中国科学院大学
University of Chinese Academy of Sciences

硕士学位论文

基于同层地址空间隔离的虚拟机内存保护技术的研究

作者姓名：刘文清

指导教师：涂碧波　研究员

中国科学院大学信息工程研究所

学位类别：工学硕士

学科专业：网络空间安全

研　究　所：中国科学院大学信息工程研究所

二〇一九年四月

Research on VM Memory Protection Technology Based on the Same Privilege-level Address Space Isolation Mechanism

by

Wenqing Liu

A Thesis Submitted to
University of Chinese Academy of Sciences
in Partial Fulfillment of the Requirement
for the Degree of
Master of Engineering

Institute of Information Engineering
University of Chinese Academy of Sciences
April, 2019

声　　明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：　　日期：

论文版权使用授权书

本人授权中国科学院信息工程研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书）

作者签名：　　导师签名：　　日期：

摘 要

一旦远程攻击者或者本地攻击者攻陷Hypervisor，他们就可以轻松地获取并访问到其余用户位于客户机内存上的敏感数据信息。因此，保护虚拟机是十分必要的，尤其是保护虚拟机内存和上下文，从而避免来自非可信Hypervisor的攻击。然而，先前的一些方法才采用了额外定制的硬件来提高系统的安全性，阻止Hypervisor对敏感数据的恶意访问，但是使用硬件对于云平台提供商来说不太方便。或者是使用高于Hypervisor特权层的软件层，将数据或者对其的访问操作放在该软件层中，从而阻止Hypervisor进一步的攻击威胁。

这篇文章提出了HyperMI框架，基于同层隔离技术创建了一个可信执行环境，用于对抗非可信的Hypervisor，提供对多租户中虚拟机的运行时保护。首先，在HyperMI中提出了一个安全隔离的可信执行环境，HyperMI World，其与Hypervisor位于同一特权级别，不是依赖更高特权级别创建的软件方案，也不是依赖定制的硬件。其次，提出了事件驱动的虚拟机监控，用于监控虚拟机与Hypervisor的交互过程，并且将交互过程重定向到HyperMI处理来进行安全检查。最后，本文提出了高效的虚拟机隔离技术，通过隔离虚拟机和Hypervisor之间的内存，实现对虚拟机内存的安全防护。

该框架的关键技术在于HyperMI剥夺了虚拟机与Hypervisor的交互功能和虚拟机的地址映射功能，将这两部分函数处理放到HyperMI中处理。最终，HyperMI实现了交互关键数据的监控，阻止系统信息泄露攻击；实现了虚拟机内存高强度隔离，当物理页被分配时能够控制物理页的分配和虚实地址映射，阻止恶意的虚拟机内存越权访问攻击。我们在X86平台使用KVM虚拟机监控器和多台Linux系统的客户机完成了HyperMI系统，该系统使用于商业云平台提供商，有一定的可移植性、一定的适用性。安全分析表明该框架可以提供有效的虚拟机内存高强度隔离和虚拟机监控，性能开销测评表明该框架的高效性。

本文的创新点主要有：

- 通过同层地址空间隔离实现与Hypervisor同特权级别的安全可信执行环境；
- 通过监控虚拟机与Hypervisor的交互数据实现虚拟机监控；
- 通过页表标记与跟踪实现虚拟机内存的高强度隔离防护。

关键词：虚拟化，虚拟机防护，虚拟机安全

Research on VM Memory Protection Technology Based on the Same Privilege-level Address Space Isolation Mechanism

Wenqing Liu (Cyberspace Security)
Directed by Bibo Tu

Once compromising the Hypervisor, remote or local adversaries can easily access other customers' sensitive data in the memory and context of guest virtual machines (VMs). Therefore, it is essential to protect VMs, especially the context and the memory of VMs from the compromised Hypervisor. However, previous efforts employ extra customized hardware which is not convenient for cloud providers to adopt widely. Or they employ new architecture relying on a higher privilege level than Hypervisor.

