面向云环境的 VMM 平台安全性加固综述

周启航[1], 贾晓启[1,2], 张伟娟[1], 姜 楠[1,2]

[1] 中国科学院信息工程研究所 北京 中国 100093
[2] 中国科学院大学网络空间安全学院 北京 中国 100093

摘要 虚拟化技术作为云计算新时代下的新技术基础设施之一, 是构建新型 IT 架构的承载技术。虚拟机监视器作为虚拟化和云计算中最重要的组件, 对云平台的安全和稳定至关重要。然而, 由于庞大且逐年增长的代码量、复杂且单一的设计模式和缺乏内部隔离, 虚拟机监视器近年来不断爆出安全问题。虚拟机监视器控制着整个虚拟化平台的正常运转, 一旦虚拟机监视器受到攻击, 云平台的所有虚拟机将暴露于威胁之中。如何对虚拟机监视器进行安全性加固成为研究热点。因此, 为了增强虚拟机监视器的安全性, 本文从虚拟机监视器系统架构角度, 全面系统性分析和总结了面向云环境的虚拟机监视器安全加固技术。首先, 分析了 Hypervisor 模型、宿主模型和混合模型三种传统虚拟机监视器的架构模型和实际的虚拟机监视器软件(Xen, KVM 和 VMWare ESX Server), 并总结它们架构中存在的弊端; 其次, 对近年来国内外的虚拟机监视器加固研究成果进行归纳, 将其分为特权域安全加固、完整性保护、错误隔离强制、最小化虚拟机监视器、嵌套虚拟化加固和硬件加密保护等类别, 并比较不同方案的优缺点; 接着, 本文提出了可信基大小、访问控制、错误隔离和性能与部署难度这四个评估维度来评价虚拟机监视器的架构安全性; 最后, 本文对下一步的虚拟机监视器安全性加固进行研究展望。

关键词 VMM 安全加固; 虚拟化安全; 可信基; 架构安全
中图法分类号 TP319 DOI 号 10.19363/J.cnki.cn10-1380/tn.2025.01.12

A Survey of VMM Security Reinforcement on Virtualization Platform

ZHOU Qihang[1], JIA Xiaoqi[1,2], ZHANG Weijuan[1], JIANG Nan[1,2]

[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100093, China

Abstract As one of the new technological infrastructure in the new era of cloud computing, virtualization technology is the bearer technology for building a new IT architecture. Virtual machine monitor, the most important component in virtualization and cloud computing, is critical to the integrity, security and stability of cloud platform. However, the virtual machine monitor has been exposed to many security problems in recent years due to a large number of codes, a complex and monolithic design pattern and a lack of internal isolation. The virtual machine monitor controls the normal operation of the entire virtualization platform. Once the virtual machine monitor is compromised, all virtual machines on the cloud platform will be exposed to threats. How to reinforce the security of virtual machine monitor has become a research hotspot. Therefore, in order to better enhance the security of the virtual machine monitor, in this paper, we systematically analyze and summarize the security reinforcement technology of virtual machine monitor from the perspective of architecture security. Firstly, we introduce and analyze three traditional architecture models of virtual machine monitors: hypervisor model, hosted model and hybrid model, as well as three actual virtual machine monitor softwares (Xen, KVM and VMWare ESX Server), and summarize the potential safety hazard of their architectures. Secondly, we survey the domestic and foreign reinforcement researches of the virtual machine monitor in recent years, sum these projects up into privileged domain security hardening, integrity protection, error isolation enforcement, minimizing virtual machine monitor, nested virtualization reinforcement and hardware encryption protection, and compare the advantages and disadvantages of different researches. Thirdly, we institute four dimensions including trusted computing base size, access control, error isolation and performance and deployment difficulty to evaluate the design of virtual machine monitor. Finally, we discuss the challenges and looks forward to the next step of virtual machine monitor security reinforcement.

Key words VMM Security reinforcement; virtualization security; TCB; architecture security

通讯作者: 贾晓启, 博士, 研究员, Email: jiaxiaoqi@iie.ac.cn。
本课题得到中国科学院网络测评技术重点实验室资助项目、网络安全防护技术北京市重点实验室资助项目、北京市科学技术委员会项目(No. Z191100007119010)、中国科学院国防科技重点实验室基金项目(No. CXJJ-20S022)资助。
收稿日期: 2020-03-19; 修改日期: 2020-05-31; 定稿日期: 2022-12-30

1 引言

1956 年 6 月, 牛津大学计算机教授 Cristopher Strachey 第一次提出虚拟化概念[1]。虚拟化代表资源的逻辑抽象, 其本身不受物理限制约束。虚拟化技术泛指一切可以被虚拟的计算机实体, 包括网络虚拟化、系统虚拟化以及存储虚拟化等。本文主要关注系统虚拟化。历史上第一台虚拟机于 1964 年 4 月 7 日诞生, 是 IBM 生产的 IBM System/360 Model 40 VM[2]。自 2006 年亚马逊推出 Amazon Web Service 以来, 虚拟化技术作为云计算基石为云平台的稳定性提供了重要支撑。虚拟化技术中最为核心的软件是虚拟机监视器(Virtual Machine Monitor, VMM), 用来管理硬件资源, 为虚拟机提供执行环境和服务例程。然而, VMM 由于拥有庞大的软件栈和最高特权级, 具有巨大的攻击面。一旦 VMM 受到攻击, 整个云平台都暴露于威胁之中。文献[3]中指出, 随着 VMM 的代码量和复杂性增加, VMM 曝光的漏洞数量也逐年增加。以 Xen 和 KVM 为例, 从 2012 年到 2020 年 7 月, Xen 被公布的漏洞数量增长到 328 个[4], KVM 的漏洞数量增长了 122 个[5](如图 1 所示)。

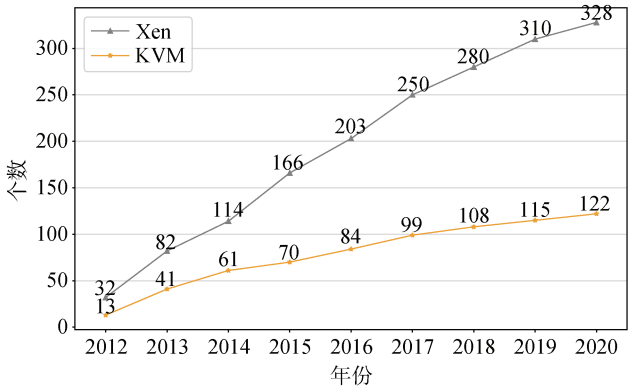

图 1 Xen 和 KVM 漏洞数量趋势

Figure 1 The increasing trend of the number of vulnerabilities in Xen and KVM

本文将 VMM 漏洞所导致的安全问题, 归纳为敏感信息泄漏、特权逃逸、拒绝服务攻击三种。敏感信息泄漏指由于系统使用或者配置错误, 导致攻击者获取到权限以外的信息, 如 CVE-2017-17045[6]。特权逃逸指攻击者可以通过远程代码执行等手段, 获得虚拟机操作系统或 VMM 的权限, 如 CVE-2017-10918[7]。拒绝服务攻击指攻击者利用 VMM 的漏洞或缺陷, 使 VMM 或者虚拟机出现异常波动, 导致虚拟机甚至整个虚拟化平台宕机。由于传统 VMM 的设计模式中, VMM 拥有最高权限、单一而复杂的高耦合设计以及所有虚拟机共享 VMM, 导致当 VMM 出现安全问题时, 整个虚拟化平台的所有虚拟机都会受到牵连。因此虚拟化安全成为业界重点研究方向, 而围绕着 VMM 架构安全, 对 VMM 进行安全性加固成为近年来的虚拟化安全研究热点。

传统 VMM 平台的安全性, 除受到 VMM 影响外, 还会受到特权域的影响, 如 Xen 的 Dom0、KVM 的宿主机操作系统等。因此, 与现有综述不同的是, 本文将面向云环境的 VMM 平台安全性加固分为 VMM 特权域安全性加固和 VMM 安全性加固。其中特权域安全加固旨在限制和隔离特权操作系统的功能权限。VMM 安全性加固分为基于完整性保护的 VMM 安全性加固、基于错误隔离的 VMM 安全性加固、基于最小化 VMM 的 VMM 安全性加固、基于嵌套虚拟化的 VMM 安全性加固和基于加密的安全性加固。本文以 Xen、KVM 和 VMWare ESX Server 为例, 首先对传统的 VMM 设计模型进行介绍和优缺点分析(第 2 章), 然后分别介绍和分析 VMM 特权域安全性加固设计(第 3 章)和 VMM 安全性加固设计(第 4 章)。最后, 本文提出了 VMM 平台安全性加固的评估维度, 并且对面向云环境的 VMM 平台安全性加固进行了研究展望(第 5 章)。

2 研究背景

2.1 VMM 架构分类

根据架构实现方式, VMM 可以分为 hypervisor 模型、宿主模型和混合模型三类[3](如图 2 所示)。Hypervisor 模型可以被看作一个专门针对虚拟化的完整操作系统, 所有的物理资源均由 VMM 管理。VMM 直接运行在硬件上, 既向下管理所有物理资源, 又需要向上提供虚拟机操作系统运行的虚拟机环境。在安全方面, 虚拟机的安全只依赖于 VMM 的安全。宿主模型中, 物理资源由宿主机操作系统负责管理和分配。宿主机操作系统本身不具备虚拟化功能, 需要 VMM 作为宿主机操作系统独立的内核模块进行。VMM 对虚拟机的部分功能支持需要调用宿主机操作系统的服务来进行。宿主模型通常将虚拟机作为宿主机操作系统的一个独立进程进行调度, 虚拟机的安全需要宿主机操作系统和 VMM 共同负责。混合模型是以上两种模型的结合。混合模型中, VMM 拥有物理资源所有权限的同时, 将一部分涉及 I/O 设备的控制权交由一个运行在特权虚拟机中的操作系统来负责。在安全方面, 混合模型的安全性依赖于特权虚拟机的权限控制和 VMM。随着虚拟化不断发展, VMM 实现过程中对各模型的界限划分逐渐模

