hypervisor 通过影子页表直接将 GVA 转成 HPA.

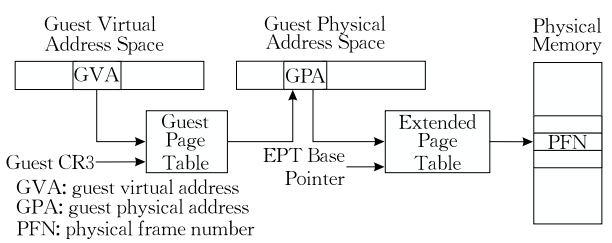

图 5　虚拟化下的内存转换

由此可见，Hypervisor 可以对其上运行的虚拟机进行内存隔离，但隔离的前提是 Hypervisor 是安全的. 为了防护本节中列出的安全威胁，学术界提出了很多针对内存安全的防护方案，在第 4 节中我们将会对这些解决方案进行详细的阐述.

4　虚拟化软件栈安全防御

虚拟化软件栈安全可分为虚拟机自身的（GOS、Apps）的安全和 Hypervisor（虚拟化层）的安全两个层次. 从可信基的角度分类，业界的安全方案可分为基于 Hypervisor 的保护（4.1 节）、Hypervisor 自身安全防护（4.2 节）以及虚拟机在不可信 Hypervisor 环境中的安全防护（4.3 节）. 而 4.3 节中的方案主要是在 Hypervisor 层之下引入新的软硬件安全模块，从隔离机制（4.3.1 节）、加密机制（4.3.2 节）和权限访问控制（4.3.3 节）这 3 个不同角度对虚拟机及内部软件进行保护. 4.4 节主要对侧信道攻击的防护进行阐述. 需要声明的是，在此节中我们不是针对所有的安全方案，而只对面向第 3 节中的重要安全威胁的解决方案进行系统的归纳、分析和总结.

4.1　基于 Hypervisor 的虚拟机安全保护

如图 2 所示，Hypervisor 位于虚拟化软件栈的最底层，拥有最高特权，并且拥有比传统 kernel 低两个数量级的代码量（见表 1），使得系统的攻击面更少、可信基更小. 除此之外，Hypervisor 对物理服务器的硬件资源拥有管理权和分配权，使得它能够截获到客户虚拟机对资源的请求和访问. 因此，从虚拟化软件栈的层次看，Hypervisor 无疑是保护虚拟机的最佳选择. 当前基于 Hypervisor 的安全防护研究主要围绕保护 GOS 安全、防护恶意 GOS 攻击用户进程以及利用虚拟机自省技术对客户虚拟机内部的攻击和恶意行为进行检测、分析.

表 1　kernel 和 Hypervisor 代码量对比

对象	代码量	可信基
Linux kernel	10 000 KLOC	Kernel
Xen	450 KLOC	Xen＋Dom0＋Qemu
KVM	380 KLOC	KVM＋Qemu
VMware ESXi	200 KLOC	ESXi

4.1.1 客户操作系统完整性防护

传统操作系统的防护有两个阶段:可信启动和运行时保护.可信启动可以利用可信计算等方式保证,但是运行时的威胁主要是恶意代码嵌入和rootkits攻击.内核rootkits是一类运行在内核空间的恶意软件,拥有与内核相同的特权级别.攻击者利用rootkits来隐藏恶意程序和文件,甚至遗留后门.传统的防病毒软件都是基于内核提供的数据,并不能用于防护rootkits攻击.因此针对破坏代码完整性的rootkits防护需提供一个与内核隔离的执行环境,在这个隔离的环境中实施安全策略.虚拟化层恰好提供了这样一个环境,并且拥有比内核更高的特权.Rootkits的防护方案有两种:(1)在代码修改或rootkits发生后,检测其存在;(2)通过对地址空间的隔离,阻止代码修改和rootkits嵌入.

传统的方案是通过分析内核静态和动态数据的完整性或控制流图的完整性,以此来探测系统中是否存在rootkits.文献[25]提出了基于状态的控制流完整性保护方案,该方案通过定期的对控制流的完整性进行动态验证,从而检测rootkits的存在.该方案由于定期监控,使得攻击者可以探测到监控规律,从而绕过监控功能.这类方案的最大不足是不能对rootkits攻击防范于未然.对此,文献[26]提出基于轻量级Hypervisor(NICKLE)的方案,该方案能够透明地阻止非法代码执行.NICKLE利用内存影射(memory shadowing)机制,透明地将已验证的虚拟机内核和可加载模块的代码拷贝到一个隔离的物理内存中作备份(shadow).这样只有认证的代码才会在安全隔离的内存中存在备份,在指令执行时,通过与shadow中的指令进行对比发现rootkits.但这只能保证代码的完整性,并且由于每次执行代码都要进行比对,性能损耗很大.上述两种方案是对静态代码进行分析,保证代码的执行流程,但是对于更改数据结构而截获控制流的rootkits,则不能防护.

为解决上述问题,通过对rootkits攻击原理的分析,研究者发现rootkits想要隐藏自身的行为,主要通过修改内核的控制结构和钩子.而这些敏感控制结构和钩子有一个显著的特点,即它们一旦被初始化则不会或极少再变更.基于此,Wang等人[27]提出了基于轻量级Hypervisor的针对钩子等数据结构的保护方案,即HookSafe.由于在Linux内核中,钩子和控制结构都是动态分配的,并且这些结构与其它数据共存在于一个页面中,而现有的软硬件技术都是基于页保护的.如果HookSafe直接利用现有的页保护机制,则会引发两个问题:不能直接对数据进行保护;带来极高的非必要的性能损耗.HookSafe为解决粒度鸿沟(protection granularity gap)问题采取了数据动态迁移、集中存储的策略,以此来提供细粒度(byte-level granularity)的数据防护.HookSafe的另一个贡献是提出了内核钩子重定向机制,并结合硬件的页保护机制对钩子的访问进行限制,使其对写操作进行验证.具体而言,HookSafe首先在线下进行分析得到所需结构体的地址,然后利用在线打补丁的方式对钩子函数的地址进行重定向,这在效率上仍有很大的提升空间.在执行时HookSafe直接重定向对这些数据的读取,而对于写操作则是调用超级调用陷入Hypervisor,由Hypervisor进行安全仲裁.但是HookSafe并没有给出具体的仲裁措施,只是利用线下分析得到的值作为参考,灵活性欠佳.HookSafe另一个不足是不能对LKM(Loadable Kernel Module)进行动态保护.

为解决LKM等内核模块问题,文献[28]提供了基于模块隔离机制的内核完整性保护方案.该方案利用轻量级Hypervisor(Barrier)提供的内存虚拟化功能对内核、静态模块以及动态扩展模块进行地址空间隔离,使得不同的模块位于不同的地址空间.在运行时只有当前模块所在地址空间的页表有效,其它模块的地址空间不可被访问.在该方案中,不同地址空间互相交互信息时需要经过严格的仲裁机制,即基于预先定义的强制访问控制规则进行判断.验证通过后才能变换页表空间,交互数据.相比其它方案,该方案有3个优点:Barrier只有极小的代码量;Barrier解决了LKM等动态内核扩展问题;并且提供了仲裁机制,能够对恶意模块进行防护.该方案在很大程度上提高了攻击内核的难度,保证了内核代码的完整性,但遗憾的是该方案只是基于PC机,并且仲裁机制比较固定,不易扩展.

上述基于Hypervisor的内核完整性保护方案可归结为3类.第1类是利用Hypervisor的内存虚拟化机制保护内核中的重要数据,比如HookSafe.该类方案利用EPT或SPT表的访问权限使得含有重要数据的内存页为只读.这样只有遵从Hypervisor仲裁的行为才能对这些数据进行修改.但是这类方案对于动态可加载模块不能验证.第2类是阻止非法代码在内核级运行.该类方案,比如NICKLE,对内核的代码进行影子备份或者在加载时对代码段的页表设置为只读,其它内核态空间设置为不可执行.这样在内核态执行其它代码时,则会陷入到

Hypervisor 中. 但是这类方案最大的不足是不能防御 return-oriented 类型的 rootkits，并且这类方法也不能灵活地动态加载其它模块. 第 3 类是通过将内核模块与内核隔离，使其利用不同的地址空间，比如 Barrier. 这是因为很多恶意代码嵌入内核主要是利用内核的动态可扩展性，比如动态可加载模块. 该类方案不仅解决了 LKM 对 Linux 内核完整性的威胁，并且将危害隔离在了模块内部.

4.1.2　防护恶意客户操作系统对进程的攻击

传统 Apps 的数据资源是由操作系统内核进行管理和分配，但是在虚拟化环境中，客户虚拟机内部的 GOS 的安全性较差. 因此，为保护进程的数据不被操作系统随意的泄漏和篡改，可以采用 3 种方式：影子页表、隔离的执行环境以及访问控制. 影子页表通过对同一进程建立两个不同的地址空间，对应于系统的内核态和用户态，这与 4.1.1 节中保护内核代码完整性的影子机制有异曲同工之妙. 而当内核态访问用户态的数据时，通过截获缺页错误，可以对数据的机密性进行保护. 隔离的执行环境是将内核和进程的地址空间进行隔离，保证进程运行在不受内核影响的环境中. 访问控制机制则首先由程序自身设置进程地址空间的访问权限，然后在 Hypervisor 的监控下系统内核按照既定的安全策略进行内存访问.

Overshadow[29] 利用了影子页表机制，该方案借鉴 Hypervisor 的 SPT 内存页表(见图 5)，对每个进程维持双影子页表，分别为系统影子表(system shadow)和应用影子表(application shadow). 通过这两个页表分别向内核和用户进程展现密文数据和明文数据. 虽然说 Overshadow 提供了两套影子页表，但用户进程的数据只存在一份，明文和密文是动态变化的. 这样 Overshadow 能够有效地防止运行时 GOS 对 Apps 代码、数据以及寄存器等信息的泄漏. Overshadow 的主要优点是加解密操作对 GOS 透明，不需要对 GOS 的内核进行修改. 为保证进程启动的安全，Overshadow 引入了 shim 机制，此机制还能够保证进程和 GOS 之间切换的安全. 但是，Overshadow 并不能防止 GOS 对用户进程数据进行篡改和删除. 另外，令人产生疑问的是由于 GOS 只能访问密文，驱动程序或者文件系统如何正常处理这些数据. 例如，用户进程需要将自身的数据通过 NIC(Network Interface Card)传递到外部网络中，Overshadow 首先将进程自身的数据加密后传递给 GOS 内核的 I/O 缓存，然后 GOS 内核将缓存中的密文数据封装成网络报文通过 NIC 转发到外部. 这样外部得到的数据将是密文，在论文中并没有对这部分进行介绍.