This thesis proposes HyperMI, a novel approach to provide runtime protection for VMs based on a privilege-level secure execution environment against compromised Hypervisor. Firstly, we propose HyperMI world which is a secure and isolated trusted execution environment. HyperMI world is designed to be placed at the same privilege-level with the Hypervisor and does not rely on any additional hardware or a higher privileged level than the Hypervisor. Secondly, we propose event-driven VM monitoring which intercepts interaction between all VMs and Hypervisor and redirects interaction process to HyperMI for security check. Thirdly, we propose effective VM isolation to provide runtime protection for VMs by isolating memory among VMs and Hypervisor securely. The key of HyperMI is that HyperMI decouples the functions of interaction between VMs and Hypervisor, and the functions of address mapping of VMs from the compromised Hypervisor. As a result, HyperMI isolates memory completely, controls memory mapping when a page is allocated to a VM and resists malicious memory of VMs accessing from the compromised Hypervisor. We have implemented a prototype for KVM Hypervisor on x86 platform with multiple Linux as guest OSes, which can be used in commercial cloud computing industry with portability and compatibility. The security analysis shows that this approach can protect VMs with effective isolation and event-driven monitoring, and the performance evaluation confirms the efficiency of HyperMI.

Keywords: Virtualization; VM Protection; VM Security

目　录

图目录

表目录

第一章　绪论

1.1　研究背景与意义

1.1.1　云坏境安全

近几年，数据泄露，网络攻击，随着物联网设备的激增，网络攻击目标泛化，并成指数级增加。用户的隐私数据泄露问题是特别严重的，数据安全问题已经发展成为一个全球性问题……信息泄露的方式从员工内部泄露到外部的黑客攻击。

云计算技术利用了一些虚拟化技术、资源动态均衡分配技术等，来给上层的多租户提供资源。这些租户使用相同的物理资源，即底层的硬件资源，这些资源是通过虚拟机监控器（VMM、VM machine monitor、Hypervisor）进行统一的分配和管理。为了能够对上层的云租户（虚拟机）进行正常的管理，对其使用物理资源，可以及时分配、管理、释放物理资源，虚拟化技术被提出来。

当前，虚拟化技术在云计算中发挥越来越重要的作用，云计算平台在互联网中越来越重要。云计算平台上的多租户共享物理资源，然而物理资源是由底层的Hypervisor进行管理的。由于Hypervisor被授予最高权限，攻击者危害Hypervisor 可能会危及整个云计算基础设施，并危及云中的任何数据。

Hypervisor存在这样的弱点：1）当Hypervisor的代码量增大时，其存在的漏洞也越多，攻击者的攻击面也越广。2）Hypervisor和VM 之间需要频繁交互，一旦Hypervisor被攻击，则VM也会受到影响。

1.1.1.1　代码量增大，攻击面越广

虚拟化技术发展很快，越来越多的功能逐渐添加到Hypervisor 中，其代码量也逐渐增大。到目前为止，内核2.6.36.1 中XEN 和KVM 的代码量分别达到30 万行[1]和33.6万行[1]，当然，越来越多的代码量，也就意味着它会存在着许多有漏洞的地方，更容易被攻击的地方。

根据CVE 漏洞网站相关的数据，从2004年到现在，有357个和XEN相关的漏洞，130个和KVM相关的漏洞[2]，参看图1.1，比如，CVE-2017-1087[3]是个高危漏洞，攻击者可以利用它进行提权来攻击Hypervisor，从而对云平台上的租户进行攻击。因为云平台上的多租户共享物理资源，其中一种主要的资源就是物理内存，当多租户中某一租户被攻击，那么很可能发生跨域攻击。