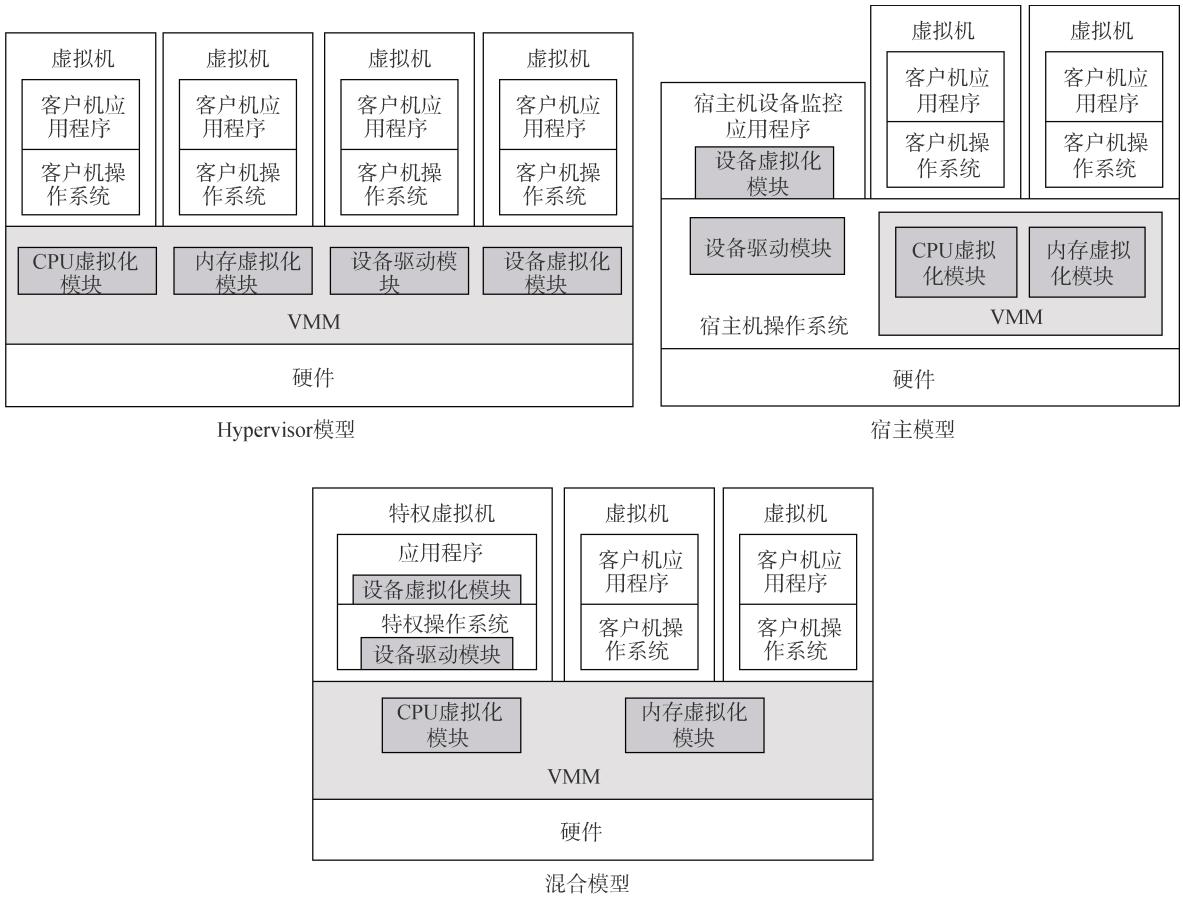

图 2　VMM 架构模型

Figure 2　The architecture model of VMM

糊。如今业界也将 VMM 模型分为 Type I 型和 Type II 型。Type I 型指拥有独立且完整内核的 VMM, 又称为裸金属型, Type II 型指作为内核模块存在, 嵌入到 Host OS 的 VMM, 又称为宿主型。

2.2　VMM 软件

VMware 是 x86 架构下, 主流虚拟化软件厂商之一。VMware 的虚拟化软件包括 ESX Server、VMware Server、VMware Workstation 和 VMware Fusion。ESX Server 基于 Hypervisor 模型, 可以运行 Linux、Windows 等主流的操作系统。VMware Server 面向服务器端, 采用宿主模型, 将 VMM 模块嵌入到 Windows 或者 Linux 操作系统中。VMware Workstation 同 VMware Server 相似, 但主要针对桌面操作系统。VMware Workstation 可以运行在 Windows 和 Linux 操作系统中。VMware Workstation 在 Mac OS X 系统中的版本为 VMware Fusion。Microsoft 的系统虚拟化产品主要包括 Virtual PC、Virtual Server 和 Hyper-V。Virtual PC 是基于宿主模型的 VMM 产品, 宿主操作系统是 Windows, 主要面向桌面系统的虚拟化产品。Virtual Server 与 Virtual PC 一样, 采用宿主模型, 但主要针对服务器系统。Hyper-V 属于裸金属虚拟化产品(Bare-Metal Virtualization), 基于混合模型, 同时部署一个特权操作系统作为特权域。Oracle 旗下的 VirtualBox 是一款宿主模型的开源虚拟化产品, 可运行在 Windows、Linux、OS X 或 Solaris 系统中。Xen 是一款基于 GPL 授权的开源虚拟化产品。Xen 基于混合模型, 直接运行在物理硬件上, 有一个特殊的操作系统作为特权域。KVM 同样也是一款基于 GPL 授权的开源软件, 采用宿主模型, VMM 被嵌套在 Linux 的内核中, 利用 Qemu 实现设备虚拟化(见表 1)。

本节将以 Xen、KVM 和 VMWare ESX Server 为例, 来介绍 VMM 的系统架构。其中, Xen 属于混合模型或 Type I 型, KVM 属于宿主模型或 Type II 型(但随着 Linux 中虚拟化支持功能增强, 也有说 KVM 是 Hypervisor 模型), VMWare ESX Server 属于 Hypervisor 型或 Type I 型。

2.2.1　Xen 虚拟化架构概述

Xen 最先发表于第 19 届 SOSP 大会[8], 旨在通过虚拟机操作系统和虚拟化层的协同设计, 提供一个

表 1 VMM 软件

Table 1 VMM softwares

VMM	VMM 架构类型	运行环境
VMware ESX Server	Hypervisor 模型或 Type I 型	直接运行在硬件平台上
VMware Server	宿主模型或 Type II 型	运行在 Windows 或 Linux 中
VMware Workstation	宿主模型或 Type II 型	运行在 Windows 或 Linux 中
VMware Fusion	宿主模型或 Type II 型	运行在 Mac OS X 中
Microsoft Virtual PC	宿主模型或 Type II 型	运行在 Windows 中
Microsoft Virtual Server	宿主模型或 Type II 型	运行在 Windows 中
Hyper-V	混合模型或 Type I 型	直接运行在硬件平台上
Oracle VM VirtualBox	宿主模型或 Type II 型	运行在 Windows、Linux、OS X 或 Solaris 中
Xen	混合模型或 Type I 型	直接运行在硬件平台上
KVM	宿主模型或 Type II 型	运行在 Linux 中

近似于原物理系统的计算机系统。Xen 最先只支持类虚拟化虚拟机, 随着 Intel VT[9]和 AMD-V[10]硬件辅助虚拟化的技术的提出, Xen 可以同时支持全虚拟化虚拟机(HVM)和类虚拟化虚拟机(PV)。Xen 位于硬件和虚拟机操作系统之间, 采用混合模型, 为虚拟机操作系统提供虚拟化硬件支持。Xen 上有一个特殊的类虚拟化操作系统作为特权操作系统或特权域, 称为 Domain0(Dom0), 其他虚拟机称为 DomainU (DomU)。Dom0 用来辅助 Xen 对其他虚拟机操作系统进行管理, 包括虚拟机的生命周期管理和虚拟机设备 IO 虚拟化支持等。Xen 作为 hypervisor 向虚拟机提供了虚拟处理器、虚拟内存、事件通道等资源。图 3 展示了 Xen 的体系结构。作为混合模型, Xen 设计了一个完整的用于支持虚拟化的内核, 实现了包括虚拟机调度、通信、内存管理、VCPU 管理等在内的所有资源管理和控制流管理的功能。这种设计使得 Xen 的代码具有高度的整合性和易迁移性。Xen 将设备驱动和设备虚拟化模块功能交由 Dom0 负责, 通过利用 Dom0 原生的设备驱动模块, 一定程度上减少虚拟化开发的复杂度。同时, Xen 独特的类虚拟化虚拟机(PV)支持, 使虚拟机的运行性能接近于物理机。

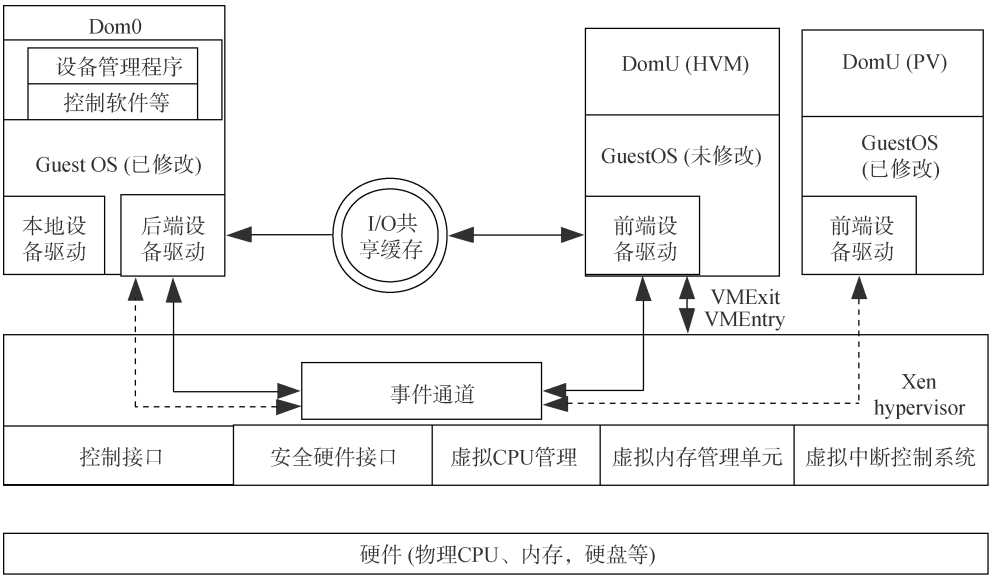

图 3 Xen 体系结构

Figure 3 The architecture of Xen

2.2.2 KVM 虚拟化架构概述

Kernel-based Virtual Machine(KVM)[11]由以色列公司 Qumranet 开发, 2007 年被集成到 Linux 2.6.20 内核中。KVM 是一款基于 Linux 操作系统的虚拟机软件, 且需要硬件虚拟化支持。KVM 的架构设计采用宿主模型, 即 KVM 作为一个模块整合到 Linux 内核中。KVM 模块负责处理器虚拟化和内存虚拟化功能, 设备虚拟化功能由 Linux 的设备驱动和 Qemu 的设备管理共同完成。随着 virtio 的提出, KVM 也支持基于前后驱动的半虚拟化 IO 操作。KVM 虚拟机的生命周期管理由 Qemu 维护, 虚拟机作为 Linux 中一个 Qemu 进程来进行调度管理, 虚拟机的每一个 VCPU 作为进程中一个线程实现。图 4 展示了 KVM 的体系结构。KVM 作为 Linux 的一个模块, 相较于 Xen 更加轻便且易于调试, 因其与 Linux 完美结合, 可以继承 Linux 的大部分功能, 作为 Linux 进程来进行管理的 KVM 虚拟机也可以直接使用 Linux 的一些安全策略来进行访问控制和资源管理。