由于操作系统的行为过于庞大，区分其是否具有恶意性是一个非常艰难、复杂的事情. 基于此，InkTag[30] 提出了一种基于虚拟化的安全框架，通过有效且安全地验证 GOS 的行为，保证即使在恶意 GOS 中也能够安全地执行 HAP(High-Assurance Process). InkTag 首次提出了基于半虚拟化的验证(para-verification)机制，该机制强制 GOS 为 Hypervisor 和 Apps 提供验证自身行为的信息和资源，这很大程度上简化了 Hypervisor 验证 GOS 行为的复杂性. InkTag 借鉴了 Overshadow 的 shim 思想，利用超级调用在切换的过程中对 HAP 的上下文进行保护，并对内存页进行完整性检查和机密性保护. 除此之外，在运行的过程中 InkTag 限制了 GOS 对 HAP 寄存器等值的修改，这样 HAP 的控制流完整性就不会被 GOS 破坏. InkTag 并没有采取将 HAP 数据/代码与 GOS 隔离的机制，而是提出了一种基于属性的访问控制机制(attribute based access control). 基于这种机制，用户则可以灵活地设置针对 HAP 的访问控制策略，为其文件的机密性和完整性保护提供支撑. 但 InkTag 需要对 GOS 进行更改，同时需要对 HAP 进行重新编写，使其支持超级系统调用. 这种不透明的方案导致其兼容性不佳. 另外，InkTag 拥有与 Overshadow 一样的不足，没有说明设备驱动和 HAP 之间如何解决明文和密文问题. 最后需要指出的是 InkTag 方案需要 GOS 内核的参与，这为 DoS 攻击提供了机会，比如 GOS 内核拒绝将信息传递给 Hypervisor，同时新引入了 GOS 与 Hypervisor 交互的接口，增大了 Hypervisor 被攻击的可能性.

TrustVisor[31] 是一个专用的 Hypervisor，它利用硬件虚拟化的特性和 TPM(Trusted Platform Module)为进程和内核提供一个隔离的运行环境，称之为 PALs(Pieces of Application Logic). 这个隔离的执行环境能够保证其内数据的完整性和机密性. TrustVisor 有两个突出点：能够对应用进程的敏感代码和数据进行细粒度的保护；Hypervisor 的代码量很小，可以方便地进行形式化验证，这很大程度地减小了 TCB. Iso-X[32] 和 Intel 的 SGX[33-35] 都借用了类似的思想. 但隔离执行环境的缺点是需要由编程者指定隔离域，这需要程序员有良好的编程习惯和编程素养.

文献[36]为安全敏感型应用(security sensitive application)提供了一个安全执行环境(AppSec). AppSec 能够依据应用程序的意图保护用户的私有数据和人机交互数据,这与 InkTag 利用访问控制保护文件的思想相似. AppSec 的 3 个主要特点是:(1)利用隔离机制防止用户与系统设备交互的数据被恶意内核截获;(2)提供了一个基于特权的窗口系统. 该系统将窗口分成两组(高特权和低特权),使得只有受保护的进程(高特权)能够访问到自身窗口中的数据;(1)和(2)的前提是 I/O 设备驱动和 X window 服务运行在一个特定、隔离的专用系统中;(3) Overshadow 和 AppSec 都引入了安全加载程序,但是 Overshadow 只在启动时对用户程序的完整性进行了验证,并没有在运行时对共享动态链接库进行保护. AppSec 能够在运行时通过存储在 Hypervisor 中的哈希值对共享动态链接程序进行验证,保证共享动态链接库不被篡改.

要完成上述工作,AppSec 最大的挑战是如何让 Hypervisor 获取受保护应用进程的页面信息(因为存在语义鸿沟). 为解决该问题,该方案在运行时截获应用程序的系统调用,通过分析系统调用的参数,追踪受保护的信息. 而当 GOS 内核对受保护内存页进行访存时,AppSec 查看 GOS 访存地址是否在应用进程给予的访问范围之内. 这样就避免了利用密码机制对进程的数据进行加密,性能上优于 Overshadow 和 InkTag 方案. 但是,这在一定程度上无法保证数据的机密性. 最后需要指出的是 AppSec 对如何区分不同系统调用以及获取系统调用参数的描述较模糊,也没有对如何得到受保护进程指定的访问配置策略进行阐述.

4.1.3 虚拟机自省(VMI)

VMI 技术已经被业界广泛使用,比如 Amazon 和 Citrix XenDesktop. Hypervisor 利用虚拟机自省可以方便地对 GOS 内部的行为进行监控和分析,并结合蜜罐和入侵检测系统(IDS)对上层系统进行防护. 学术界对 VMI 的研究主要围绕两个方面:建立一个 VMI 通用框架;针对具体的安全威胁利用 VMI 技术获取相应的数据,以此进行系统防护. VMI 技术是从 Hypervisor 层获取虚拟机的信息,需要将低级别的字节码转化为高级别的包含虚拟机系统语义的信息(语义鸿沟). 为了克服语义鸿沟这一难题,VMI 采取 3 种语义隔离的方法:根据预提供的特定操作系统的语义信息,在 GOS 外部将监控数据进行语义还原(out-of-band);在 GOS 内部根据自身的内部信息提供结构数据(in-band);通过硬件结构的语义信息从客户虚拟机中获取所需数据(derivation),不依赖软件架构. 这 3 种隔离方法中前两种方法使用比较普遍.

LibVMI[37]通过对 XenAccess[38]进行扩展,提供了一套桥接 GOS 和 Hypervisor 语义鸿沟的通用库. LibVMI 提供了对 Hypervisor 的广泛支持,包括 Xen、KVM 和 Qemu 等,是当前应用最广泛的 VMI 框架之一. LibVMI 的设计原则是尽可能地减少对 Hypervisor 的修改,能够透明地获取客户虚拟机的内部信息(Linux 和 Windows),并且为上层用户提供友好的 API 接口和便利的扩展性. 文献[39]基于 LibVMI 提供了虚拟化环境下的入侵检测系统. 该系统弥补了传统病毒检测和 rootkits 探测系统的不足,并且能够深层次地对虚拟机内部的恶意威胁(比如 rootkits)进行实时检测. 文献[40]借鉴虚拟机自省技术,提供了一种可以对虚拟机内部信息进行验证的语言,该语言被称为 VMI-PL. 用户利用该语言可以方便地对虚拟机的内存数据、信息流、寄存器以及事件进行监控和拦截. 在 VMI-PL 中,用户以系统库的形式为应用提供数据、事件和数据流三类探针. 基于这三类探针 VMI-PL 不仅可以对客户虚拟机的内存数据进行探测,还可以对其输入输出以及事件进行探测. 但是 VMI-PL 的性能损耗很大,性能损耗高达 18%.

虚拟机自省技术与电子取证技术的原理类似,RTKDSM[41]利用电子取证工具 Volatility① 解决语义鸿沟问题,进而提供对虚拟机的内部数据的安全验证和监控. 在 RTKDSM 中存在两种模式:通用 VMI 模式和监控模式. 通用 VMI 模式由一个专用虚拟机利用 VMI 原理对虚拟机内部的数据进行提取,而监控模式是利用 Hypervisor 和 EPT 页表项的权限机制保证数据在发生更改后能够及时地被 VMI 机制得知. 但是该方案是基于页保护机制,而监控的对象是字节,因此在读写受监控页内其它非监控数据时也会发生异常,产生非必要的性能损耗.

图 6 展现的是基于 Hypervisor 的安全防护方案的框架图. Hypervisor 中的安全模块(影子页表、完整性验证模块、访问控制模块以及虚拟机自省等)独立于上层的客户虚拟机,用于对其上运行的虚拟机进行监控、授权和探测,并可结合 IDS、蜜罐和恶意行为分析等系统防护技术进行全面防护. 这些方

① An advanced memory forensics framework. https://github.com/volatilityfoundation

案能够对 GOS 的安全起到加强作用，但性能损耗还需进一步降低. 再者，Hypervisor 自身的安全问题未加以考虑.

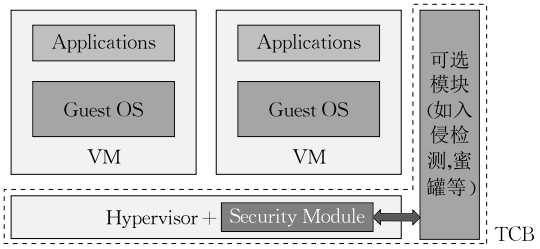

图 6　基于 Hypervisor 的 OS 防护框架

4.2　Hypervisor 及特权域的安全保护

4.1 节的方法只对恶意的 GOS 和 Apps 有效，并没有将虚拟化自身的威胁(Hypervisor 和特权域)和云管理员考虑在内①. 例如，上述方案中假设虚拟机之间是安全隔离的，并且虚拟机不能对底层的 Hypervisor 进行破坏，VMI 技术甚至利用特权域(包含整个系统内核)对 GOS 进行操作截获、数据分析. 然而，现今 Hypervisor 拥有越来越多的代码量，攻击面也随之剧增，安全问题日益突出②.

针对虚拟化自身的安全威胁，当前的云服务提供商通常采用被动打补丁的方式解决. 然而，这不仅给用户带来了不便，而且这种方式也只能防范已公布的漏洞，对于零日攻击或潜在的漏洞仍然无能为力. 为解决虚拟化自身的安全问题，研究者主要关注虚拟化 TCB(减少攻击面和代码)的减小和虚拟化自身完整性保护两个方面.

4.2.1　减小攻击面

攻击者可以通过在云平台租赁虚拟机直接对 Hypervisor 和特权域进行攻击，因此 Hypervisor 是攻击者面临的第一道防线. Hypervisor和特权域拥有的巨大攻击面是虚拟化层的最大威胁来源. Hypervisor 的代码量随功能的增强越来越多. 不仅如此，由于 Hypervisor 对硬件资源进行管理和分配，虚拟机需要和 Hypervisor 进行频繁的交互，然而交互越多受到攻击的可能性就越大. 根据测试，从虚拟机陷入到 Hypervisor 的主要原因(80%以上)是页错误和 I/O 请求[1]. 因此，研究者们开始尝试建立一种具有较小代码基和攻击面的虚拟化软件架构，其主要方式是减少 Hypervisor 代码量和改变虚拟化层的结构以减少客户虚拟机与 Hypervisor 的交互.

在 Xen 虚拟化架构中，特权域(Dom0)有两个功能：设备模拟和虚拟机管理. 在完成虚拟机管理过程中，Dom0 通过外映射(foreign mapping)将其他用户的虚拟机的内存页映射到自身空间中，这使得具有庞大脆弱性(代码量大)的 Dom0 可被攻击者用来作为攻击其它虚拟机的跳板. 因此为了防止恶意 Dom0 对其它虚拟机进行攻击、破坏，文献[42]提出将 Dom0 从虚拟化层的 TCB 中去除. 该方案主要为客户虚拟机提供了 3 点改进：安全的运行环境(寄存器和内存)、安全网络接口和后备存储. 在该方案中，为了给客户虚拟机提供安全的执行环境，Dom0 只能访问到 DomU 的密文信息(除 DomU 授权给 Dom0 的内存区域之外). 反之，当 DomU 访问自身信息时，则首先会对密文进行完整性验证，然后才进行数据访问，防止代码篡改和嵌入. 对于外映射内存，则由 Hypervisor 强制 Dom0 释放相应的页面.