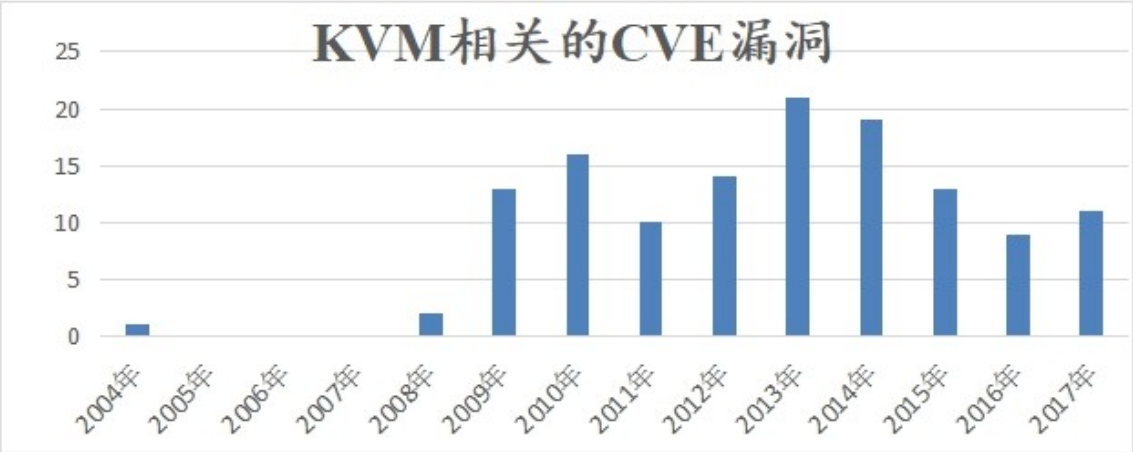

图 1.1　2004-2017年KVM相关漏洞

1.1.1.2　Hypervisor和VM 之间频繁交互

Hypervisor管理着所有VM的运行和物理资源分配。I/O设备的访问、内存资源的分配等。由于VM运行的过程中会使用到一些特权指令，但是这些指令不能被模拟，必须陷入到Hypervisor中进行处理，在这个过程中它们需要大量的交互。交互的过程很容易被攻击。

从上述的相关信息中可以得到，所以从Hypervisor 的角度对虚拟机进行保护是至关重要的。一个是控制Hypervisor的代码量，尽量不增加其代码量，另一方面，对Hypervisor和VM之间的交互进行监控保护。并且可以直接隔离保护VM的物理内存资源，从而达到最终的目标——减少来自恶意的Hypervisor攻击，对VM进行保护。

1.1.2　云平台架构

1.1.2.1　Hypervisor

在虚拟化层，作为一个模块或者与宿主机操作系统一体化成为虚拟机监控器，为上层多租户提供统一的资源分配和管理。对于底层的硬件，虚拟化技术提供了各种虚拟化，对内存、CPU、外设等，从而使得底层的虚拟化对于上层的虚拟机是透明的，虚拟机可以正常地运行。虚拟化技术根据对敏感指令的处理方式不同可以被分为3类，二进制模拟、半虚拟化、全虚拟化。