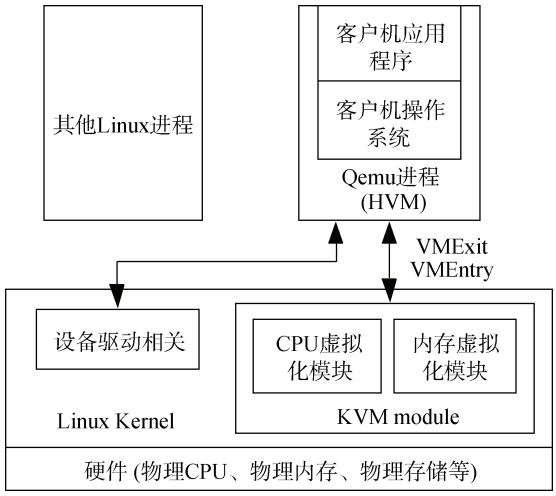

图 4 KVM 体系结构

Figure 4 The architecture of KVM

2.2.3 VMWare ESX Server 架构概述

VMWare ESX Server[12]是面向企业级用户服务器整合服务的数据中心级虚拟化平台。VMWare ESX Server 可以根据平台管理员的配置动态地为虚拟机分配 CPU、内存、磁盘和网络等资源, 从而提供大型机级别的容量利用率和服务器资源控制。图 5 展示了 VMWare ESX Server 的体系结构。VMWare ESX Server 的主要组件包括 VMM、资源管理器、服务控制台和硬件接口等。VMM 主要用于为虚拟机提供虚拟化硬件环境和硬件设备, 每一个虚拟机都有自己的一套虚拟化抽象视图, 并与运行在同一物理系统上的其他虚拟机隔离。资源管理器对计算机物理资源进行划分, 并为每个虚拟机分配其所需资源。硬件接口用于提供特殊硬件服务, 并隐藏该服务与系统其他部分的硬件差异。硬件接口组件中主要包括设备驱动程序和 VMFS。服务控制台用于启动系统、初始化资源管理器和 VMM。启动成功后, 服务控制台将控制权移交于这些组件。此外, 服务控制台还运行一些用于部署管理员函数的应用程序。

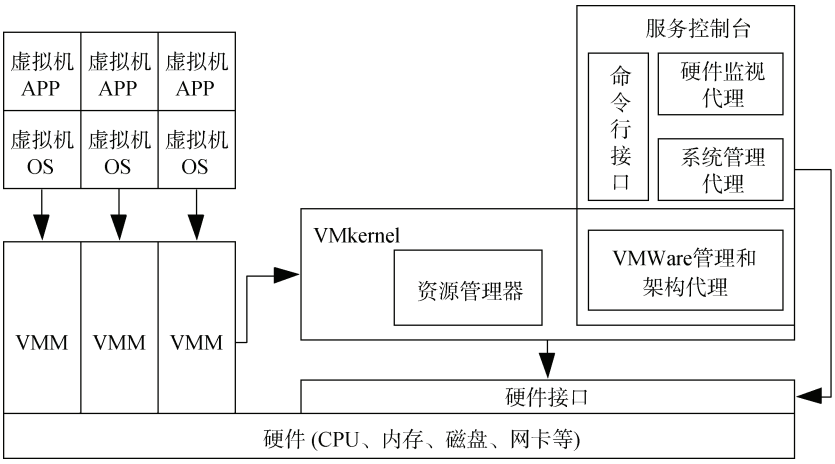

图 5 VMWare ESX Server 体系结构

Figure 5 The architecture of VMWare ESX Server

2.3 VMM 架构安全隐患

传统 VMM 软件因完备的虚拟化支持和良好的可迁移性而受到追捧。然而随着虚拟化技术不断发展, 虚拟化平台功能日益强大, VMM 架构的安全性也受到了人们关注。我们将 VMM 的架构安全隐患归纳为三个原因。

(1) 庞大的可信基。随着 VMM 所提供的服务越来越多, VMM 的代码量也逐渐增大。逐渐增多的代码量导致了 VMM 中漏洞数量增多。据统计, 从 2012 年到 2020 年 7 月, Xen 曝光的漏洞数量增加了 328 个, KVM 增加了 122 个。此外, 由于 VMM 的部分功能需要宿主机操作系统的支持, 因此一些特定的操作系统(如 Xen 的 Dom0、包含 KVM 模块的 Linux)也包括在虚拟化平台的可信基中(如表 2)。

表 2 Xen 与 KVM 的(软件层)可信基

Table 2 The TCBs of Xen and KVM

系统	架构类型	可信基
Xen	混合型	Xen hypervisor+Dom0
KVM	宿主型	Kvm module+Linux kernel+Qemu

(2) 虚拟机与 VMM 缺乏访问控制。 传统虚拟化平台设计中, 虚拟机使用敏感指令将会触发 VMExit 直接陷入到 VMM 中。控制权移交到 VMM 后, VMM 根据陷入信息进行相应的处理。整个过程中, VMM 不会对虚拟机进行访问控制和信息的正确性判断。因此, 虚拟机中不恰当的操作可能对 VMM 的安全性和稳定性产生影响。反之亦然, VMM 处理完毕后, 通过 VMEntry 直接将控制权移交回虚拟机,

同样缺乏相应的控制检查。

(3) 高耦合设计。虚拟化平台中, 所有虚拟机共享一个 VMM, 如果某一个虚拟机的不恰当操作导致 VMM 软件崩溃, 其他虚拟机均会受到影响。此外, VMM 作为一个庞大的软件系统, 其内部包含着多种功能模块, 然而严重安全漏洞多位于特定模块中, 如内存虚拟化模块、IO 模块等。高危模块和普通模块之间缺乏有效隔离, 也是导致 VMM 安全问题的原因之一。

3 VMM 特权域安全性加固

特权域指拥有较高权限的特权操作系统。特权操作系统承担着设备虚拟化、虚拟机生命周期管理以及虚拟化平台维护等任务。受到攻击的特权域不仅可以通过调用 VMM 接口对 VMM 进行攻击, 还可以通过直接内存映射等手段破坏虚拟机的机密性和完整性。特权域安全性加固主要以分解并隔离特权操作系统的功能为主。

Disaggregated Xen[13]是针对 Xen 的特权系统 Dom0 的重构方案。Disaggregated Xen 移除了 Dom0 中所有涉及创建虚拟机的代码, 把这些代码隔离到一个单独的虚拟机中, 称为 DomB(Domain Builder)。DomB 是一个名为 MiniOS 的类虚拟化操作系统, 负责加载虚拟机磁盘文件、为虚拟机分配物理内存映射、复制内核文件到新创建的虚拟机地址空间、初始化虚拟机页表以及启动虚拟机。MiniOS 没有物理设备驱动、文件系统, 无法进行相关 IO 操作。Disaggregated Xen 使用文件系统调用的方式, 通过调用 Dom0 中的 IO 服务来完成 DomB 的 IO 操作。Dom0 中部署一个名为 vfsback 的前端 IO 服务。Vfsback 接收和处理来自 DomB 的 IO 请求, 并将结果返回给 DomB。由于没有 TCP/IP 协议栈, Dom0 和 DomB 之间无法通过网络通信。Disaggregated Xen 借用微内核通信的 IPC(inter-process communication)思想[14-16], 部署了一套基于 IVMC(inter-VM communication)通信机制, 使用 IDL(Interface Definition Language)来定义通信接口[17], 通过 DomB 的虚拟机 ID 来限制接口通信。Disaggregated Xen 通过移除 privcmd 驱动, 取消了 Dom0 所有的用户空间特权操作。此外, Disaggregated Xen 添加一个名为 gntdev 的设备用于授权表映射管理。Disaggregated Xen 将 Dom0 的用户态空间移出虚拟化平台可信基, 减少虚拟化平台特权操作系统的攻击面。由于某些特定配置的设备可以进行直接内存访问, Disaggregated Xen 并未将 Dom0 的内核态空间移出可信基。因此无法对 Dom0 内核的安全性提供保护。

Xoar[18]在 Disaggregated Xen 的基础上, 进一步分解 Dom0, 并且将 Dom0 的用户态空间和内核态空间完全移出可信基。Xoar 将 Dom0 分解成 7 种不同类型的服务虚拟机, 分别是: PCIBack、Bootstrapper、Toolstack、Builder、BlkBack、NetBack 和 XenStore。每种组件只负责单一职能。PCIBack 负责虚拟化 PCI 设备相关功能, 包括初始化硬件和 PCI 总线、设备透传以及虚拟化 PCI 配置; Bootstrapper 用于启动服务虚拟机; Toolstack 用户普通虚拟机的生命周期管理; Builder 用于创建虚拟机; BlkBack 负责为普通虚拟机提供块设备虚拟化功能; NetBack 负责为普通虚拟机提供网络设备虚拟化功能; XenStore 作为最重要的组件, 进一步划分为 XenStore-Logic 和 XenStore-state 两个部分, 提供系统配置注册表管理和监控。其中, Bootstrapper、Builder、XenStore 和 PCIBack 由所有虚拟机共享, 其他三个服务虚拟机由每个普通虚拟机独立拥有。Xoar 采用 microreboots[19]和“crash-only software”的工作, 利用快照和回滚技术对服务虚拟机进行错误恢复。Xoar 为服务虚拟机各分配一块“recovery box”[20], 一旦调用快照机制, 就将服务虚拟机被修改的内存状态存入“recovery box”, 虚拟机回滚后, 将相关内存恢复到服务虚拟机中。Xoar 更细致地拆分了 Dom0 的功能并且为每个虚拟机分配单独的服务虚拟机, 有效地减少攻击面和增强错误隔离。然而, Xoar 更细致地分解 Dom0 导致部署复杂度增加, 频繁地组件间通信和上下文切换增加了系统的性能开销。