TrustOSV[1] 同样针对 Xen 框架，旨在为用户提供一个可信执行环境. 该方案主要解决两个问题：减少虚拟机在运行时对 Hypervisor 的频繁交互，从而减少 Hypervisor 被恶意虚拟机攻击的机会；阻止内部人员从用户虚拟机的内存和存储设备中窃取数据. 为实现目的一，在启动用户虚拟机之前，TrustOSV 需要对虚拟机的初始状态进行远程验证，确保运行的是用户预期的虚拟机. 然后 TrustOSV 采用资源预分配的策略，即根据用户租赁的资源，TrustOSV 提前将 CPU 核心(cores)和内存分配给客户虚拟机. 资源预分配机制减少了虚拟机与 Hypervisor 的直接交互，减少了 Hypervisor 被攻击的机率. TrustOSV 的资源预分配策略能够防止跨域访问和跨虚拟机的 Cache 攻击. TrustOSV 的另一个特点是提供了丰富的远程验证服务，能根据用户的要求返回验证结果. 为达到目的二，TrustOSV 没有使用网络仿真机制，而是利用了一个软 NIC(在管理系统中)进行统一网络数据转发. TrustOSV 的认证依赖于 Hypervisor 中的代理. 该方案利用内存锁定机制(memory lockdown)保证代理的完整性，但是如何判断恶意和良性操作，文中没有给出说明. 除此之外，恶意 Hypervisor 提供给认证模块的数据可能是虚假数据，认证机制的不可伪造性不能被保证. 最后需要说明的是，TrustOSV 通过修改 GOS 的文件系统保证磁盘 I/O 数据安全，但并不保证网络 I/O 数据的安全.

TrustOSV 只是减少了客户虚拟机与 Hypervisor

① Black Hat: Mind Your Hypervisors, Says Security Researcher (Eduard Kovacs). http://www.securityweek.com/black-hat-mind-your-hypervisors-says-security-researcher, 2014, 8, 8

② Hypervisors the clouds potential security achilles heel (Steven J). http://www.zdnet.com/article/hypervisors-the-clouds-potential-security-achilles-heel/2014, 3, 29

的交互,而 NoHype[43-44] 认为 Hypervisor 的主要用途是便于虚拟机管理,并不是虚拟化中必不可少的部件.因此,NoHype 直接将整个 Hypervisor 从云虚拟化平台中移除.NoHype 利用设备的虚拟化特性[45],将设备进行物理隔离和预分配.这样在 NoHype 框架中客户虚拟机可以直接访问硬件.但在虚拟机创建、启动阶段,需要一个临时的 Hypervisor 完成硬件检测和虚拟机资源分配.NoHype 避免了虚拟机之间的资源共享,每个虚拟机具有独立的 CPU 核心和内存.这样跨域访问、重映射和数据泄漏攻击得到了很好的防御,比 TrustOSV 实现的更为彻底.NoHype 去掉了整个虚拟化层,大大减少了 Hypervisor 的攻击面,并且带来了性能上的优化.但是最大的不足是需要硬件虚拟化的支持,而对于磁盘存储和 GPU 硬件虚拟化还在尝试阶段,对于虚拟化桌面和具有图像(3D)、视频工作负载的客户虚拟机,NoHype 显然不是最佳选择.

NOVA[46] 借用了微内核的思想,提供了一种瘦小的虚拟化架构,该框架主要基于一个能被独立设计、开发和验证的可信模块.在 NOVA 中,传统的 Hypervisor 被拆分为不同的模块,主要有四部分:Micro-Hypervisor、根分区管理器(root partition manager)以及用户层的 VMM(Virtual Machine Monitors)和设备驱动,其中 Micro-Hypervisor 运行在 CPU root 模式的内核层,其余服务都以进程的形式运行在 CPU root 模式的用户层.这使得NOVA 在提高安全性的同时降低了底层接口的复杂度.NOVA 中每个虚拟机都有一个独立的用户 VMM 进程与之对应,这与错误域隔离机制相似,将危害限制在一个独立的域中.由于设备驱动以进程形式运行在用户空间,对于 DMA 攻击,也只能影响到自身的虚拟机.此外,NOVA 基于权能的访问控制,使得不同模块拥有不同的访问权限,每个模块依据权限机制拥有最小特权,防止了非法访问.但 NOVA 减小 Hypervisor 攻击面的同时也消减了 Hypervisor 的功能,并且在客户虚拟机因异常退出时(VM_EXIT)不仅要切换 CPU 模式,还需要切换 CPU 的运行级别.另外,NOVA 将部分原来 Hypervisor 的功能降级到用户层,这只是减小或减弱了这些代码所含漏洞的影响,并未根除.同时需要对现有的驱动等软件进行重写编写,兼容性差.

文献[47-48]是针对 KVM 的专用方案,该方案不同于需要对虚拟化进行重新设计的 NOVA,而是从逻辑上对 KVM 的代码进行分割.文献[47](HyperLock)提供了一个安全、隔离的 Hypervisor 运行环境.HyperLock 通过对 Hypervisor 的代码数据进行备份,使得每个虚拟机都对应一个独立运行的 KVM 副本(shadowed KVM),而且在这个独立的副本地址空间中只能执行有限的指令.文献[48]将 HyperLock 中的 KVM 副本进行重新改造,将 KVM 分成了两部分:低特权级 KVM(运行在用户层)和高特权级 KVM(运行在内核层).该方案将低特权 KVM 进行实例化,使得每个虚拟机都拥有一个独立的低特权 KVM 实例完成常用的虚拟化功能,类似于错误域隔离的思想,这与 NOVA 的 VMM 实例化一样,目的是通过地址空间隔离将攻击限制在一个可控范围内.而敏感指令、中断异常和硬件虚拟化等高特权操作交由共享且微小的高特权 KVM 完成.这在隔离攻击域的同时,降低了 Hypervisor 的可信基.但是这些方案需要对 KVM 的代码进行大幅度的改动,并且将 KVM 的代码移植到用户空间是一个难度大,且耗时费力的工作.

文献[42]通过将特权域(管理系统)从虚拟化层的 TCB 中剔除从而减小 Hypervisor 被攻击的可能性和防止特权域对客户虚拟机数据的窃取和篡改.TrustOSV 将管理系统运行在预分配的硬件资源中,并且通过减少虚拟机与Hypervisor 的交互防止客户虚拟机对 Hypervisor 的攻击.此外,TrustOSV 为用户提供了平台认证和软件认证的服务,使得用户相信虚拟化平台的安全性.与 TrustOSV 思想类似,但 NoHype 是从硬件的角度出发,直接去除了虚拟化层,将客户虚拟机直接运行在预分配的硬件资源上.但是由于现有硬件对虚拟化支持的不足,使得 NoHype 的思想只能维持在理论层次.NOVA 与前 3 种方案都不同,NOVA 是对虚拟化框架进行了重新设计,采用微内核的思想,将虚拟化层分为用户态软件和内核态核心.而 HyperLock 通过对 KVM 进行隔离达到与 NOVA 类似的目的.NOVA 和 HyperLock 使得虚拟机不能直接对 Hypervisor 进行攻击.并且,每个虚拟机对应一个用户层的虚拟机管理器实例,隔离了错误域,使得攻击的危害不易扩展.

4.2.2 Hypervisor 完整性

由于 Hypervisor 位于虚拟化的最底层,负责虚拟化的整个软件栈安全,同时也是攻击者面临的最后一道软件层防线.然而减少攻击面并不能完全保证虚拟化层的安全.因此,需要提供安全机制来保护 Hypervisor 的安全.

可信启动[49]能够保证 Hypervisor 启动的安全性，但是 Hypervisor 运行时的威胁却如梦魇般相随.保证 Hypervisor 运行时的安全最有效、最常用的方法就是保证 Hypervisor 运行时的完整性，其框架图见图 7.

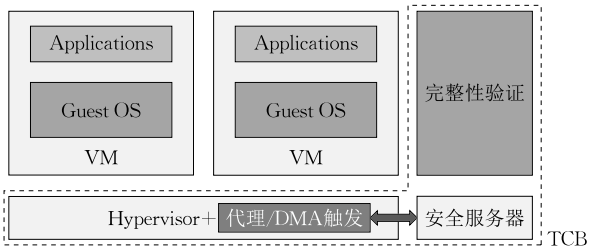

图 7　Hypervisor 的完整性保护框架

HyperCheck[50]和 HyperSentry[51]采用完整性验证的方式，实时地保证 Hypervisor 不被篡改，防范 rootkits 攻击等. HyperCheck 提供了一种基于硬件辅助的 Hypervisor 完整性探测框架，利用 SMM 对系统内存和寄存器制作快照，然后通过 NIC 将内存数据发送给远程的分析服务器进行分析(物理内存获取模块). 为了便于获取物理内存的数据，HyperCheck 提出借助符号表和寄存器值的方案，防止因拷贝-变化攻击(copy and change attack)而获取错误数据(寄存器验证模块)，例如攻击者不改变原有 Hypervisor 的页表，而是通过修改 CR3 等寄存器伪造一个新的页表. 远端分析服务器通过对内存快照分析而验证 Hypervisor 的完整性，同时利用带外机制(out-of-band)监测对 NIC 设备的 DoS 攻击(分析模块). 在上述方案中，完整性验证操作需要由 Hypervisor 自身触发，这期间可能存在擦洗攻击，即在完整性验证之前攻击者已擦除其攻击痕迹.

而 HyperSentry[51]提出了针对运行时 Hypervisor 完整性的隐秘度量框架. 与 HyperCheck 相比，HyperSentry 没有利用 PCI 设备，而是采用了软件组件的方式. 这种组件隔离于 Hypervisor，能够对 Hypervisor 的代码和重要数据结构进行隐秘的、实时的完整性度量. HyperSentry 最大的创新是利用智能平台管理接口(IPMI)隐秘触发度量操作，防止 Hypervisor 隐藏其攻击痕迹，从而避免擦洗攻击. HyperSentry 相比 HyperCheck 另一创新是突破了系统管理模式的限制，提供了完整性度量代理. 如此，使得即使触发 SMM 操作时客户虚拟机在运行，仍可完成对 Hypervisor 的验证.

HyperCheck 和 HyperSentry 只是对 Hypervisor 的静态数据或代码进行保护，并不能防护动态数据篡改引起的攻击. 它们应该结合 HookSafe 的思想对动态数据进行动态防护. 此外，他们都利用了 SMM 机制，然而 SMM 有自身的局限性：性能是正常模式的十分之一；在 SMM 操作时需要冻结所有的 CPU 核心；只能访问 4GB 之内的物理内存. 这给虚拟化框架带来了巨大的性能损耗.