1.1.2.2　KVM

KVM是其中一种基于硬件虚拟化技术的虚拟机监控器，如图1.2。

1.1.2.3　虚拟机上下文切换

虚拟机监控器为上层虚拟机提供资源分配和管理技术，每一个物理核每次只能运行虚拟机或者Hypervisor，所以在一个物理核上需要进行系统切换，在切换的过程中存

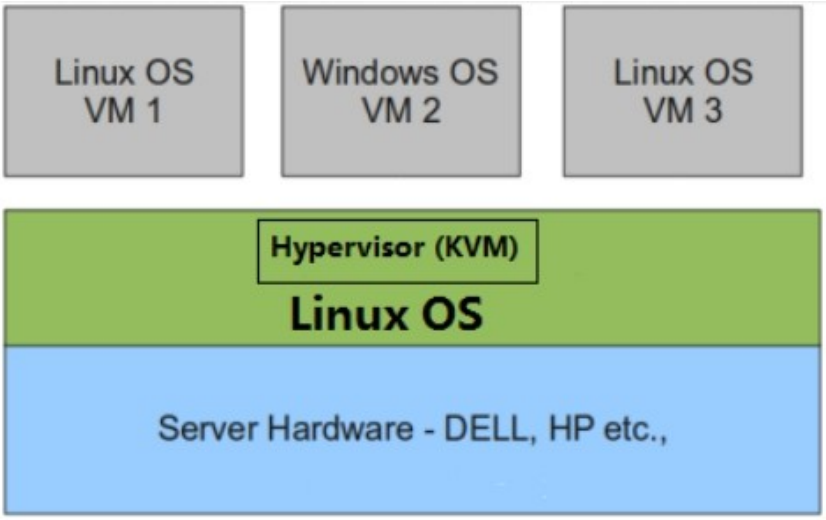

图 1.2 KVM架构图

在寄存器等系统的关键数据结构的内容变化，以便在下次切换时，能够正常进入系统。这些关键数据结构被称为上下文，这些数据被存放在VMCS结构体中。

虚拟机上下文切换主要实现虚拟机与宿主机的交互，该交互过程包含两个过程，即虚拟机进入（VM ENTRY）和虚拟机退出（VM EXIT）。虚拟机退出是因为虚拟机运行时，遇到一些敏感指令和特权指令无法处理时，需要退出到Hypervisor，让Hypervisor执行这些指令。那么这个过程叫做虚拟机退出，在此期间会进行一些交互，将虚拟机的寄存器信息和一些重要的状态信息写入到VMCS结构体中，同时读取VMCS中有关宿主机的寄存器等信息，转到宿主机执行状态。在这个过程中，会根据退出的不同原因处理退出事件。虚拟机进入则是相反的过程，将宿主机的寄存器信息和一些重要的状态信息写入到VMCS结构体中，同时读取VMCS中有关虚拟机的寄存器等信息，转到虚拟机执行状态。

1.1.3 安全问题

Hypervisor位于云环境中最底层架构，为上层的虚拟机提供资源管理功能，当Hypervisor被攻击后，上层的虚拟机的安全性会受到严重威胁。

Hypervisor作为一种模块或者一个操作系统，在内核层或者虚拟化层拥有最高权限，管理着上层的虚拟机，对于虚拟机的物理资源分配、访问、管理和释放操作，提供所有的管理。例如，当虚拟机启动时，需要内存，Hypervisor为其分配足够的内存资源，确保其内存访问时成功的地址翻译，能够确保访问到真正的物理内存，及时地进行内存的申请和释放，对于磁盘、网络等同样做处理。当Hypervisor不可信的时候，保护虚拟机的安全性是至关重要的，并且远比想象中的复杂。

Hypervisor有两种类型，半虚拟化和全虚拟化，虚拟机都依赖于Hypervisor，Hypervisor能为虚拟机的运行提供内存、进程、文件、设备管理和访问等，Hypervisor位

于系统硬件和虚拟机之间，这一事实使得虚拟机的安全与整个宿主机和Hyperisor的安全性绑在了一起，从而使得虚拟机所承受的攻击面增大，除了自身软件安全漏洞，还可能涉及来自不可信Hypervisor的攻击。首先，远程攻击者或者本地攻击者可以通过各种方法来攻击Hypervisor使得其不安全，远程攻击者可以通过同驻攻击检测位于同一物理机器上的虚拟机，随后通过利用虚拟机上的软件漏洞实现提权，逐渐攻陷Hypervisor；本地攻击者可以直接通过利用Hypervisor上的软件漏洞攻陷Hypervisor。从而通过Hypervisor利用高权限攻击上层的虚拟机。其次，对于虚拟机来说，面对的威胁主要是两点：1）自身系统软件的漏洞被攻击者利用；2）通过被攻陷的高权限级别的Hypervisor进行攻击。该攻击可以对Hypervisor与虚拟机的交互数据、系统关键数据、硬件信息等进行破坏，进一步进行虚拟机跨域攻击、虚拟机逃逸攻击。