SSC[21]的出现出于两个原因, 第一个原因同 Disaggregated Xen 和 Xoar, 即 Dom0 拥有较高的权限和较为复杂的代码; 第二个原因是因为虚拟机用户无法灵活地控制虚拟机的安全服务。SSC 将 Dom0 分成一个 domain-building domain(DomB)、一个系统 Domain(Sdom0)和一组面向用户的 Meta-Domains。面向客户的 Meta-Domains 包含每个用户的特权虚拟机(Udom0s)、用户虚拟机(UdomUs)、服务虚拟机(service domains, SDs)和互相信任的服务虚拟机(mutually-trusted service domains, MTSDs)。SSC 利用 vTPM[22]技术可信启动 Xen、Sdom0 和 DomB。DomB 内部集成基于 vTPM 协议的安全度量模块, 用于可信创建虚拟机并返回用户一个用于表示用户 Meta-domains 的标识符。Sdom0 用于根据用户的请求开启或关闭 UdomU 和运行设备驱动。Udom0s 用于管理和监控用户虚拟机。SDs 拥有部分 Udom0 的特权操作。MTSDs 作为一个用户和云服务提供商共

同信任的区间, 存放着双方共同制定的虚拟机安全策略。各组件之间使用 SSL 机制通信。SSC 在移除 Dom0 特权的同时, 有效地解决了用户无法灵活控制虚拟机安全策略的问题。

虚拟化平台中, 特权域的安全性加固旨在分解和隔离特权操作系统的功能, 减少虚拟化平台管理员的特权和特权操作系统的攻击面。Disaggregated Xen 将 Dom0 的用户态空间从可信基中移除, 减少和边界化虚拟化平台可信基, 增强特权操作系统的安全性。然而, 由于未对 Dom0 的内核进行防御, Disaggregated Xen 无法抵制来自 Dom0 内核态空间的威胁, 也无法控制 Dom0 中设备 DMA 问题。Xoar 进一步将 Dom0 分解成 7 个服务类型的虚拟机, 将 Dom0 的内核态和用户态空间共同从虚拟化平台可信基中移除, 缓解了 Dom0 导致的安全问题。尽管 Xoar 有效防御了来自 Dom0 的安全威胁, 但其碎片化设计导致了频繁地服务虚拟机之间通信问题, 实现起来更加复杂。SSC 在分解 Dom0 的同时, 重新在特权操作系统中创建了一块用于虚拟机用户和虚拟化平台共同制定安全策略的组件, 使用户对虚拟机的管理更加灵活。此外, SSC 采用基于硬件 TPM 的可信启动认证, 更加安全, 但特殊硬件的选择不利于虚拟化平台的迁移和快速部署。

4 VMM 安全性加固

第 3 章介绍了 VMM 特权操作系统的安全性加固方案, VMM 特权域安全性加固没有解决 VMM 的安全隐患, 它只是对与 VMM 交互的特权软件进行了接口限制, 避免 VMM 功能被滥用。本章将介绍针对 VMM 的安全性加固方法, 旨在构建更安全的虚拟化平台 VMM 软件。本章将 VMM 加固分为基于完整性保护的 VMM 加固、基于错误隔离和恢复的 VMM 加固、基于最小化 TCB 的 VMM 加固、基于嵌套虚拟化的 VM 重构和基于加密的 VMM 加固五类。

表 3 特权域安全性加固对比

Table 3 Comparison of security hardening of privileged domains

系统名称	VMM	特权域组件解耦	组件间通信方法	系统效果	不足
Disaggregated Xen[13]	Xen	Dom0、DomB	基于 IDL 的 IVMC	将 Dom0 的用户态空间移出可信基, 减少可信基	DomB 功能缺失
Xoar[18]	Xen	Bootstrapper、PCIBack、ToolStack、XenStore-Logic、XenStore-State、Builder、BlkBack、NetBack	IVC+接口复用	将 Dom0 的用户态和内核态空间移出可信基, 减少可信基	组件间通信影响平台性能
SSC[21]	Xen	SDom0、DomB、UDom0、SD、MRSD	IVC+SSL 信道	将 Dom0 分解为 admin 和 users 两个相对立且不信任的可信基, 用户可以灵活地控制虚拟机安全策略	组件间通信影响平台性能

4.1 基于完整性保护的 VMM 安全性加固

基于完整性保护的 VMM 加固旨在对 VMM 的完整性进行检测或控制, 以减缓或及时发现 VMM 中的威胁。其中包括被动型完整性保护和主动型完整性保护。

4.1.1 基于被动型完整性保护的 VMM 安全性加固

被动型完整性保护主要指在 VMM 运行过程中, 对 VMM 进行内存或可执行文件的完整性度量和认证, 有效地检测 VMM 被恶意修改的内存。其中包括内存快照完整性度量和事件触发完整性度量。

(1) 内存快照完整性度量

内存快照完整性度量指通过间接性地获取和度量 VMM 内存快照, 以检测 VMM 是否被破坏。HyperGuard[23]和 HyperCheck[24]基于内存快照的方式, 依赖 x86 架构的 SMM(System Management Mode), 对 VMM 进行完整性度量。SMM 是 x86 架构下, 用来处理系统管理操作的模式, 用来管理和监控不同的系统资源、控制硬件以及运行特定的代码。当触发特定的软件或硬件事件后, CPU 会接受到一个 SMI, 然后进入 SMM 模式。SMM 是一个独立且被保护的执行环境, 不受软件系统控制。在 SMM 中, CPU 会运行在一个名为 SMRAM 的内存空间里, 并且所有相关的运行代码也都存储在该内存中。SMRAM 可以通过内存控制器的 D_LCK 位进行锁定, 一旦锁定 SMRAM, 任何特权软件都不能干涉 SMRAM 中的程序运行。HyperCheck 是虚拟化平台额外添加一个针对 VMM 静态数据结构完整性度量的程序, 利用 SMM 特性, 运行在 SMM 模式中, 以检测针对 VMM 静态数据结构的攻击。HyperCheck 通过网络设备间隔性获取 VMM 的物理内存快照, 并通过一个系统分析程序对内存进行完整性检验。此外 HyperCheck 还定期获取 CPU 寄存器状态以检查 CPU 寄存器的安全性。HyperGuard 采用和 HyperCheck 相似的机制完成完整性检测。HyperGuard 和 HyperCheck 的完整性

检测程序都需要 VMM 主动触发, 恶意的攻击者可以进行擦除攻击(scrubbing attack), 在检测程序触发之前擦除自己的行为。

HyperSentry[25]同样使用 SMM 机制, 进行内存快照完整性度量。与 HyperGuard 和 HyperCheck 不同, HyperSentry 采用 IPMI(Intelligent Platform Management Interface)[26]信道, 以便隐蔽地触发监测程序。IPMI 使一个面向服务器平台, 可以直接部署在硬件和固件上的管理接口。IPMI 的管理函数独立于主处理器、BIOS 和特权软件, 可以在不需要 VMM 介入的情况下, 隐秘地切换到 SMM 中的检测程序。IPMI 依赖于一个嵌入在主板上的微控制器 BMC (Baseboard Management Controller)来管理接口。所有对于 IPMI 的远程访问都需要进行认证。HyperSentry 利用 IPMI 机制, 将监测程序存放在 SMRAM 的同时, 隐蔽地进行周期性内存快照检测。HyperSentry 可以有效地防御擦洗攻击。然而, 利用 SMM 进行完整性度量需要依赖 SMM 处理程序的安全性, 已有研究表明, SMM 处理程序也并非安全。

使用基于协处理器的内存快照完整度量可以有效避免上述问题。Copilot[27]使用名为 EBSA(Intel StrongARM SA-110/21285 evaluation board)[28]的 PCI 设备对系统运行时内存快照进行完整性度量。尽管 Copilot 主要针对操作系统内核完整性度量, 但这种方法也基于对 VMM 的内存度量。EBSA 模式下, 可以通过禁止接收来自主处理器的所有输入, 确保 EBSA 模式中的程序不受主处理器上运行的所有程序影响。通过对内核指令和跳转指针的监控, Copliot 确保了内核的完整性。然而由于存在语义鸿沟, EBSA 中的程序无法检测 CPU 状态。此外, 攻击[29]可以有效地绕过 Copilot 检测机制。

内存快照完整性度量可以有效地检测 VMM 内存恶意篡改行为, 但是由于内存快照周期性地获取 VMM 内存, 因此无法防止周期间隔中对 VMM 内存的破坏攻击, 如瞬时攻击[30]。

(2) 事件触发完整性度量

事件触发完整性度量指基于事件驱动对 VMM 内存进行完整性度量。当触发事件发生时, 检测程序立刻进行完整性度量, 不受时间限制, 可以有效地防止瞬时攻击。

Vigilare[31]是一套利用 ARM 的 SoC 硬件安全机制, 监听主机系统流经总线的所有流量的事件监听完整性保护机制。Vigilare 运行在 ARM SoC 中的 Linux 里, 不会受到主机系统的任何影响。Vigilare 分为两个功能组件: 一个组件是用于收集总线流量的硬件组件, snooper; 另一个是用于分析流量和完整性度量的迷你计算机系统 verifier。Vigilare 的所有操作均不依赖主机系统, 因此可以抵御来自主机系统的潜在攻击。通过独立地度量主机系统的流量信息, Vigilare 确保主机系统的完整性。

MGUARD[32]利用 FB-DIMM 硬件技术[33], 对 DRAM 中内核相关的所有关键数据结构进行度量, 以确保内核数据的完整性。当内核被加载到内存中后, 任何对内核敏感数据结构所在的内存页进行的修改, 都被 MGUARD 视为是恶意操作。MGUARD 捕获这些 DRAM 页面修改事件, 利用 FB-DIMM 的串口, 透明地将页面传到数据分析程序中。MGUARD 所有安全性操作都依赖硬件机制, 因此可避免主机系统干扰。

与 Vigilare 类似, ED-minitor[34]也通过监听 VMM 所有的事件流量, 对 VMM 进行完整性检测。但 ED-monitor 将自身代码与 VMM 放入同一地址空间, 并在 VMM 中加入钩子函数[35]来进行事件触发。ED-monitor 利用地址随机化(address space randomization, ASR)和指令特权限制(instrumentation-based privilege restriction, IPR)来确保自身安全。ASR 用于随机化 ED-monitor 的代码和数据内存加载位置, IPR 用于限制 VMM 执行特权执行, 确保 VMM 所有的特权指令执行必须经过 ED-monitor。

事件触发完整性度量可以有效地抵御瞬时攻击, 但其针对特定事件的触发和复杂的设计逻辑, 不易部署和实现, 缺乏灵活性。

4.1.2 基于主动型完整性保护的 VMM 安全性加固

被动型完整性保护虽然可以有效地检测恶意篡改 VMM 的行为, 但需要在恶意攻击发生之后才可以检测到, 无法对恶意攻击进行预防。本节将对主动型完整性保护进行介绍。主动型完整性保护旨在通过对 VMM 的控制流进行强制限制, 有效预防针对 VMM 的恶意攻击。