在 4.1.1 节中我们提到完整性保护的方法有两种，完整性验证只能保证攻击 Hypervisor 后能被及时地觉察，但并不能防范于未然. 对此，HyperSafe[52]提出了在整个生命周期内，针对 Hypervisor(Type-1)控制流完整性的自保护方案(未引入新的软硬件). HyperSafe 提出了两个关键技术：不可被绕过的内存锁定机制(nonbypassable memory lockdown)和受限的指针索引(restricted pointer indexing). 首先是通过不可绕过的内存锁定机制保护 Hypervisor 的代码和静态数据不能被篡改. 一旦页面被锁定，任何对这个页的修改都需要先解锁. 而解锁操作禁止对 Hypervisor 的代码进行修改或引入外部代码，这提供了和 HyperSentry 等类似的代码完整性保护机制. 虽然不可绕过的内存锁定机制保证了 hypervisor 代码的完整性，但是控制结构或变量仍可导致控制流劫持. 由于内存锁定机制并不能灵活地适用于动态数据的保护，HyperSafe 提出数据重定向机制，其目的是将动态数据转换成指针索引(restricted pointer indexing). 而这些索引值是通过线下执行 Hypervisor 而得到的执行流图. 这样就把执行流的可能性限制在了可信集之内. 不可绕过的内存锁和受限的指针索引保证了 Hypervisor 控制流的完整性. 但文章中只是说明了在对页表进行更改时需要验证操作行为是否是合法的，并没有提出判断操作合法性的依据和标准.

总的来说，这些方案主要是通过完整性检测手段来发现针对 Hypervisor 的攻击，例如 rootkits 攻击、控制流截获和 DMA 数据篡改等. HyperSentry 比 HyperCheck 高明之处在于利用了硬件的特性，比如 IPMI、LAPIC 以及性能计数(performance counters). 但是它们两者都利用了硬件的 DMA 特性对 Hypervisor 的内存进行扫描、拷贝，这也为内部攻击者提供了契机. 同时对于具有 IOMMU 功能的芯片，则需要对 IOMMU 进行配置使其能够对 Hypervisor 的地址空间进行访问. HyperSafe 克服了 HyperSentry 和 HyperCheck 的不足，能够防护 return-oriented 类型的攻击. 但 HyperSafe 的自保护

机制需要进行代码剔除,其灵活性不高.

4.3 不安全虚拟化环境下的安全防护

4.2 节主要集中在保护 Hypervisor 完整性和减小 Hypervisor 的攻击面这两个方面.即使这样,仍不能确保 Hypervisor 的安全,Hypervisor 仍然存在安全漏洞和脆弱性.因此,在 Hypervisor 不可信的情况下如何保证虚拟机的安全则显得尤为重要.

4.3.1 基于隔离机制的防护

多租户模式和资源共享是云计算的主要威胁[53],而内存去重(memory deduplication)机制更加剧了这一威胁[54].客户虚拟机之间的隔离性是由 Hypervisor 维护的,而在不可信的虚拟化环境下同一物理机内虚拟机之间的隔离性难以保证,这可能导致客户虚拟机的数据泄漏、跨域访问、控制流截获等威胁.为此,研究者尝试更改软件和硬件架构对虚拟机进行安全隔离,其抽象架构如图 8 所示.在图 8 中,通用内存是虚拟机共享的资源,而安全内存是安全模块专用的内存.

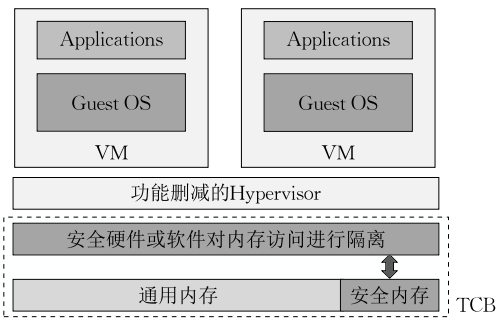

图 8 Hypervisor 及虚拟机之间内存隔离框架

既然 Hypervisor 的安全威胁很难防护,且云提供商内部员工的最小权限很难适当划分,研究者尝试将 Hypervisor 的机制(管理)和策略(控制)进行分离以此来保障虚拟机的安全.基于此,CloudVisor[55] 首次提出了将决策权和管理能力分离,透明地对虚拟机进行保护.CloudVisor 方案主要有两个贡献:利用嵌套虚拟化技术将资源管理和权限控制隔离;提出了基于页属主的虚拟机隔离机制.Cloudvisor 利用嵌套 Hypervisor[56] 机制将传统 Hypervisor 的决策权下降给底层微小的嵌套 Hypervisor,这样限制了传统 Hypervisor 对虚拟机资源的直接访问.CloudVisor 通过将传统的虚拟机的 VM_EXIT 入口重定向到自身,这样 CloudVisor 就可以截获虚拟机与传统 Hypervisor 的交互.在此过程中,CloudVisor 需对相关信息进行记录、存储以备后续之用,而虚拟机异常的处理则交由传统的 Hypervisor.CloudVisor 为了将不同虚拟机以及虚拟机与 Hypervisor 之间隔离,在分配页面时通过查询页面属主表保证一个页面只能属于一个虚拟机.除此之外,CloudVisor 限制 Hypervisor 对虚拟机 EPT 表的更新,防止恶意 Hypervisor 违反页属主唯一的原则.CloudVisor 同时借鉴了 Overshadow 加密机制,在 Hypervisor 申请更新 EPT 表时,如果页面属主与 EPT 不匹配则需要对页内数据进行加密.而对于 DMA 攻击,CloudVisor 则在 IOMMU 中利用了与内存隔离相似的机制.CloudVisor 将虚拟机的内存从 IOMMU 页表中移除.当对这些页进行 DMA 访问时,则会产生 IOMMU页错误,然后交由 CloudVisor 进行处理.通过这些机制,CloudVisor 能够保护虚拟机的数据不被其它虚拟机窃取.然而在虚拟化中,虚拟机需要频繁地与 hypervisor 进行交互,这使得 CloudVisor 在切换过程中性能损耗较大.并且由于 CloudVisor 引入了新的软件层,在一定程度上增加了虚拟化软件栈的复杂度和潜在安全漏洞.

与 CloudVisor 类似,SMMU[57] 和 H-SVM[58-59] 通过对 CPU 进行逻辑扩展实现虚拟机间的隔离.该方案的隔离性并不基于软件层,可信基只包含内存、CPU 和 Cache 等硬件资源.H-SVM 方案借鉴了 CloudVisor 的内存属主机制,并且基于 CPU 的 SMM 运行模式利用微码实现了虚拟机内存页的隔离.H-SVM 首先扩展了内存分配(map)和释放(unmap)指令,在 Hypervisor 进行内存管理时,需要调用 SMM 中的微码,只有通过对物理页属主的审查后才能进行相应内存分配.H-SVM 借鉴了 CloudVisor 对内存页属主的跟踪机制,利用 OPT 表(Ownership Page Table)记录每个物理页的属主.在 Hypervisor 调用 map 分配内存时,H-SVM 首先根据安全内存中的 OPT 表判断 Hypervisor 分配的物理页是否已有属主.如果这个物理页是空闲的,则分配给这个虚拟机,并标记其属主为此虚拟机的 ID.这样保证了每个物理页只能分配给一个虚拟机.但是对于虚拟机之间和虚拟机与 Hypervisor 之间的共享内存,H-SVM 提供了一个接口可供客户虚拟机自己设置共享页.在 H-SVM 的后期实现中,作者利用 SMM 的隔离性和高特权级进一步对 H-SVM 进行了扩展,将安全模块与 Hypervisor 进行了完全隔离.但 SMM 的缺点也是显而易见的.H-SVM 需要 Hypervisor 配合来完成其操作,这对 Hypervisor 不透明,并且也没有像 HyperSentry 提供隐秘触发机

制.此外,H-SVM 只能防范软件层攻击,对于物理攻击无能为力.NoHype[43-44]也利用了隔离的思想,将用户的操作、数据等限制在一个独立的环境中.但不同的是,NoHype 是对物理资源进行绝对隔离,做到了资源专用.NoHype 不仅可以防止跨域内存访问和 Cache 共享,还可以避免客户虚拟机间的相互影响.

上述方案的内存隔离粒度都是基于页级别的.Iso-X[32]和 Intel SGX[33-35]通过对 CPU 进行安全扩展将内存的保护粒度缩小到了隔间(segment),其保护对象不再是客户虚拟机,而是将保护对象缩小到进程内部模块.在 Iso-X 和 SGX 方案中,每个进程的地址空间被分割成多个安全区域和不安全区域,这些区域是由程序员编写程序时设定的.这样只有程序员指定的安全代码域能访问到指定的敏感数据,非安全区的代码对这些敏感数据的访问会引发异常.Iso-X 利用位图中的比特位(bit)来对每个物理页进行记录,用于标记这个物理页是否属于一个安全区域,这样在每次内存访问时,只需要对这个页的比特位进行查询,就可以保证不可信区域访问不到敏感数据.Iso-X 相比 Intel SGX,能够对安全区域中的页进行动态增加,在 SGX 中一旦隔间被创建则立刻进行封装,不可再被扩展.然而,Intel SGX 利用封装机制能够对数据进行认证、加密等,防止其内数据被篡改和窃取.这两种方案都是在 CPU 访存时对操作进行判断,而指令级的访存非常频繁,带来的性能损耗较大.再者,这两种方案需要从编程语言、编译器以及 CPU 硬件整个架构进行全套更新,适用性比较差.

H-SVM 方案与 CloudVisor 最大的不同是 H-SVM 是基于硬件实现的.除此之外,H-SVM 为客户虚拟机提供了灵活的共享页配置接口,而 CloudVisor 不能提供虚拟机之间的共享.但是 H-SVM 需要 Hypervisor 调用 SMM 中微指令对内存分配和释放进行保护,需要对 Hypervisor 进行一定程度的修改,并且由于 SMM 的天生缺陷使得 H-SVM 的性能较差.CloudVisor 通过截获 VM_EXIT 不需要 Hypervisor 的参与即可完成虚拟机之间的隔离.NoHype 提供的是虚拟机之间的物理隔离,因此其安全性是最高的.Iso-X 和 SGX 不同于 H-SVM 等方案,这两种方案的保护粒度是隔间,保护对象更具体,同时减少了非敏感数据保护而损耗的性能.但这两种方案的推广和实施需要一定的时间.

4.3.2　基于加密机制的防护

虽然隔离机制比加密机制拥有更小的性能损耗,但不能实时地防范物理攻击.在传统的 CPU 架构中,内存数据面临这样的安全悖论:加密,数据无法被 CPU 处理;不加密,数据的安全性和隐私性得不到保证.因此,若想 CPU 能够处理内存中的加密数据,则需要对 CPU 进行加/解密安全扩展,使其拥有处理密文的能力.CPU 安全扩展可以很好地为云虚拟化环境提供由软件到硬件的可信链路,框架图见图 9 所示.