虚拟化软件栈系统的脆弱性给虚拟机的安全性带来一定的挑战，加剧了虚拟机自身的安全风险。为了保证虚拟机的安全性，当前的研究提出了一些虚拟化安全对策，虚拟机隔离与访问控制、虚拟机安全加固、Hypervisor安全防护以及针对各种体系架构的隔离组件，部分策略针对于商业上的云平台租户来说可移植性、可用性有一定的弱势，对系统的平台具有一定的依赖性（ARM中TrustZone、Intel中的SGX、AMD中的SEV）。因此，如何在存在安全隐患的Hypervisor环境下提供对上层虚拟机的内存安全防护，保证系统的实用性、可移植性，不过度依赖平台系统，是一个重要的议题。

云环境下，对虚拟机的安全性有两个要求，交互安全和虚拟机自身内存数据安全。

A. 虚拟机与Hypervisor交互安全性

底层的Hypervisor为上层的虚拟机提供内存、文件、进程、设备资源管理，它们之间存在过多的交互。一个CPU上一次只能运行一个系统，虚拟机和Hypervisor采用分时策略在CPU上运行。还有一个关键数据——上下文，不能忽略，其发挥的作用是在虚拟机和Hypervisor在CPU上进行交换时记录各自的系统关键数据。Hypervisor的不可信，会恶意篡改上下文，篡改或者泄露关键数据，导致系统被进一步的攻击。那么，保证交互安全性具有重要意义。

B. 虚拟机自身内存数据的安全性

云环境中的宿主机支持多租户，即一台物理宿主机上可以存在被用户使用的许多虚拟机，这些虚拟机共同使用硬件资源（内存、外设等），同时工作在宿主机上。它们共同使用宿主机的内存资源，那么内存资源的协调和管理是由Hypervisor来完成。当虚拟机共享物理内存时，正常的虚拟机会被恶意第三方（恶意的虚拟机）攻击，造成内存信息泄露，打破了云租户的隔离边界。

那么，保证虚拟机之间内存隔离安全性具有重要的意义。

C. 可移植性

首先，软件移植性是设计出来的软件是独立于运行的硬件计算机环境，一般指处

理器平台，各种计算机架构对应的安全硬件特性（SGX、Trustzone、SMM等），特殊寄存器等。对于当前的云平台提供商来讲，良好的可移植性可以减小成本、软件开发周期、系统的适应性等。

其次，针对于商业版的软件开发，本系统要做到可移植性良好的功能。当前硬件平台包含有Intel处理器，ARM处理器，AMD处理器等，不同的处理器中包含有不同的特殊组件。

Intel处理器中包含SGX（Software Guard Extensions），是TEE（可信执行环境）的一种实现技术，面向应用程序开发人员[4]。英特尔公司通过在第六代英特尔酷睿处理器上引入有关SGX的新指令集，使用特殊的指令和软件将应用程序代码放到Enclave中执行。Enclave理解为一个数据运行的安全环境，可以称它为“小黑匣”。Enclave提供一个隔离的可信执行环境，当BIOS、驱动程序、Hypervisor被攻陷时，Enclave仍然能够提供对代码和数据的保护，保障用户数据和代码的安全性。可以这样理解，SGX的软件保护功能并不是识别或者隔离系统中的恶意软件，而是将合法软件针对敏感数据（如加密密钥、密码、用户数据等）的操作封装于Enclave中，使得恶意软件无法对敏感数据进行访问。