HyperSafe[36]将 CFI 技术[37]与 VMM 结合, 在 VMM 运行期间对其进行运行时控制流完整性保护。HyperSafe 对 VMM 的控制流完整性保护包括静态控制流保护和动态控制流保护。HyperSafe 通过不可绕过的内存锁和受限制的指针索引技术来确保 VMM 控制流完整性。不可绕过的内存锁基于 x86 架构的内存安全机制, 对页表进行 $W \oplus X$ 写保护。HyperSafe 利用 CR0 的写保护位对页表进行更新控制。不可绕过的内存锁主要用于保护静态控制流完整性, 即 VMM 的数据和代码安全。受限制的指针索引技术通过控制流图、目标表和指针索引来实现动态控制流

地访问限制, 确保了动态控制流的执行路径安全。控制流图通过静态分析程序的执行路径产生。目标表包含了所有间接跳转的合法目标位置, 所有的指针跳转都需要通过指针索引来完成。控制流完整性强制策略可以有效地预防潜在的恶意攻击, 但每次跳转都需要进行控制流判断, 造成了较大的性能开销。

4.2 基于错误隔离的 VMM 安全性加固

如 2.4 小节所分析, 传统的虚拟化平台中, VMM 设计具有高耦合性, 所有的虚拟机共享一个 VMM, 任何对 VMM 的恶意攻击或错误配置, 将会导致整个虚拟化平台中所有的虚拟机受到牵连。此外, 由于 VMM 运行在最高特权级, 当 VMM 错误崩溃时, 整个平台需要重新启动, 严重影响了虚拟化平台的稳定性和安全性。基于错误隔离的 VMM 安全性加固可以有效地解决这些问题。

HyperLock[38]提出一种 hypervisor shadowing 技术, 有效地为每个虚拟机创建一个影子 hypervisor, 确保每个 hypervisor 只能影响其对应的虚拟机。具体来说, 影子 hypervisor 为每个虚拟机提供一个影子副本。系统中只有一个物理副本, 影子副本是物理副本的一个拷贝, 用来服务隔离的 hypervisor 实例。因为所有的影子副本共享静态 hypervisor 的代码, HyperLock 利用重复删除数据技术[39]维持一个物理副本, 避免大量的内存开销。此外, HyperLock 通过增强 W⊕X 机制、相同指令对齐原则确保 HyperLock 和宿主机操作系统彼此隔离。HyperLock 有效地隔离 VMM 错误的影响范围, 并且减少了 VMM 可信基。

DeHype[40]认为在 HyperLock 的设计中, KVM 依然运行在最高特权级, 且需要复杂的工作来避免特权代码误用。DeHype 系统按照最小特权原则, 将 KVM 大部分代码从 Linux 内核态移动到用户态, 并且为每个虚拟机生成独立的用户态 KVM 实例。DeHype 利用依赖解耦技术打破了 KVM(宿主型 VMM)和宿主机操作系统之间的依赖性, 将 KVM 解耦为一个用于执行 KVM 特权代码的内核扩展, 名为 HypeLet, 和多个降权后的用户态 KVM 实例。DeHype 将降权的 KVM 代码整合到用户态的 Qemu 进程中。当 Qemu 发送 IOCTL 请求到 KVM 时, 用户态 KVM 接受请求并进行处理。DeHype 使用 memory rebasing 技术将内存虚拟化的功能交由用户态 KVM 处理。用户态 KVM 被整合到 Qemu 进程中, 所有虚拟机都拥有功能完整且相互隔离的 KVM 实例, 确保任何一个 KVM 实例的错误只会影响到对应的虚拟机。由于 KVM 实例被降权到用户态, 通过系统调用与 HyperLet 进行交互, 导致了较大的性能开销。

SecFortress[41]在 HyperLock 和 DeHype 的基础上, 进一步将宿主机 Linux 内核、QEMU 和 KVM 模块移出可信基。SecFortress 利用嵌套内核和 CPU 分页技术, 将虚拟化平台解耦为一个降权的宿主机内核、一个较小的可信模块和多个 KVM 实例, 通过创建一套交叉隔离的 Hypervisor 安全加固方案, 为虚拟机和虚拟化平台创建一套双向隔离和保护机制。

表 4 基于完整性保护的 VMM 安全性加固比较

Table 4 Comparison of VMM security reinforcement based on integrity protection

系统	触发方式	触发条件	硬件辅助机制	是否防御擦除攻击	是否防御瞬时攻击
HyperGuard[23]	被动触发	周期性内存快照捕获	SMM(SMI 接口)	否	否
HyperCheck[24]	被动触发	周期性内存快照捕获	SMM(SMI 接口)	否	否
HyperSentry[25]	被动触发	周期性内存快照捕获	SMM(IPMI 接口)	是	否
Copilot[27]	被动触发	周期性内存快照捕获	EBSA	是	否
Vigilare[31]	被动触发	总线流量监听	ARM SoC	是	是
MGUARD[32]	被动触发	内核内存页监听	FB-DIMM	是	是
ED-minitor[34]	被动触发	钩子函数流量监听	无	是	是
HyperSafe[36]	主动触发	VMM 代码 CFI 控制	无	是	是

与 HyperLock、DeHype 和 SecFortress 不同, Nexen[42]是一个针对 Xen(混合型 VMM)的错误隔离 VMM 加固系统。Nexen 采用嵌套内核技术[43], 将 Xen 分解为一个用于安全控制的安全监视器、一个用于共享服务的共享域和多个用于虚拟机服务的 Xen slices。其中, 安全监视器作为可信基, 用于监视所有 MMU 更新, 通过控制 MMU 操作, Nexen 在虚拟机之间进行隔离控制和权限管理。每个虚拟机有一个属于自己的 Xen slice, Xen slice 用于绑定虚拟机并提供大部分虚拟化功能支持, 虚拟机产生的任何不恰当操作只会影响对应的 Xen slice。每个 Xen slice 有自己独立的内存地址空间, Xen slice 之间共享代码但隔离数据, 隔离操作由安全监视器完成。Xen slice 之间需要进行跨边界交互, 处于性能考虑, 额外设计一

个服务共享域, 共享拥有自己的内存地址空间且同样由安全监视器隔离控制。虚拟机与 VMM 之间的交互通过一个硬编码的跳转门进行, 从而保证了虚拟机和 VMM 之间的访问控制安全。Nexen 通过对 Xen 进行功能性解耦和安全性访问控制, 在提供完整虚拟化功能的同时, 为每个虚拟机分配一个单独的 VMM 实例和隔离的运行环境, 增强整个虚拟化平台的安全性和稳定性。

基于错误隔离的 VMM 安全性加固为每个虚拟机提供一个单独的 VMM 实例, 有效地隔离了 VMM 错误对整个平台的影响, 增强了虚拟化平台的安全性和稳定性。

4.3 基于最小化 VMM 的 VMM 安全性加固

本文 2.4 小节中提到, VMM 庞大的代码量导致其拥有巨大的攻击面, 高度复杂的组件设计和高耦合性增加了 VMM 安全保护的难度。基于完整性保护的 VMM 安全性加固可以有效地检测和防御针对 VMM 的完整性破坏攻击, 但无法减少 VMM 的代码量。基于错误隔离的 VMM 安全性加固可以将 VMM 的攻击影响限制在特定范围内, 一定程度上减少可信基, 但对 VMM 的功能完整性有一定的影响。研究证明[44], 更少的代码量可以提高程序的可信度。因此, 保证 VMM 功能完整性的同时, 最小化 VMM, 是 VMM 安全性加固的研究热点之一。一些研究工作[45-47]从不同角度和场景最小化 VMM。最小化后的 VMM 功能性完整并且易于进行完整性验证, 可以借助形式化论证等方法[48]来验证最小化后 VMM 的安全性。本节将举例介绍最小化 VMM 的研究进展。

NOVA[49]遵循最小特权原则, 利用微内核技术, 将 VMM 分为多个虚拟化功能组件并且将部分组件从 VMM 中移出。NOVA 将 VMM 分成一个 microhypervisor、一个根分区管理器、用户虚拟机、设备驱动和一些系统服务组件。Microhypervisor 作为可信基运行在 CPU 最高特权级上。NOVA 创建了五种类型的内核保护对象: 保护域、执行上下文、调度上下文、传送口和信号量。内核保护对象为 VMM 提供了基于能力的超级调用接口和组件间通信认证。NOVA 采用全新的 VMM 设计, 相比于传统 VMM, 大幅度减少可信基(一个数量级)。但是, 频繁地组件间通信和特权模式切换, 导致 NOVA 的性能较差。

Dichotomy[50]将传统 VMM 分为 hyperplexor 和 featurevisor。Hyperplexor 是一个安全且小型的 hypervisor, 用于硬件资源的管理和为 featurevisor 提供服务。Featurevisor 是一个修改过的用户态 hypervisor, 用于向虚拟机提供虚拟化服务支持。Dichotomy 根据不同虚拟机的服务需求, 提供不同的 featurevisor 对象。Dichotomy 采用 Hypervisor-as-a-Service 的思想, 把 hypervisor 看作一种服务, 采用按需分配的原则为虚拟机提供支持。此外, Dichotomy 提出短暂虚拟化的概念, 通过 featurevisor 灵活地将虚拟机管理权限暂时性交由 hyperplexor 管理, 减少不必要的性能切换开销。

HypSec[51]是一个结合硬件支持的, 基于微内核思想设计实现的最小化 VMM 系统。HypSec 将 VMM 分解成一个拥有少量代码的可信基 corevisor 和一个不可信的虚拟化处理组件 hostvisor。Corevisor 作为可信基, 拥有所有硬件资源的访问和管理权限, 提供 CPU 虚拟化和内存虚拟化数据访问支持, 通过强制访问控制策略保护虚拟机 CPU 和内存核心数据。此外, corevisor 与 VM 之间采用端到端加密 IO 来保护 IO 数据的安全性。所有的异常和中断必须由 corevisor 进行控制和处理以确保只有虚拟机可以访问其敏感数据。Hostvisor 拥有大部分的代码, 包括部分虚拟化功能代码和全部宿主机操作系统代码。Hostvisor 负责 IO 虚拟化、中断虚拟化和虚拟机生命周期的管理。Hostvisor 拥有导入和导出虚拟机加密数据的权限, 但是无法对数据进行读写操作。Corevisor 和 hostvisor 之间通过 ARM VE[52] (Virtualization Extension)进行物理隔离。ARM VE 将异常级别(Exception level)按照由低到高分为 EL0-EL3 四种。虚拟化环境下, EL0 为用户模式, EL1 为虚拟机操作系统模式, EL2 为 hypervisor 模式, EL3 为安全监视模式。异常处理只能由低异常级别迁移到高异常级别。EL2 相较于 EL1 和 EL0 拥有更高的特权级别和独立执行环境, 与 EL1 和 EL0 实现物理隔离。虚拟化平台启动过程中, HypSec 将 corevisor 加载到 EL2, hostvisor 加载到 EL1, 普通的用户程序运行在 EL0。Corevisor 运行在更高的特权级, 确保 hostvisor 无法破坏 corevisor 的代码和数据的同时, corevisor 还对虚拟机和 hostvisor 进行内存访问控制。此外, HypSec 利用 ARM TrustZone 技术[53], 支持可信执行环境启动虚拟机镜像和安全持续存储。确保虚拟机生命周期安全。HypSec 与 Dichotomy 类似, 都将传统 hypervisor 为一个可信 hypervisor 和一个用于普通管理的 hypervisor。但 Dichotomy 采用软件实现, 而 HypSec 借助硬件安全机制实现。