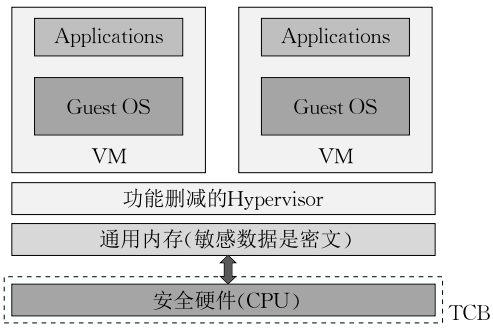

图 9　利用加解密防止物理攻击框架

随着云计算的发展,研究者开始关注利用 CPU 的安全扩展保护虚拟机的安全,扩展的主要模块是 Cache、TLB 及其处理逻辑.Bastion[60]利用微处理器来加强 Hypervisor 抵抗物理攻击和软件攻击,并提供基于模块化、细粒度的策略用于保护虚拟机内部进程的数据.在 Bastion 框架中,应用程序分为多个模块,每个模块有独立的模块号.Bastion 在 CPU 的 Cache 和 TLB 中进行了模块和策略扩展,保证只有进程内部的模块能够访问自己的数据.但 Bastion 只支持单处理器,并且不适用于保护整个用户进程和用户地址空间的数据.SecureME[61]弥补了 Bastion 的不足,它借鉴了 Overshadow 的思想,针对 Hypervisor 和客户虚拟机分别表现为不同的数据形式,是 Overshadow 在硬件上的实现.此外,SecureME 通过对页表权限进行扩展,来实现不同进程之间的安全通信和页共享.SecureME 采用 AISE(Address Independent Seed Encryption)内存加密算法和 BMT(Bonsai Merkle Tree)哈希树[62],保证数据的机密性和完整性.AISE 的优势是不直接对数据进行加密,只加密数据对应的 seed,然后利用加密后的 seed 与数据异或产生密文.Seed 是地址独立的,并且每加密一次都会发生改变.而 BMT 是针对 AISE 存在的,在 BMT 中只对 seed 的组成部分进行完整性保护,而不是对整个数据区域进行保护.

SecureME 利用 AISE 和 BMT 极大地减小了性能损耗和空间损耗.但是 SecureME 与 Bastion 有一个共同缺点,即把 Hypervisor 放在了一个隔离的环境中作为 TCB 的一部分.

HyperCoffer[63]同样利用 AISE 和 BMT 保证虚拟机内存数据机密性和完整性.在该方案中,将内存虚拟化从不可信的 Hypervisor 中解耦出来,使得内存管理不依赖于虚拟机对内存的访问.其次,对 CPU 虚拟化进行更改,针对不同的虚拟机退出给予不同的处理,由硬件提供必要的技术支持.再次,对 I/O虚拟化进行控制,由虚拟机对 I/O 控制信息和数据进行保护,从而在不修改硬件设备的前提下,实现与硬件的交互,使得尽可能多的数据处于保护状态.就内存管理而言,在 HyperCoffer 中每个虚拟机拥有自己的密钥 K_{vm}.HyperCoffer 利用 K_{vm} 对种子(seed)进行加密,产生一次一密的 pad,然后利用 pad 与内存页中的数据异或产生密文.而种子的保护是通过 BMT 哈希树完成的,BMT 树的根(root)存放在虚拟机表中,由 CPU 自身提供的 K_{mem} 进行加解密和 BMT 保护.为了避免 CPU 和虚拟机之间的语义鸿沟,在 HyperCoffer 中引入了 VM-shim 机制.当从虚拟机切换到 Hypervisor 时,首先利用 shim 在 Hypervisor 和虚拟机之间传递必要的数据,从而使得虚拟机不需修改即可在 HyperCoffer 上运行.对于磁盘 I/O,因为在 Cache 和内存之间进行了加解密操作,因此 shim 不需要对这些数据进行解密就可以直接写入到磁盘中.HyperCoffer 虽然没有将Hypervisor 作为其可信基的一部分,但是引入的 VM-shim 机制同样增大了可信基,并且 VM-shim 和客户虚拟机都必须可信.另外,HyperCoffer 的加解密能够解决磁盘存储的安全,因为内存中存储的数据都是密文.但文章中没有提供针对网络 I/O 的解决方案.

Intel 的 SGX 利用了粒度更小的内存保护机制.在 4.3.1 节中已经提到 SGX 不仅提供了隔离机制,还利用数据封装机制抵御物理攻击.Intel SGX 的目的是将敏感数据存放在一个隔离的执行环境中,而这个执行环境由 CPU 进行保证,而且当外部代码访问敏感数据时,不允许对数据进行解密操作,防止了数据泄漏.在文献[64]中,研究者通过在 Windows 系统中添加一个 SGX 驱动程序模拟了 SGX 技术,以此保证上层进程数据的安全.这是一次对 SGX 技术方案的系统验证和实践.

Bastion 和 SecureME 通过修改 Hypervisor,并结合硬件安全扩展实现对虚拟机内部模块和进程的保护,虽然 Bastion 和 SecureME 将 Hypervisor 作为自身 TCB 的一部分,但是这两种方案都对 Hypervisor 提供了保护机制.HyperCoffer 虽然将 Hypervisor 排除在其 TCB 之外,但是 HyperCoffer 引入了 VM-shim 机制,并且保护的对象是虚拟机,其前提条件是虚拟机内部的操作系统和应用程序都是可信的.显然这些方案各有千秋,最好的方案是将 Hypervisor 和虚拟机的 GOS 都不作为 TCB 的一部分.Intel 的 SGX 正是出于此目的,利用硬件机制保证虚拟机内部进程的安全,保护粒度比 Bastion 和 SecureME 更小.SGX 不仅使用了隔离机制对安全和非安全的代码、数据进行分离,并且利用加密机制对数据进行封装,保证数据的机密性,防止物理攻击.但是 SGX 利用了专用 Cache 机制,将安全代码放在了 EPC(Enclave Page Cache)中,然而 Cache 毕竟有限,需要频繁地将一些代码块换出到常用内存中,这导致频繁的加解密和完整性验证,性能损耗较大.并且 EPC 访问时还需要对一些结构体进行查询,这更加剧了系统的性能损耗.

4.3.3 基于访问控制的防护

访问控制的目的是通过限制主体对客体的访问能力及范围,保证客体不被非法访问.当前的所有的硬件框架(x86、Arm、Power 等)和虚拟化软件栈(hypervisor、GOS、Apps)都使用包含性访问内存权限集(inclusive memory permissions).这导致高优先级的软件层能够无限制地访问低优先级软件层的代码和数据.在虚拟化平台中这种机制威胁着整个软件栈的安全.因此我们可以利用访问控制机制来阻止这一威胁,使其满足以下两点:防止非法用户访问受保护的数据资源;允许合法用户访问受保护的数据资源.

HyperWall[65]首次利用权限预设定机制来保护物理页不被 Hypervisor 和 DMA 恶意访问.其主要思想是利用 CIP(Confidentiality and Integrity Protection)表对每个物理页标注 Hypervisor 和 DMA 的访问权限.为保证 CIP 表的安全、权限设置与检查的正确执行,HyperWall 为其提供了一个安全可信的执行环境和存储域.但是,HyperWall 的权限机制需要用户自己为其虚拟机提供一个 pre-CIP 表,在这个 pre-CIP 表中指定了允许被 Hypervisor 和 DMA 访问的虚拟机的地址区域.在虚拟机创建时,或者有映射更新时,HyperWall 则根据用户提供的 pre-CIP

表(用户指定的是 GOS 的物理内存的访问权限)来设定真实物理内存的访问权限.在内存被 Hypervisor 或 DMA 访问时,HyperWall 查阅 CIP 表查看其是否有权进行访问.不仅如此,HyperWall 还提供了认证机制,在虚拟机启动完成之后,HyperWall 利用在启动过程中得到的 K_{enc} 和 K_{hash} 对虚拟机的初始状态进行哈希计算,以此来保证用户提供的 pre-CIP 表没被篡改.在虚拟机运行过程中,用户也可以进行虚拟机安全验证,查看是否存在非法访问自己虚拟机的内存,这也可为 IDS 和恶意行为检测等提供依据.但 HyperWall 方案也存在不足,它对虚拟机用户的要求较高,需要用户对虚拟机系统的内存部署非常熟悉,这样才能对每个页面的权限进行正确设定.HyperWall 另一个不足之处是数据遗留,对于异常终止的虚拟机,不能对其内存进行安全回收.

而 NIMP[66] 提出了一种非包含性的内存权限机制,该机制能够抵御跨层(cross-layer)攻击.在 NIMP 中每个物理页拥有一个两字节的权限位集合,其中的 9 比特位(bits)用来表明 Hypervisor、内核和进程对这个页的访问权限,这些访问权限是在分配页面的时候由安全硬件模块依据权限规则进行设定.而访存指令会进行相应扩展,添加该访存指令应具有的访存权限.这样就保证了低特权层(GOS)的内存页不能被高特权层软件(Hypervisor)访问.而在 CPU 的特殊 Cache 中存在 7 条权限转换规则,以此保证权限不能被非法转换.例如,NIMP 中提供的规则保证了攻击者不能将可写的页直接转换为可执行.但是双映射和跨域访问攻击仍然存在.例如,同一宿主机中运行的两个虚拟机,其系统分别为 GOS1 和 GOS2.GOS1 的指令具有 RW 权限,而 GOS2 的内存页的内核权限是 RW.利用重映射使得 GOS1 可以访问 GOS2 的页,从而实现跨域访问.因此,NIMP 只是一个雏形,尚需进一步研究.

上述方案都是利用访问控制来对内存的安全进行防护.HyperWall 利用 CIP 表对恶意 Hypervisor 和 DMA 进行防范,但是保护对象是客户虚拟机,对于虚拟机内部的安全威胁则是无能为力.NIMP 与 HyperWall 的思路不同,NIMP 的保护对象可细化到应用程序,主要是防止跨层攻击,使得高特权层的软件只能按照低特权层软件(用户层)预期的访问规则进行访问,以此满足用户的安全需求.这两种方案都需要对代码进行修改,HyperWall 需要修改 Hypervisor,而 NIMP 对进程、GOS 和 Hypervisor 都要修改.NIMP 只是一个特殊类型的方案,该方案针对的是特定攻击类型,并不能作为通用方案防护大部分攻击.

4.4　侧信道攻击和隐蔽信道攻击的防护

在云计算虚拟化环境中,不同客户虚拟机共享资源,对于内存资源可以通过隔离、加密和安全 CPU 等方案进行防护,但是对于共享 Cache 的侧信道攻击,这些方案却无能为力.从已有的研究可知,侧信道攻击已经从 L1 Cache 渗入到 LLC(Last Level Cache)[67],这使得攻击强度更大,获取的信息更多.