ARM处理器上包含Trustzone组件，用来在ARM处理器上提供独立的安全操作系统及硬件虚拟化技术，为手机安全支付的过程中提供TEE。该技术是一种软件和硬件结合的技术，将处理器上的软硬件资源隔离成两个环境，分别是安全环境和普通环境，其意图类似于SGX，阻止恶意软件访问敏感数据，主要是通过在硬件上提供资源隔离安全方案，软件上提供基本的安全服务和接口实现，将敏感操作和敏感数据（如指纹识别、密码处理、数据加解密、安全认证等）放在安全环境中执行和访问，其余放在普通环境中处理。

AMD处理器SEV（Secure Encrypted Virtualization），即安全加密虚拟化技术，和SME（AMD Secure Memory Encryption）技术，即安全内存加密技术，为系统提供安全机制。SME技术使用一个单密钥来加密系统内存，这个密钥被保存在系统启动时的AMD安全处理器中。SEV技术针对每一台虚拟机采用一个密钥，通过这样的方法来隔离客户机和Hypervisor，密钥被AMD的安全处理器管理，客户机可以确定哪些物理内存页可以被加密。这两种技术可以保证敏感数据的安全性，在虚拟机内存隔离上具有一定的意义，防止敏感数据泄露和恶意访问。

在不依赖硬件平台上开发的软件具有一定的意义，对于云平台提供商来说，方便，快捷，系统开发代价低，损耗小。

综上所述，如何在非可信云环境中构建一个可信执行环境来保证虚拟机内存的保密性和完整性是虚拟机隔离的一个重要分支。当涉及到虚拟机隔离是，对于商业平台（云平台提供商）来说，保证系统性价比，软件系统的可移植性、不依赖系统硬件平台是必须要考虑的因素。

1.1.4 研究意义

宿主机支持多租户运行使得计算量和计算速度发展越来越快。Hypervisor的代码量随着功能逐渐添加而组件增大，那么代码所面临的代码漏洞也越来越多。在非可信执行环境下保证上层虚拟机的内存安全性具有很大的研究意义，虚拟机隔离技术发展越来越成熟。选用同层地址空间隔离创建类似SGX、Trustzone的可信执行环境这种软件方法避免了特定硬件设备的定制。如果虚拟化环境被攻破，虚拟机和Hypervisor的交互会受到严重威胁，导致随后的虚拟机之间发生通信或者虚拟机跨域攻击，边界隔离性受到威胁，从而导致虚拟机内存保密性和完整性无法得到保障。

首先，攻击者通过远程攻击攻破其中一台虚拟机，通过虚拟机逃逸攻击实现提权取得宿主机的控制权，随后可以进行虚拟机跨域攻击[5]，即一台虚拟机去攻击另一台虚拟机，虚拟机的内存信息无法保障。在宿主机上无法实现对虚拟机的保护，虚拟机逃逸攻击、虚拟机跨域攻击、提权攻击、内存信息泄露都无法防范。

其次，攻击者可以通过本地攻击的方式攻击Hypervisor，Hypervisor本身存在一定的脆弱性。Hypervisor本身因其代码量逐渐扩大、功能复杂性导致其安全性是一个开放性问题。在不安全的环境下保证虚拟机的安全性是一个研究的议题，虚拟机隔离技术在这种场景下也相对重要。

同样，国内外研究学者对Hypervisor的安全性也提出了一些策略，保证其完整性安全[6]，对Hypervisor重新改造，导致系统改造大；或者添加一些定制的硬件，保证虚拟机的敏感数据的机密性和完整性，对于云平台提供商来说并不合适，因为每次对系统功能的添加都会导致大量的系统更改。那么，软件方法对于提供商来说是越来越重要，性能开销越小对云平台提供商越有利。

综上，宿主机和Hypervisor本身的脆弱性会增加虚拟机面对的攻击面，创建安全隔离的可执行环境（TEE）具有重要的意义[7]，将虚拟机交互数据和虚拟机本身内存数据访问的关键操作放在TEE中执行，虚拟机的机密性和完整性才能受到保障；合理的软件方法能够保证系统的可移植性，对云平台提供商具有一定的可实用性。若能实现上述要求的方法，则能保证虚拟机内存数据安全性。