NoHype[54]认为 VMM 不是虚拟化平台必不可少的一部分, 应该彻底将 VMM 移除。NoHype 通过系统管理软件直接对虚拟机进行物理资源分配。NoHype 借助硬件隔离技术, 严格划分每个虚拟机的

物理资源范围。虚拟机拥有隔离范围内所有物理资源的控制权限, 运行期间不需要与任何 VMM 交互。NoHype 彻底移除了传统意义上的虚拟化层, 减少了可信基的同时, 提高了虚拟机运行性能。然而, NoHype 只能支持单核虚拟机运行, 并且依赖于硬件的物理隔离, 丧失了虚拟化平台虚拟机管理的灵活性和可扩展性。此外, 与类虚拟化类似, NoHype 需要对虚拟机内核进行额外地修改, 增加了开发和部署的难度。

最小化 VMM 的研究根本目的在于减少 VMM 的代码量, 减少平台可信基, 对 VMM 进行安全性加固。上述研究针对不同的研究场景或 VMM 模型, 基于同样的目的, 进行了各种各样的最小化 VMM 设计。然而, 目前为止, 依然没有一套通用的最小化 VMM 设计方案, 确保 VMM 功能完整性的同时避免额外的性能开销。因此, 如何最小化 VMM 依然作为一个开放性问题, 有待研究人员进一步研究。

4.4 基于嵌套虚拟化的 VMM 安全性加固

Turtles Project[55]实现了嵌套虚拟化技术, 允许在虚拟机中嵌套运行虚拟机。嵌套虚拟化技术从传统 VMM 外部引入一层抽象虚拟化抽象层, 抽象虚拟化层的运行特权级高于传统 VMM, 可以对传统 VMM 进行任意安全监控。本节将介绍利用嵌套虚拟化技术进行 VMM 安全性加固的研究。

CloudVisor[56]旨在利用嵌套虚拟化技术, 对 VMM 进行权限限制和隔离, 以保护虚拟机安全。CloudVisor 将传统 VMM 进行降权处理, 降低传统 VMM 的运行特权级到 ring1, 而 CloudVisor 本身运行在最高的 ring0 特权级, 确保 CloudVisor 可以透明地监视传统 VMM 的所有行为。CloudVisor 通过监控虚拟机和 VMM 所有的特权指令操作, 控制虚拟机和 VMM 之间的指令执行。CloudVisor 追踪并监视所有的虚拟机页表更新操作, 不允许未经授权的 VMM 修改虚拟机页表。此外, CloudVisor 采用加解密的方式提供 I/O 保护, 通过 MD5 和 Merkletree[57]确保磁盘数据的完整性。此外, CloudVisor 利用硬件提供的可信执行技术(trusted execution technology, TXT)[58]和可信平台模块(trusted platform module, TPM)技术[59]为虚拟机提供启动保护和安全校验。CloudVisor 有效地限制了 VMM 的特权操作, 确保虚拟机不被受到攻击的 VMM 影响。然而, 特权控制导致频繁地上下文切换, 造成了性能损失。特别是对于 I/O 操作频繁的虚拟机, CloudVisor 的性能损失高达 54.4%。

CloudVisor-D[60]与 CloudVisor 有共同的目的, 但是解决了 CloudVisor 性能损失严重的问题。CloudVisor-D 同样采用嵌套虚拟化技术对 VMM 进行权限限制。与 CloudVisor 不同, CloudVisor-D 解耦嵌套虚拟化设计模型, 将虚拟化平台分解为一个 RootVisor, 一个 SubVisor 和一组 Guardian-VMs。RootVisor 位于嵌套虚拟化层, 运行在特权模式, 负责启动虚拟机和虚拟机内存隔离。CloudVisor-D 每个虚拟机分配一个独立的 Guardian-VM。Guardian-VM 并不是一个完整的虚拟机, 它只包含一部分虚拟化功能。Guardian-VM 维护了一张跳转表, 监控和认证虚拟机与 SubVisor 或 RootVisor 之间所有控制权地转移。SubVisor 是传统 VMM(除去 Guardian-VM 中的功能), 用来提供普通虚拟化服务。Guardian-VM 和 RootVisor 作为可信基, 保证虚拟机安全性。CloudVisor-D 利用 Intel VMFUNC(EPTP switching)硬件技术, 动态隔离 Guardian-VM 和 SubVisor 的内存区域。CloudVisor-D 将 Guardian-VM 与虚拟机部署在同一特权级, 避免了大量上下文切换, 减少了性能开销。

基于嵌套虚拟化的 VMM 安全性加固透明地监视 VMM 的运行状态, 有效地降低了 VMM 的安全威胁。然而, 嵌套虚拟化技术不可避免地引入了性能问题。CloudVisor-D 虽然一定程度上降低了性能开销, 但它需要对虚拟机进行大幅度修改, 并且需要特定的硬件支持, 增加了开发的难度。此外, 嵌套虚拟化同样引入嵌套虚拟化层作为可信基, 其可信基依然较大, 嵌套虚拟化层的安全性有待考究。

4.5 基于加密的 VMM 安全性加固

利用加密技术来减少受到攻击的 VMM 对虚拟机的影响也是一种研究思路。利用加密技术可以有效地保护虚拟机数据的机密性。本节将举例介绍基于加密的 VMM 安全性加固研究。

安全处理器在过去二十年被广泛地研究[61-65], 其中最为重要的是在处理器中加入了 AISE+BMT 加密算法[65]。AISE(address independent seed encryption)是一种内存加密算法, 在 AISE 中, 处理器利用加密数据块的种子生成一个伪随机数, 并将伪随机数和加密数据快进行异或运算产生密文。加密数据快的种子由 LPID(Logic per-page ID)、计数器和偏移位组成。系统每进行一次加密运算, 加密数据块的种子都会改变一次。 BMT(Bonsai Merkle tree)哈希树用于对 AISE 加密数据块的种子进行完整性保护。BMT 把种子的更新值以密文的方式存储到哈希树中。处理器对种子的加载必须首先经过哈希树的完整性认证。AISE+BMT 中, AISE 保护数据机密性, BMT 保护数据的完整性。

HyperCoffer[66]采用基于 AISE+BMT 的安全处理器加密方式, 对虚拟机数据进行保护。HyperCoffer 利用处理器中 AISE 加密逻辑, 确保虚拟机数据在处理器外部以密文存在。不同虚拟机通过不同密钥隔离, 禁止 VMM 对虚拟机数据进行越权访问。HyperCoffer 为每个虚拟机设计一个 VM-Shim, 用于处理虚拟机和 VMM 交互。SecureME[67]与 HyperCoffer 采用同样的加密方式, 但是 SecureME 的场景与 HyperCoffer 不同。此外, SecureME 需要改变虚拟机中的程序逻辑, 而 HyperCoffer 对虚拟机内部保持透明。

AMD SME(Secure Memory Encryption)[68], 安全内存加密技术通过专用硬件在内存控制器(on-die memory controller)对内存进行加密。每个控制器 controller 包含一个 AES 引擎, 该引擎在将数据写入 DRAM 时对数据进行加密, 读取时对数据进行解密。数据加密使用 128 位的密钥完成。密钥由 AMD 的安全处理器管理, 该处理器是一个 32 位 microcontroller (ARM Cortex-A5), 作为一个安全子集系统集成在 AMD 的 SoC 中。AMD SEV[68](Secure Encrypted Virtualization), 安全虚拟化加密技术 , SEV 是 SME 针对虚拟化的进一步安全扩展。它支持一个 hypervisor 控制下的多个 VM 加密, 且使用不同加密密钥。SEV 硬件用 VM ASID 标记所有的代码和数据, ASID 表明数据来自哪个 VM 或者打算用于哪个 VM。该标记在 SoC 内部时一直与数据保持一致。虚拟机可以自主控制加密和共享范围。与 AISE+BMT 相比, AMD SME/SEV 加密粒度更细且更加灵活。Fidelius[69]在软件层扩展了 AMD SEV 技术, 将 VMM 的资源管理平面与功能提供平面进行逻辑切分, 通过不可绕过的内存隔离技术对其进行隔离。Fidelius 取消 VMM 对敏感资源的管理和访问权限, 将这些资源存放到一个隔离内存环境中。Fidelius 通过强制访问策略控制 VMM 对敏感资源的访问。此外, Fidelius 部署 SEV API 重用技术, 对虚拟机提供生命周期保护。通过重新设计 SEV 的 SEND 和 RECEIVE APIs, Fidelius 实现了虚拟机加密镜像的安全启动。Fidelius 提供了两个类虚拟化接口用于 I/O 加解密使用。出于性能考虑, Fidelius 采用文献[43]的方法, 创

表 5 VMM 安全性加固对比

Table 5 Comparison of security reinforcement of VMM

系统	加固方案	是否减少可信基	是否增强 VMM 组件安全	是否进行 VMM 错误隔离	虚拟机和 VMM 之间是否有访问控制
HyperGuard[23]	VMM 完整性检测	否	是	否	否
HyperCheck[24]	VMM 完整性检测	否	是	否	否
HyperSentry[25]	VMM 完整性检测	否	是	否	否
Copilot[27]	VMM 完整性检测	否	是	否	否
Vigilare[31]	VMM 完整性检测	否	是	否	否
MGUARD[32]	VMM 完整性检测	否	是	否	否
ED-minitor[34]	VMM 完整性检测	否	是	否	否
HyperSafe[36]	强制 VMM 完整性	否	是	否	否
HyperLock[38]	VMM 错误隔离	是	是	是	否
DeHype[40]	VMM 错误隔离	是	否	是	否
Nexen[42]	VMM 错误隔离	是	是	是	是
NOVA[49]	最小化 VMM	是	是	是	否
Dichotomy[50]	最小化 VMM	是	是	否	否
HypSec[51]	最小化 VMM	是	是	否	是
NoHype[54]	彻底移除 VMM	没有 VMM	没有 VMM	没有 VMM	没有 VMM
CloudVisor[56]	降权监控 VMM, 保护虚拟机机密性和完整性	否	否	否	是
CloudVisor-D[60]	降权监控 VMM, 保护虚拟机机密性和完整性	否	是	否	是
HyperCoffer[66]	保护虚拟机机密性和完整性	是	否	否	是
Fidelius[69]	保护虚拟机机密性和完整性	是	是	否	是