基于时间驱动的侧信道攻击主要根据加密算法所用时间来推测密钥信息.根据攻击者所处的位置,基于时间的侧信道攻击又可分为两类:被动攻击和主动攻击.被动式攻击有一定的局限性,因为攻击者不能够直接访问和操作受害者的 Cache,攻击者不能获取其所监测操作的精确使用时间(如受网络延迟影响).而主动攻击能够和受害者同处于一台物理机,这样攻击者则可以在其上运行代码,如此便能够直接操作 Cache 和计时器.基于轨迹驱动的侧信道攻击是危害最大且最有潜力的侧信道攻击,但该类攻击需要攻击者和受害者处于同一台物理机.该类攻击最具有代表性的策略是 Prime+Probe,即攻击者首先访问特定的内存地址用于填充 Cache 行,特定时间后攻击者再次访问这些地址,测量其访问所用时间.如此循环多次,利用测量到的时间就可以推算出受害者的密钥信息.基于访问驱动的侧信道攻击不仅需要攻击者与受害者处于同一物理机,而且攻击者还需要频繁地查看和测量 Cache 的状态,从而获得数据.攻击者首先通过自身程序访问一些固定的地址填充 Cache,然后转让给受害者运行,攻击者的数据将会从 Cache 中踢出.然后攻击者抢占继续执行,经过特定访问和检测 Cache 的命中情况,攻击者可以得到受害者访存的信息.这些攻击依赖 3 个条件:一是共享 Cache 的状态可被攻击者访问,并且根据其不同做出预测;二是可抢占执行,并且对 Cache 状态影响极小;三是能访问系统时钟,以此区分事件的不同.

针对时间驱动和轨迹驱动的攻击,文献[68]提出了一种系统级的基于 Cache 的侧信道防护方案(STEALTHMEM),该方案在 LLC 中为每个 CPU 核心提供一组锁定的 Cache 行,这些 Cache 行不会因其它 CPU 核心的 Cache 紧缺而被换出.该方案从上述侧信道攻击的第一个条件出发,防止攻击者获取硬件 Cache 的状态信息.但由于 Cache 没有提供硬件机制保证 Cache 行的独立性,该方案由 Hypervisor 提供一些隐形内存(stealth memory)来确保每个

CPU 核心都有一些锁定的不被共享的 Cache 行. 这样在虚拟机之间切换时,将该 CPU 核心的隐形内存放入被换出虚拟机的上下文中,并从换进虚拟机上下文中将隐形内存读入 Cache 中. 如此这样切换之后观测不到被换出客户虚拟机对 Cache 状态的影响. 这样每个虚拟机将自身的敏感数据放在各自的隐形内存页中,而这些隐形内存页在 Cache 中不进行共享. 这样隐藏了敏感数据的访存模式,从而防止了跨虚拟机的侧信道攻击.

侧信道攻击除 Cache 共享之外还需要抢占执行和获得精确时钟. 基于此,文献[69]提供了基于软件的 Cache 隔离方案,该方案利用虚拟机的调度机制,保证每个虚拟机的虚拟 CPU 的最小运行时间,从而限制了抢占式调度的频率,使攻击者不能够及时、准确地得到 Cache 的状态信息. 但是这也为高优先级或者交互性任务带来一定的延误. 该方案是在虚拟机之间调度时,通过加入噪音等方式防止跨虚拟机的侧信道攻击.

4.5 讨论和总结

上述安全机制的分类是根据可信基来划分的,但并不是绝对的,可能存在重叠. 例如,NoHype 既可以归类到减少攻击面和 TCB,也可以归类到通过隔离机制保护虚拟机. 从我们的文献调研中得知,针对虚拟化层的安全防护方法主要有 3 大类:第 1 类是基于 Hypervisor 的虚拟机保护,这类方案利用 Hypervisor 的高特权对虚拟机进行监控和保护;第 2 类是基于软件重构或完整性对现有 Hypervisor 的保护;第 3 类是在不可信 Hypervisor 环境下的虚拟机安全防护,该类方案利用隔离机制对虚拟机以及 Hypervisor 进行域隔离,并且从加密和权限划分两个方面防范高特权软件层对低特权软件层的内存访问. 基于 Cache 的侧信道攻击是虚拟化中的特殊攻击类型,其防护方案也是软硬兼有. 总的来说,这些方法涵盖了当前虚拟化安全的重要方面,从软件到硬件,从应用程序的保护到Hypervisor的保护,从减少攻击面到添加附加安全模块.

通过对这些方法的分析,在表 2 中我们将现有的解决方案细分为 10 类,并分析了其威胁抵御情况. 前 3 类方案是基于 Hypervisor 的安全保护:其中前两类是对虚拟机的内核和进程的保护方案;第 3 类

表 2 代表性方案与安全威胁防御对应表

类别	方案	威胁														
		拒绝服务	代码篡改	控制流截获	rootkit 后门	数据丢失	数据泄漏	DMA 攻击	多重映射	跨域访问	VM 镜像篡改	Cache 攻击	内存转存	物理攻击	线路窃听	权限提升
基于虚拟化的内核完整性防护	NICKLE		√		√											√
	HookSafe		√	√	√											
	Barrier		√	√	√											
基于虚拟化的敏感进程保护	Overshadow		√				√	√								
	InkTag						√		√							
	TrustVisor		√				√	√								
	AppSec		√				√		√							
虚拟机自省及其应用	LibVMI			√	√											
	文献[39]			√	√											
	VMI-PL			√	√											
	RTKDSM			√	√											
减少虚拟化软件栈的可信基(特权域和 VMM)	文献[42]						√		√		√					
	TrustOSV						√		√	√	√	√				
	NoHype						√		√	√	√	√				
	NOVA						√	√	√							√
	HyperLock						√			√	√					√
Hypervisor 完整性防护	HyperCheck	√	√		√											
	HyperSentry	√	√		√											
	HyperSafe		√	√	√											
非可信环境下 VM 隔离	CloudVisor						√	√	√	√	√					
	H-SVM						√	√	√	√	√					
非可信环境下的内存区域隔离	Iso-X							√	√	√						
	SGX					√	√		√	√			√	√		
	Bastion					√	√	√	√	√	√		√	√	√	
非可信环境下 VM 安全	SecureME						√	√	√	√	√		√	√	√	
	HyperCoffer						√	√	√	√	√			√	√	
Hypervisor 的权限限制	HyperWall					√	√	√								√
	NIMP						√									√
侧信路防护	文献[68-69]											√				

通过 VMI 技术获取到虚拟机内部的数据结构，基于这些数据结构分析虚拟机的安全状况. 第 4 类和第 5 类方案是对虚拟化层的防护，为前 4 类方案提供安全基础：其中第 4 类方案是减少攻击面，这类方案的灵活性较差；而第 5 类方案是从完整性出发，保证 Hypervisor 代码和执行流的完整性. 虚拟化层的保护方案具有一定的局限性. 因此，后面的 4 类方案是在 Hypervisor 不可信的情况下提供安全保护. 最后一类方案则是对侧信道攻击的安全防护. 这些方案从不同的层次解决了当前虚拟化环境中的部分安全威胁. 例如，对于 Hypervisor 的完整性问题，HyperSentry、HyperCheck 以及 HyperSafe 解决了静态代码和数据的完整性，而 HyperSafe 还解决了控制流截获攻击. 而 NOVA 和 TrustVisor 等方案将 Hypervisor 的代码降低了两个数量级，减少了大量攻击面，给 rootkits 攻击、后门遗留提高了门槛. 而 CPU 安全扩展的研究旨在从根本上解决云虚拟化中的安全问题，为下一代 CPU 的设计指明方向.

从表 2 中我们可以看出，这些方案没有对现有的威胁进行全面的防护，仍然存在很多未解决的安全问题和空白区需要产业界和学术界去研究和填补. 例如，隔离机制能够防止不同虚拟机之间以及与 Hypervisor 之间的跨域访问，但是隔离机制并不适用于数据共享. 而基于权限的机制通过限制恶意访问保护数据安全，解决了共享问题. 除此之外，这些方案还存在一些共性问题. 第一，目前虚拟化软件栈中的安全问题备受重视，虽然产业界和学术界提出了很多研究方案，但是这些方案都具有很强的针对性，方案之间的关联性很差，没有提出一套体系化的针对虚拟化软件栈的保护方案. 这导致虚拟化软件栈的保护没有一个统一的框架，使得云计算厂商或公司不能够将这些不同层次的方案融合到自己的产品中. 第二，虽然现在硬件厂商和学者提出了利用 CPU 的安全机制提供数据保护的方案，但是这些方案并没有紧密结合云计算的特点（尤其是虚拟化的多层次和多租户特点）. 并且现今 CPU 架构是一种通用型的架构，产业界并没有提出一种针对虚拟化环境的专用 CPU 架构.

5　未来研究方向

虽然虚拟化安全不是一个新颖的话题，但是近年来云计算的发展给虚拟化带来了新的机遇和挑战. 虚拟化的安全问题再次回归为研究热点. 从表 2 可看出，近年来虚拟化安全虽取得众多成果，但针对现存的安全问题仍捉襟见肘，针对众多潜在的安全威胁和漏洞更是力不从心. 由此可见，虚拟化安全还有很大的研究和提升空间，迫切需要研究出一套高效、可行且易实施的虚拟化安全防护方案. 从研究的发展趋势看，我们认为未来的虚拟化软件栈安全防护应该着重从以下几个方向进行研究.

首先，结合虚拟化软件栈的特点，建立一个一体化的虚拟化纵深防御体系. 从图 4 中可以发现，虚拟化软件栈中的每层并不是单独存在的，而是相互关联、融为一体的. 然而，从第 4 节中可以得知，现在的安全方案主要针对虚拟化的每一层进行独立防护. 这种层层独立的方案不仅增加了性能开销，同时还存在众多重复工作，使得整个系统越来越繁杂、冗余，并且这些方案之间的兼容性较差. 因此在未来的工作中，我们应将虚拟化软件栈作为一个整体，结合虚拟化的多层次性，着重研究基于多层次的隔离机制、密码机制和访问控制机制，提出一个精炼、高效的多层次虚拟化软件栈防护方案. 如图 10 所示，通过统一的框架对 Hypervisor、虚拟机内部的 GOS 和应用程序进行统一的行为监控、资源信息控制、密钥管理、虚拟机间的隔离以及虚拟机内部的模块和应用之间隔离. 通过多层隔离机制能够对同一层的不同对象进行隔离，实现资源独立. 通过多层访问控制使得不同层对共享的资源分配不同的访问权限. 多层监控结合恶意行为分析实现整体的病毒、木马防护. 多层密钥管理对每一层的密钥进行抽象，提供统一的密钥管理接口，即安全又便捷. 统一框架能够从横向和纵向两个不同维度防护软件栈的威胁，还可以增加信息认证功能为云用户提供确凿的运行时证据，证明其所属数据和软件的安全. 在统

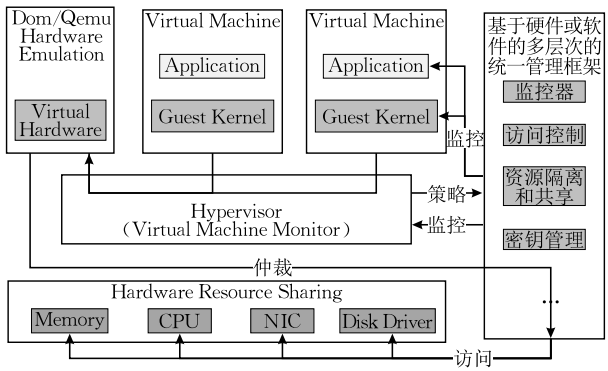

图 10　一体化的防御框架

一框架中依据软件栈的需求对数据共享以及动态迁移进行适应,在满足安全的同时保证云计算的服务质量.