建了一个与 VMM 同特权级的安全轻量级特权环境。然而, Fidelius 无法防止来自虚拟机对 VMM 的破坏。

X86 结构下, 与 AMD SEV 技术类似的是 Intel SGX 技术[70-71]。Intel SGX(Software Guard Extensions)技术提供了一个物理隔离可信执行的 enclave 环境, 用户定义的敏感数据和代码被隔离运行在 enclave 中, enclave 中的数据在 enclave 中以明文存在, 离开 enclave 后以密文存在。SGX 主要防止高特权级软件对用户程序进行完整性和机密性破坏, 除此之外, SGX 由于内存限制(支持 128MB/256MB 内存大小), 不适合内存吞吐量较大的程序, 如何有效结合 SGX 和 VMM 加固有待进一步研究。

5 研究展望

5.1 VMM 平台安全加固评估维度

根据 2.4 小节的安全分析及 VMM 安全性加固方案的优缺点, 本文提出了 VMM 架构设计的四个评估维度。

表 6 性能比较

Table 6 Performance comparison

性能测	HyperLock (KVM)	DeHype(KVM)	HypSec (KVM)	Nexen (Xen)	Fidelius (Xen)	Cloudvisor (Xen)	CloudVisor-D (Xen)	HyperSafe (BitVisor)
perlbench*	4.6%	0.7%	/	≈Xen	1.4%	/	/	/
bzip2*	0.7%	4.7%	/	≈Xen	0.6%	/	/	2%
gcc*	1%	5.2%	/	<Xen	5.9%	/	/	/
mcf*	1.9%	5%	/	<Xen	17.3%	/	/	/
gobmk*	0.9%	4.2%	/	≈Xen	0.7%	/	/	/
hmmer*	0.9%	2%	/	≈Xen	≈Xen	/	/	/
sjeng*	1.4%	5%	/	≈Xen	2%	/	/	/
libquantum*	1.3%	1.5%	/	<Xen	7.5%	/	/	/
h264ref*	0.7%	3.5%	/	≈Xen	0.5%	/	/	/
omnetpp*	2.5%	3.2%	/	≈Xen	16.3%	/	/	/
astar*	0.8%	3.7%	/	≈Xen	/	/	/	/
xalancbmk*	1.8%	4.3%	/	/	/	/	/	/
kernel decompress	0.3%	3%	/	/	/	/	0.4%	2%
kernel compilation	3.8%	6%	2%	0.7%	/	6.0%	4.7%	6.2%
EPT violation handling☆	/	/	/	/	/	1133.7%	84.8%	/
Hypercall☆	/	/	10.6%	/	/	166.3%	2.9%	/
Virtual IPI☆	/	/	14%	/	/	90.3%	18.8%	/
Memcached	/	/	13%	/	/	0.1%	1.8%	/
Hackbench	/	/	1.2%	/	/	/	0.3%	/
Apache	/	/	12%	/	/	0.2%	1.4%	/
MySQL	/	/	9%	/	/	/	0.3%	/
dbench	/	/	/	/	/	54.4%	2.8%	/
TCB Size	4K SLOC	2.3K SLOC	8.5K SLOC	/	1.8K SLOC	/	5.8K SLOC	/

*项表示使用 SPEC CPU2006 的 CPU 和内存性能测试结果;
☆项表示使用 Microbenchmark 的 CPU 性能测试结果;
kernel compilation 表示源码编译内核的内存和磁盘性能测试结果;
Hackbench 表示调度压力性能测试结果;
MySQL 表示使用 sysbench 对 MySQL 进行的负载性能测试结果;
kernel decompress 表示解压内核压缩包的 CPU 性能测试结果;
Memcached 表示使用 memtier 的内存性能测试结果;
Apache 表示使用 ApacheBench 的性能测试结果;
dbench 表示 I/O 性能测试结果

(1) 可信基大小及保护。可信基作为整个 VMM 平台的安全中枢, 对 VMM 平台起着至关重要的作用。因此, 应该尽可能减少的可信基代码量, 降低漏洞存在的风险, 减小攻击面。有效的硬件安全机制、形式化论证等技术也为可信基的保护提供安全支撑。

(2) 访问控制。虚拟机执行特权操作时, CPU 切换到 VMM 中, VMM 处理完毕后再将控制权移交回虚拟机。不恰当的虚拟机操作可能引起 VMM 崩溃, 同时受攻击的 VMM 可能破坏虚拟机的敏感数据。在虚拟机和 VMM 之间进行严格的访问控制和信息

校验, 可以有效减少上述危害。

(3) 错误隔离。虚拟化平台中, 所有虚拟机共享一个 VMM, 单个虚拟机对 VMM 的影响, 将牵连到整个虚拟化平台虚拟机。良好的 VMM 隔离和错误恢复, 可以将单个虚拟机引发的错误隔离到固定的 VMM 实例中, 有效避免整个虚拟化平台的宕机。

(4) 性能和部署难度。性能的好坏决定了 VMM 架构的实用性, 只有拥有良好性能的 VMM, 才能为进行大量计算的云平台提供稳定支撑。复杂性指 VMM 的部署难度, 过于零碎的组件拆分和复杂的逻辑控制, 不利于 VMM 的广泛部署。根据性能测试指标共性, 本文挑选 HyperLock、DeHype、HypSec、Nexen 、 Fidelius 、 Cloudvisor 、 Cloudvisor-D 和 HyperSafe 8 个典型系统作为性能比较对象, 代表了基于错误隔离、基于最小 VMM、基于加密、基于嵌套虚拟化和基于完整性保护的安全架构方案。本文选择 22 个通用指标作为比较对象, 用于表示 CPU、内存和 I/O 的性能损耗。比较显示, 所有方案 CPU 性能损耗较低, 内存和 I/O 开销属于主要性能损失点。结合各方案实现要点, 我们将性能的损耗点细分为为: 1) 频繁地内存地址空间切换, 即页表切换和 TLB Flush; 2) 不同运行特权级上下文切换; 3) 内存或磁盘频繁加解密操作。具体见表 6 (表格括号中的 VMM 表示相关系统的性能测试基准 VMM; 表格中的百分数表示性能损失情况; <VMM 表示性能优于对应 VMM 基准值; ≈表示与对应的 VMM 基准值几乎相同; 空白项表示该系统未进行相关测试)。

5.2 未来工作

尽管 VMM 安全性加固技术已经取得一些研究进展, 然而依然没有一款 VMM 安全性加固设计应用到实际的工业生产中。

首先, 集权式认证模型利用控制流完整性模型等技术, 为 VMM 整个生命周期提供严格的访问控制, 增强安全性的同时, 也带来了不可避免的认证开销。其次, 细粒度的 VMM 内部隔离机制增强 VMM 容错性的同时, 导致了复杂的执行逻辑和不完整的功能支持, 并且特权操作的限制给性能带来了不容忽视的影响。基于现有 VMM 安全性加固方案中存在的缺陷, 本文归纳出三个开放性问题: (1) 如何从集权访问认证控制到分布式访问认证控制? (2)如何在提供细粒度 VMM 内部隔离的同时确保 VMM 功能的完备性, 并减少不必要的上下文切换所导致的性能开销? (3) 硬件技术的发展给 VMM 安全性加固带来了前所未有的机遇, 相较于软件, 硬件具有更安全的保护机制和更低的性能损耗, 如何结合硬件架构, 设计一套低开销、高安全的 VMM 虚拟化平台有待进一步研究。

6 总结

本文旨在对面向云环境的 VMM 平台安全性加固方案进行分析, 总结了目前已有的研究成果及存在的问题。首先介绍了传统 VMM 的架构设计和 VMM 软件, 并从安全角度对传统 VMM 进行了设计缺陷分析。其次本文将面向云环境的 VMM 平台安全性加固的研究现状分为 VMM 特权域安全性加固和 VMM 安全性加固, 并对每种类型的典型案例进行梳理。最后我们提炼出 VMM 安全性加固的四个核心点和三个开放性问题。在后续研究中, 笔者将在前人研究的基础上, 从四个核心点出发, 针对三个开放性问题, 设计一个低性能、高安全的虚拟机监视器系统。

参考文献

[1] Christopher S. Time sharing in large, fast computers, Information Processing[C]. *The 1st International Conference on Information Processing,* 1959: 336-341.

[2] IBM System/360 Model 40. https://www.ibm.com.

[3] 英特尔开源软件技术中心. 复旦大学并行处理研究所. 系统虚拟化原理与实现[M]. 北京:清华大学出版社, 2009: 65.

[4] Xen cve. https://xenbits.xen.org/xsa/. Jul. 2020.

[5] KVM cve. https://nvd.nist.gov/vuln/search. Jul. 2020.

[6] CVE-2017-17045. http://cve.mitre.org/. Sept. 2017.

[7] CVE-2017-10918. http://cve.mitre.org/cgi-bin/. Jul. 2017.

[8] Barham P, Dragovic B, Fraser K, et al. Xen and the Art of Virtualization[C]. *The nineteenth ACM symposium on Operating systems principles*, 2003: 164-177.

[9] Intel VT. https://software.intel.com. May. 2018.

[10] AMD-V. http://support.amd.com/en-us/search/tech-docs.Oct.2020.

[11] Kernel-based Virtual Machine. http://www.linux-kvm.org/. Jan. 2010.

[12] Dive into the VMware ESX Server hypervisor. https://developer.ibm.com/depmodels/cloud. Sept. 2011.

[13] Murray D G, Milos G, Hand S. Improving Xen Security through Disaggregation[C]. *The fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2008: 151-160.

[14] Hohmuth M, Peter M, Härtig H, et al. Reducing TCB Size by Using Untrusted Components: Small Kernels Versus Virtual-Machine Monitors[C]. *The 11th workshop on ACM SIGOPS European workshop*, 2004: 22-es.

[15] Singaravelu L, Pu C, Härtig H, et al. Reducing TCB Complexity for Security-Sensitive Applications: Three Case Studies[C]. *The 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, 2006: 161-174.

[16] L. Reuther, V. Uhlig, and R. Aigner. Component Interfaces in a Microkernel-based System[C]. *The 3rd Workshop on System De-*

sign Automation, 2000.