其次,我们认为计算机安全体系结构是云计算安全的基石和未来重要研究方向,在硬件层为多层次的虚拟化软件栈防护提供技术支撑.现有 CPU 提供的安全机制,如不可执行(NX)、SMEP 和 SMAP 等只能防御来自用户空间的攻击.针对虚拟化系统级的安全,研究者尝试提出安全增强的 CPU 框架进行防护.受此启发,我们认为未来 CPU 及相关硬件应该从以下两点出发,为上层软件栈的多层防御提供支持.

第一,我们可以借鉴"软件定义网络"的思想,将单一物理服务器的内部看作一个"网络",而 I/O 设备(如网卡、磁盘)和 CPU(cache)等部件看作"网络节点",每个"网络节点"都有其自身的安全机制(如加解密)和存储部件(存储每个虚拟机的密钥).而内存和缓存等中转设备看作"路由节点",只进行数据存储和传递.根据软件层的需求,在每次创建虚拟机时由硬件芯片(如 CPU)为每个虚拟机创建随机密钥,并将该密钥传递给各个"网络节点".此密钥将和虚拟机镜像配对存储(虚拟机迁移时使用).在软件执行过程中,利用该密钥对每层的敏感数据(由程序、GOS 或 Hypervisor 指定)或在跨层或跨域访存的数据加密.

第二,对 CPU 的安全特性进行扩展,并结合现有的 CPU 特性(如 VPID 和 PCID 等)对多层防御方案提供支持.比如,在 TLB 进行内存地址转化时,对虚拟地址进行判别,若是代码则保证其完整性(图 11 中的 BMT 树).否则,则判断当前 CPU 的运行级别是否与数据地址所在的级别一致,一致则访问.否则(比如 ring 0 访问用户空间的内存地址),对数据进行加密,如在 I/O 缓存和系统调用访问时,则要对数据进行加密,然后在设备驱动或硬件芯片中对密文进行解密,从而完成相应操作.

这种软-硬结合的方案可以为虚拟化软件栈提供一个体系化的纵深防御体系,如图 11 所示.从图中可以看出,基于硬件的方案不仅能够防止软件层次的攻击,而且能够抵御内部管理人员的物理攻击,如线路窃听.云提供商可以利用该机制向用户提供充分的证据表明自身的可信度,消除用户对数据安全的疑虑.

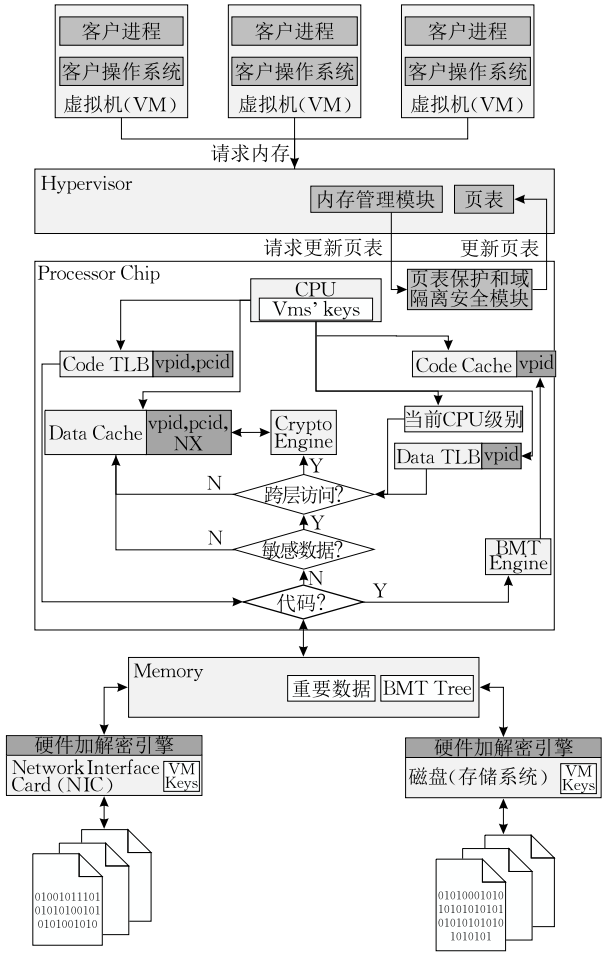

图 11 软硬结合的防御框架

6 结束语

云计算中虚拟化软件栈的安全问题是当前安全领域最重要的问题,其研究受到了工业界和学术界的广泛关注,近年来也取得了一定的进展,但安全问题仍未攻克.本文旨在对云计算环境下的虚拟化软件栈进行安全分析,并深入剖析虚拟化软件栈所面临的安全挑战.本文首先介绍了云计算架构的基本内容和虚拟化技术.其次,文中针对虚拟化软件栈的特性深入剖析了其可能存在的攻击面和安全威胁.随之,文中对虚拟化软件栈安全方面的现有研究成果进行了分类和总结.这些研究成果主要体现在 Hypervisor 安全、体系架构扩展、安全隔离和权限控制等方面.最后给出了现有方案的使用范围和存在的问题,并对虚拟化安全机制的未来研究方向进行了分析和建议.上述工作比较全面地分析了虚拟化软件栈在云计算安全中的重要问题,为进一步的

研究工作奠定了基础.

在后续研究中，我们将在此基础上研究基于多层的防御系统和配套的硬件安全机制的扩展，主要针对多租户环境 Hypervisor、虚拟机和应用程序间的资源隔离和控制，通过软-硬件相结合，提出一种基于多层的资源隔离和访问控制的数据保护方案，营造一个安全的云环境.

除此之外，我们希望研究者应该将自身的研究与云提供商合作，产学结合，将方案在真实的环境中进行测试、评估，赢得云服务提供商的信服、用户的信赖.

参　考　文　献

[1] Wang X, Qi Y, Dai Y, et al. TrustOSV: Building trustworthy executing environment with commodity hardware for a safe cloud. Journal of Computers, 2014, 9(10): 2303-2314

[2] Szefer J, Lee R B. A case for hardware protection of guest VMs from compromised hypervisors in cloud computing// Proceedings of the 31st International Conference on Distributed Computing Systems Workshops. Minneapolis, USA, 2011: 248-252

[3] Zhang Y, Juels A, Reiter M K, et al. Cross-VM side channels and their use to extract private keys//Proceedings of the ACM 19th Conference on Computer and Communications Security. Raleigh, USA, 2012: 305-316

[4] Vaquero L M, Rodero-Merino L, Morán D. Locking the sky: A survey on IaaS cloud security. Computing, 2011, 91(1): 93-118

[5] Wang L, Von Laszewski G, Younge A, et al. Cloud computing: A perspective study. New Generation Computing, 2010, 28(2): 137-146

[6] Neiger G, Santoni A, Leung F, et al. Intel virtualization technology: Hardware support for efficient processor virtualization. Intel Technology Journal, 2006, 10(3): 167-177

[7] Strongin G. Trusted computing using AMD "Pacifica" and "Presidio" secure virtual machine technology. Information Security Technical Report, 2005, 10(2): 120-132

[8] Varanasi P, Heiser G. Hardware-supported virtualization on ARM//Proceedings of the 2nd Asia-Pacific Workshop on Systems. Shanghai, China, 2011: 51-55

[9] Gunawi H S, Hao M, Leesatapornwongsa T, et al. What bugs live in the cloud?: A study of 3000+Issues in Cloud Systems//Proceedings of the ACM 5th Symposium on Cloud Computing. New Seattle, USA, 2014: 1-14

[10] Perez-Botero D, Szefer J, Lee R B. Characterizing hypervisor vulnerabilities in cloud computing servers//Proceedings of the 2013 International Workshop on Security in Cloud Computing. Hangzhou, China, 2013: 3-10

[11] Yu L, Weng C, Li M, et al. Security challenges on the clone, snapshot, migration and rollback of Xen based computing environments//Proceedings of the 5th Annual ChinaGrid Conference. Guangzhou, China, 2010: 223-227

[12] AlZain M A, Soh B, Pardede E. A survey on data security issues in cloud computing: From single to multi-clouds. Journal of Software, 2013, 8(5): 1068-1078

[13] Xiao J, Lu L, Huang H, et al. Hyperprobe: Towards virtual machine extrospection//Proceedings of the 29th Large Installation System Administration Conference. Washington, USA, 2015: 1-12

[14] Fernandes D A B, Soares L F B, Gomes J V, et al. Security issues in cloud environments: A survey. International Journal of Information Security, 2014, 13(2): 113-170

[15] Cui L, Li J, Wo T, et al. HotRestore: A fast restore system for virtual machine cluster//Proceedings of the 28th Large Installation System Administration Conference. Seattle, USA, 2014: 10-25

[16] Cui L, Li B, Zhang Y, et al. HotSnap: A hot distributed snapshot system for virtual machine cluster//Proceedings of the 27th Large Installation System Administration Conference. Washington, USA, 2013: 59-74

[17] Stewin P, Bystrov I. Understanding DMA malware// Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Heraklion, Greece, 2012: 21-41

[18] Wu Z, Xu Z, Wang H. Whispers in the hyper-space: High-speed covert channel attacks in the cloud//Proceedings of the 21st USENIX Security Symposium. Bellevue, USA, 2012: 159-173

[19] Weiss M, Heinz B, Stumpf F. A cache timing attack on AES in virtualization environments//Proceedings of the International Conference on Financial Cryptography and Data Security. Bonaire, Netherlands, 2012: 314-328

[20] Yarom Y, Falkner K. Flush+Reload: A high resolution, low noise, L3 cache side-channel attack//Proceedings of the 23rd USENIX Security Symposium. San Diego, USA, 2014: 719-732

[21] Xu Y, Cui W, Peinado M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems// Proceedings of the 36th IEEE Symposium on Security and Privacy. San Jose, USA, 2015: 640-656

[22] Irazoqui G, Inci M S, Eisenbarth T, et al. Wait a minute! A fast, Cross-VM attack on AES//Proceedings of the 17th International Workshop on Recent Advances in Intrusion Detection. Gothenburg, Sweden, 2014: 299-319