[17] DICE User's Manual. Technical report, Technische Universit at Dresden, http://os.inf.tu-dresden.de/ dice/manual.pdf. 2007.

[18] Colp P, Nanavati M, Zhu J, et al. Breaking up is Hard to Do: Security and Functionality in a Commodity Hypervisor[C]. *The Twenty-Third ACM Symposium on Operating Systems Principles*, 2011: 189-202.

[19] G. Candea, S. Kawamoto, Y. Fujiki, et al. Microreboot — a technique for cheap recovery[C]. *6th Symposium on Operating System Design and Implementation*, 2004: 31-44.

[20] Baker M, Sullivan M. The recovery box: Using fast re-covery to provide high availability in the UNIX environ-ment[C]. *USENIX Summer Conference*, June 1992: 31-43.

[21] Butt S, Lagar-Cavilla H A, Srivastava A, et al. Self-Service Cloud Computing[C]. *The 2012 ACM conference on Computer and communications security*, 2012: 253-264.

[22] Berger S, Caceres R, Goldman K, et al. vTPM: Virtualizing the Trusted Platform Module[C]. *The 15th USENIX ecurity Symposium*, 2006.

[23] Wojtczuk R and Rutkowska J. Xen 0wning trilogy[C]. *Black Hat conference,* 2008.

[24] Wang J, Stavrou A, Ghosh A. HyperCheck: A Hardware-Assisted Integrity Monitor[C]. *The 13th international conference on Recent advances in intrusion detection*, 2010: 158-177.

[25] Azab A M, Ning P, Wang Z, et al. HyperSentry: Enabling Stealthy In-Context Measurement of Hypervisor Integrity[C]. *The 17th ACM conference on Computer and communications security*, 2010: 38-49.

[26] Intel, HP, NEC, and Dell. IPMI v2.0. http://download.intel.com, Feb. 2004.

[27] N. L. Petroni, T. Fraser, J. Molina, et al. Copilot - A Coproces-sor-based Kernel Runtime Integrity Monitor[C]. *The 13th USENIX Security Symposium*, 2004: 13-13.

[28] Intel StrongARM EBSA-285 evaluation board. http://netwinder.osuosl.org. May. 1995.

[29] Rutkowska J. Beyond The CPU: Defeating Hardware Based RAM Acquisition Tools[C]. *Blackhat*, February 2007.

[30] Wei J P, Payne B D, Giffin J, et al. Soft-Timer Driven Transient Kernel Control Flow Attacks and Defense[C]. *2008 Annual Computer Security Applications Conference*, 2008: 97-107.

[31] Moon H, Lee H, Heo I, et al. Detecting and Preventing Kernel Rootkit Attacks with Bus Snooping[C]. *IEEE Transactions on Dependable and Secure Computing*, 2015: 145-157.

[32] Liu Z Y, Lee J, Zeng J Y, et al. CPU Transparent Protection of OS Kernel and Hypervisor Integrity with Programmable DRAM[C]. *The 40th Annual International Symposium on Computer Architecture*, 2013: 392-403.

[33] JEDEC Standard. Fbdimm: Architecture and Protocol. Jan. 2007.

[34] Deng L, Liu P, Xu J, et al. Dancing with Wolves: Towards Practical Event-Driven VMM Monitoring[C]. *The 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2017: 83-96.

[35] Payne B D, Carbone M, Sharif M, et al. Lares: An Architecture for Secure Active Monitoring Using Virtualization[C]. *2008 IEEE Symposium on Security and Privacy*, 2008: 233-247.

[36] Wang Z, Jiang X X. HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity[C]. *2010 IEEE Symposium on Security and Privacy*, 2010: 380-395.

[37] Abadi M, Budiu M H, Erlingsson Ú, et al. Control-Flow Integrity Principles, Implementations, and Applications[J]. *ACM Transactions on Information and System Security*, 2009, 13(1): 1-40.

[38] Wang Z, Wu C, Grace M, et al. Isolating Commodity Hosted Hypervisors with HyperLock[C]. *The 7th ACM european conference on Computer Systems*, 2012: 127-140.

[39] Kernel Samepage Merging. http://lwn.net/Articles/330589/. Dec. 2009.

[40] Wu Chiachih, Wang Z, and Jiang X X. Taming Hosted Hypervisors with (mostly) Deprivileged Execution[C].*20th Annual Network and Distributed System Security Symposium,* 2013.

[41] Zhou Q H, Jia X Q, Zhang S Z, et al. SecFortress: Securing Hypervisor Using Cross-Layer Isolation[C]. *2022 IEEE International Parallel and Distributed Processing Symposium*, 2022: 212-222.

[42] Shi L, Wu Y M, Xia Y B, et al. Deconstructing Xen[C]. *2017 Network and Distributed System Security Symposium*, 2017.

[43] Dautenhahn N, Kasampalis T, Dietz W, et al. Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation[C]. *The Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015: 191-206.

[44] Singaravelu L, Pu C, Härtig H, et al. Reducing TCB Complexity for Security-Sensitive Applications: Three Case Studies[C]. *The 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, 2006: 161-174.

[45] McCune J M, Parno B J, Perrig A, et al. Flicker: An Execution Infrastructure for Tcb Minimization[C]. *The 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, 2008: 315-328.

[46] McCune J M, Li Y L, Qu N, et al. TrustVisor: Efficient TCB Reduction and Attestation[C]. *2010 IEEE Symposium on Security and Privacy*, 2010: 143-158.

[47] Szefer J, Keller E, Lee R B, et al. Eliminating the Hypervisor Attack Surface for a more Secure Cloud[C]. *The 18th ACM conference on Computer and communications security*, 2011: 401-412.

[48] Klein G, Elphinstone K, Heiser G, et al. seL4: Formal Verification of an OS Kernel[C]. *The ACM SIGOPS 22nd symposium on Operating systems principles*, 2009: 207-220.

[49] Steinberg U, Kauer B. NOVA: A Microhypervisor-Based Secure Virtualization Architecture[C]. *The 5th European conference on Computer systems*, 2010: 209-222.

[50] Williams D, Hu Y H, Deshpande U, et al. Enabling Efficient Hypervisor-As-a-Service Clouds with Ephemeral Virtualization[C]. *The 12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2016: 79-92.

[51] Li S W, Koh J S, Nieh J. Protecting Cloud Virtual Machines from Commodity Hypervisor and Host Operating System Exploits[C]. *The 28th USENIX Conference on Security Symposium*, 2019: 1357-1374.

[52] Dall C and Nieh J. KVM/ARM: Experiences Building the Linux

ARM Hypervisor. Technical Report CUCS-010-13, Department of Computer Science, Columbia University, Jun. 2013.

[53] ARM Security Technology-Building a Secure System using TrustZone Technology. http://ifoceter.arm.com. Apr. 2009.

[54] Keller E, Szefer J, Rexford J, et al. NoHype: Virtualized Cloud Infrastructure without the Virtualization[C]. *The 37th annual international symposium on Computer architecture*, 2010: 350-361.

[55] Ben-Yehuda M, Day M D, Dubitzky Z, et al. The turtles project: Design and implementation of nested virtualization[C]. *9th USENIX Symposium on Operating Systems Design and Implementation*, 2010: 423-436.

[56] Zhang F Z, Chen J, Chen H B, et al. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization[C]. *The Twenty-Third ACM Symposium on Operating Systems Principles*, 2011: 203-216.

[57] Merkle R C. Protocols for Public Key Cryptosystems[C]. *1980 IEEE Symposium on Security and Privacy*, 1980: 122-134.

[58] Intel Trusted Execution Technology. https://www.intel.com/ technology/security, 2010.

[59] Trusted Computing Group. Trusted platform module. http://www.trustedcomputinggroup.org, 2010.

[60] Mi Z, Chen D H, Zang B, et al. (Mostly) Exitless VM Pro-tection from Untrusted VMM through Disaggregated Nested Virtualization[C]. *The 29th Usenix Security Symposium,* 2020: 1695-1712.

[61] Thekkath D L C, Mitchell M, Lincoln P, et al. Architectural Support for Copy and Tamper Resistant Software[C]. *The ninth international conference on Architectural support for programming languages and operating systems*, 2000: 168-177.

[62] Lie D, Thekkath C A, Horowitz M. Implementing an Untrusted Operating System on Trusted Hardware[C]. *The nineteenth ACM symposium on Operating systems principles*, 2003: 178-192.

[63] Gassend B, Suh G E, Clarke D, et al. Caches and Hash Trees for Efficient Memory Integrity Verification[C]. *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings*, 2003: 295-306.

[64] Yan C Y, Englender D, Prvulovic M, et al. Improving Cost, Performance, and Security of Memory Encryption and Authentication[C]. *33rd International Symposium on Computer Architecture*, 2006: 179-190.

[65] 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[C]. *40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007: 183-196.

[66] Xia Y B, Liu Y T, Chen H B. Architecture Support for Guest-Transparent VM Protection from Untrusted Hypervisor and Physical Attacks[C]. *2013 IEEE 19th International Symposium on High Performance Computer Architecture*, 2013: 246-257.

[67] Chhabra S, Rogers B, Solihin Y, et al. SecureME: A Hardware-Software Approach to Full System Security[C]. *The international conference on Supercomputing*, 2011: 108-119.

[68] AMD SME/SEV, https://developer.amd.com/. Sept. 2018.

[69] Wu Y M, Liu Y T, Liu R F, et al. Comprehensive VM Protection Against Untrusted Hypervisor through Retrofitted AMD Memory Encryption[C]. *2018 IEEE International Symposium on High Performance Computer Architecture*, 2018: 441-453.

[70] McKeen F, Alexandrovich I, Berenzon A, et al. Innovative Instructions and Software Model for Isolated Execution[C]. *The 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013: 1.

[71] Costan V, Devadas S. Intel SGX Explained[J]. *IACR Cryptology EPrint Archive*, 2016, 2016: 86.

周启航 于 2022 年在中国科学院大学网络空间安全专业获得博士学位。现任中国科学院信息工程研究所助理研究员。研究领域为虚拟化安全、系统安全。Email: zhouqihang@iie.ac.cn

贾晓启 于 2010 年在中国科学院研究生院信息安全专业获得博士学位。现任中国科学院信息工程研究所研究员。研究领域为系统安全。Email: jiaxiaoqi@iie.ac.cn

张伟娟 于 2018 年在中国科学院信息工程研究所信息安全专业获得博士学位。现任信息工程研究所高级工程师。研究领域为云计算安全。Email: zhangweijuan@iie.ac.cn

姜楠 于 2018 年在电子科技大学信息安全专业获得学士学位。现在中国科学院大学网络空间安全专业攻读博士学位。研究领域为系统安全、虚拟化安全。Email: jiangnan@iie.ac.cn