[23] Irazoqui G, Eisenbarth T, Sunar B. S$A: A shared cache attack that works across cores and defies VM sandboxing—and its application to AES//Proceedings of the 36th IEEE Symposium on Security and Privacy. San Jose, USA, 2015: 591-604

[24] Parno B, Lorch J R, Douceur J R, et al. Memoir: Practical state continuity for protected modules//Proceedings of the IEEE 32nd Symposium on Security and Privacy. Oakland, USA, 2011: 379-394

[25] Petroni Jr N L, Hicks M. Automated detection of persistent kernel control-flow attacks//Proceedings of the 14th ACM Conference on Computer and Communications Security. Alexandria, USA, 2007: 103-115

[26] Riley R, Jiang X, Xu D. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing//Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection. Cambridge, USA, 2008: 1-20

[27] Wang Z, Jiang X, Cui W, et al. Countering kernel rootkits with lightweight hook protection//Proceedings of the 16th ACM Conference on Computer and Communications Security. Chicago, USA, 2009: 545-554

[28] Hua J, Sakurai K. Barrier: A lightweight hypervisor for protecting kernel integrity via memory isolation//Proceedings of the 27th Annual ACM Symposium on Applied Computing. Trento, Italy, 2012: 1470-1477

[29] Chen X, Garfinkel T, Lewis E C, et al. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems//Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems. Seattle, USA, 2008: 2-13

[30] Hofmann O S, Kim S, Dunn A M, et al. InkTag: Secure applications on an untrusted operating system//Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems. Houston, USA, 2013: 265-278

[31] McCune J M, Li Y, Qu N, et al. TrustVisor: Efficient TCB reduction and attestation//Proceedings of the 31st IEEE Symposium on Security and Privacy. Oakland, USA, 2010: 143-158

[32] Evtyushkin D, Elwell J, Ozsoy M, et al. Iso-X: A flexible architecture for hardware-managed isolated execution//Proceedings of the 47th International Symposium on Microarchitecture. Cambridge, UK, 2014: 190-202

[33] Anati I, Gueron S, Johnson S, et al. Innovative technology for CPU based attestation and sealing//Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. Tel Aviv, Israel, 2013: 97-104

[34] Hoekstra M, Lal R, Pappachan P, et al. Using innovative instructions to create trustworthy software solutions//Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. Tel Aviv, Israel, 2013: 81-88

[35] McKeen F, Alexandrovich I, Berenzon A, et al. Innovative instructions and software model for isolated execution//Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. Tel Aviv, Israel, 2013: 73-80

[36] Ren J, Qi Y, Dai Y, et al. AppSec: A safe execution environment for security sensitive applications//Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Istanbul, Turkey, 2015: 187-199

[37] Xiong H, Liu Z, Xu W, et al. Libvmi: A library for bridging the semantic gap between guest OS and VMM//Proceedings of the 12th International Conference on Computer and Information Technology. Chengdu, China, 2012: 549-556

[38] Payne B D, De Carbone M D P, Lee W. Secure and flexible monitoring of virtual machines//Proceedings of the 23rd Annual Computer Security Applications Conference. Miami Beach, USA, 2007: 385-397

[39] Lamps J, Palmer I, Sprabery R. WinWizard: Expanding Xen with a LibVMI intrusion detection tool//Proceedings of the 7th International Conference on Cloud Computing. Alaska, USA, 2014: 849-856

[40] Westphal F, Axelsson S, Neuhaus C, et al. VMI-PL: A monitoring language for virtual platforms using virtual machine introspection. Digital Investigation, 2014, 11(2): 85-94

[41] Hizver J, Chiueh T. Real-time deep virtual machine introspection and its applications//Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Salt Lake, USA, 2014: 3-14

[42] Li C, Raghunathan A, Jha N K. Secure virtual machine execution under an untrusted management OS//Proceedings of the 3rd Cloud Computing. Miami, USA, 2010: 172-179

[43] Keller E, Szefer J, Rexford J, et al. NoHype: Virtualized cloud infrastructure without the virtualization//Proceedings of the 37th Annual International Symposium on Computer Architecture. Saint Malo, France, 2010: 350-361

[44] Szefer J, Keller E, Lee R B, et al. Eliminating the hypervisor attack surface for a more secure cloud//Proceedings of the 18th ACM Conference on Computer and Communications Security. Chicago, USA, 2011: 401-412

[45] Smolyar I, Ben-Yehuda M, Tsafrir D. Securing self-virtualizing ethernet devices//Proceedings of the 24th USENIX Conference on Security Symposium. Washington, USA, 2015: 335-350

[46] Steinberg U, Kauer B. NOVA: A microhypervisor-based secure virtualization architecture//Proceedings of the 5th European Conference on Computer Systems. Paris, France, 2010: 209-222

[47] Wang Z, Wu C, Grace M, et al. Isolating commodity hosted hypervisors with hyperlock//Proceedings of the 7th ACM European Conference on Computer Systems. Bern, Switzerland, 2012: 127-140

[48] Wu C, Wang Z, Jiang X. Taming hosted hypervisors with (Mostly) deprivileged execution//Proceedings of the 20th Annual Network & Distributed System Security Symposium. San Diego, USA, 2013: 146-161

[49] Lin K J, Wang C Y. Using TPM to improve boot security at BIOS layer//Proceedings of the IEEE 2nd International Conference on Consumer Electronics. Las Vegas, USA, 2012: 376-377

[50] Zhang F, Wang J, Sun K, et al. HyperCheck: A hardware-assisted integrity monitor. Dependable and Secure Computing, 2014, 11(4): 332-344

[51] Azab A M, Ning P, Wang Z, et al. HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity// Proceedings of the 17th ACM Conference on Computer and Communications Security. Chicago, USA, 2010: 38-49

[52] Wang Z, Jiang X. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity//Proceedings of the 31st IEEE Symposium on Security and Privacy. Berkeley, USA, 2010: 380-395

[53] Gupta D, Lee S, Vrable M, et al. Difference engine: Harnessing memory redundancy in virtual machines. Communications of the ACM, 2010, 53(10): 85-93

[54] Suzaki K, Iijima K, Yagi T, et al. Memory deduplication as a threat to the guest OS//Proceedings of the 4th European Workshop on System Security. Salzburg, Austria, 2011: 1-6

[55] Zhang F, Chen J, Chen H, et al. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization//Proceedings of the 23rd ACM Symposium on Operating Systems Principles. Cascais, Portugal, 2011: 203-216

[56] Ben-Yehuda M, Day M D, Dubitzky Z, et al. The turtles project: Design and implementation of nested virtualization// Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation. Vancouver, Canada, 2010: 423-436

[57] Jin S, Huh J. Secure MMU: Architectural support for memory isolation among virtual machines//Proceedings of the 41st International Conference on Dependable Systems and Networks Workshops. Hong Kong, China, 2011: 217-222

[58] Jin S, Ahn J, Cha S, et al. Architectural support for secure virtualization under a vulnerable hypervisor//Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. Porto Alegre, Brazil, 2011: 272-283

[59] Jin S, Ahn J, Seol J, et al. H-SVM: Hardware-assisted secure virtual machines under a vulnerable hypervisor. IEEE Transactions on Computers, 2015, 64(10): 2833-2846

[60] Champagne D, Lee R B. Scalable architectural support for trusted software//Proceedings of the 16th International Symposium on High Performance Computer Architecture. Bangalore, India, 2010: 1-12

[61] Chhabra S, Rogers B, Solihin Y, et al. SecureME: A hardware-software approach to full system security//Proceedings of the 25th International Conference on Supercomputing. Tucson, Arizona, 2011: 108-119

[62] Rogers B, Chhabra S, Prvulovic M, et al. Using address independent seed encryption and bonsai merkle trees to make secure processors OS-and performance-friendly//Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture. Chicago, USA, 2007: 183-196

[63] Xia Y, Liu Y, Chen H. Architecture support for guest-transparent VM protection from untrusted hypervisor and physical attacks//Proceedings of the 19th International Symposium on High Performance Computer Architecture. Shenzhen, China, 2013: 246-257

[64] Baumann A, Peinado M, Hunt G. Shielding applications from an untrusted cloud with haven. ACM Transactions on Computer Systems, 2015, 33(3): 56-82

[65] Szefer J, Lee R B. Architectural support for hypervisor-secure virtualization//Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems. London, UK, 2012: 437-450

[66] Elwell J, Riley R, Abu-Ghazaleh N, et al. A non-inclusive memory permissions architecture for protection against cross-layer attacks//Proceedings of the 20th International Symposium on High Performance Computer Architecture. Orlando, USA, 2014: 201-212

[67] Liu F, Yarom Y, Ge Q, et al. Last-level cache side-channel attacks are practical//Proceedings of the 36th IEEE Symposium on Security and Privacy. San Jose, USA, 2015: 605-622

[68] Kim T, Peinado M, Mainar-Ruiz G. STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud//Proceedings of the 21st USENIX Security Symposium. Bellevue, USA, 2012: 189-204

[69] Varadarajan V, Ristenpart T, Swift M. Scheduler-based defenses against cross-VM side-channels//Proceedings of the 23rd USENIX Security Symposium. San Diego, USA, 2014: 687-702

ZHU Min, born in 1988, Ph. D. candidate. His current research interests include information security and privacy, and cloud computing security.

TU Bi-Bo, born in 1977, Ph. D., professor. His research interests include network and system security, operating system, and virtualization security.

MENG Dan, born in 1965, Ph. D., professor. His research interests include computer architecture, network and system security.

Background

Cloud computing using the virtualization technology dynamically organizes the physical infrastructures to build shared resources pools for providing elastic cloud services in the data center. The benefits brought by cloud computing model include but are not limited to: reduced support and hardware needs, relief of the burden for IT management, universal data access with independent geographical locations, and less environmental impact etc. Cloud customers or users lease the resources in form of virtual machines. So in this multi-tenancy mode, virtual machines belonging to different customers while running at the same server, which poses a great challenge to the security of virtualization technology. If no appropriate security solutions are proposed, this flexibly computing paradigm could become a huge failure. In recent years, there are various state-of-the-art research works that focusing on virtualization security, include hypervisor-based virtual machine protection, virtual machine isolation and access controls etc. These researches have gained lots of achievements. In this paper, we systematically summarized and analyzed these research works. In this survey, we firstly analyzed the existing security threat in virtualization software stack of cloud computing. And then we assessed the advantages and weaknesses of these research works in detail. In the last, we discussed the security threats that have not been addressed by these existing research works and gave suggestions for the future work.

Currently we focus on virtualization software stack security, including hypervisor integrity protection, virtual machine introspection and virtual machine isolation.

This work is supported by the National Key R&D Plan of China (Grant No. 2016YFB0801002). This project aims to promote the security and applicability of cloud computing, and meet the government's demand.