1.2 国内外研究现状

为了保障云环境下虚拟机中用户的隐私安全，国内外学者从不同的角度实现了不同的方法。主要是对Hypervisor进行完整性验证和防护，或者在非可信Hypervisor环境下实现可信执行环境（TEE）实现对虚拟机的安全防护，TEE可以通过软件实现或者硬件实现。如下是国内外学者的部分研究成果。

国内

针对在不可信的执行环境中提供对虚拟机的保护。国内各研究学者提出了各自的保护策略。复旦大学提出CloudVisor[8]和Tinychecker[9]，都是使用更高特权级别的软件

控制对低特权级的访问[10]。Cloudvisor利用嵌套虚拟化将策略从Hypervisor剥离，透明的对虚拟机进行保护。Tinychecker利用嵌套虚拟化技术透明地检查和恢复Hypervsior的故障，利用嵌套虚拟化提供的隔离性可以实现安全的IDS和蜜罐。比如说，V-Met[11]将基于VMI的IDS隔离于虚拟化系统，避免不安全虚拟化系统对IDS 的影响。上海交通大学并行与系统安全实验室提出了Nexen[12]，主要通过将Xen进行功能分割，以最小特权的方式将主要功能保留，每个VM拥有一个Xen片段，它们公共的功能是由共享服务域进行处理，同时采用了嵌套虚拟化的技术布置了一个安全的监控器，用来监控内存管理和特权指令操作，针对DOS攻击进行防御。西安交通大学提出了Secpod[13]，在x86平台上创建安全的执行环境，性能开销较嵌套虚拟化和微内核小。

国外

普林斯顿大学提出了NoHype[14]的方案，利用设备的虚拟化特性通过资源预分配的方式实现资源的物理隔离，进行了彻底的隔离；北卡罗纳州立大学提出了对Hypervisor进行分割的技术，从逻辑上对Hypervisor的代码进行分割，保证每个虚拟机对应一份可运行的Hypervisor。韩国科技大学提出H-SVM[15]的方法，利用CPU的硬件隔离特性，将Hypervisor中相应的功能替换为自身提供的微代码，从而限制Hypervisor对虚拟机内存资源的操作。VMWare公司提出的虚拟化平台vSphere，其可以构建高响应的云计算基础架构,使用户能够自信地运行关键业务程序，高效地对其业务作出响应。

针对国内外学者提出的研究成果进行详细分析，主要是创建可信执行环境和VM的防护两个部分。可信执行环境包含同层地址空间隔离技术实现隔离空间[16],[17],[18]，定制硬件实现隔离空间[19],[20]等。对VM的防护主要是重组Hypervisor，实现对Hypervisor的完整性保护[18],[21],[22]，实现不同组件和功能。

1.2.1 同层隔离技术

1.2.1.1 SKEE

为了提供对内核监控和保护，SKEE创建了与内核同级别的可信执行环境TEE，基于ARM平台，其主要目标是允许对内核进行安全的监视和保护，而不需要高级特权软件和虚拟化扩展的积极参与。

SKEE[23]提供了一套保证隔离的新技术。它创建了一个内核无法访问的受保护地址空间，内核和独立环境共享相同的特权级别时。SKEE通过阻止内核管理自己的内存转换表来实现隔离技术。因此，内核被迫切换到SKEE来修改系统的内存布局。反过来，SKEE会验证来自内核的修改请求并且不会影响受保护地址空间的隔离。从操作系统内核到SKEE的切换只通过一个控制良好的开关门。这个开关门经过精心设计，因此它的执行顺序是原子的和确定性的。这些属性的结合保证了潜在的非可信内核不能利用交换序列来破坏隔离。如果内核试图违反这些属性，它只会导致系统失败，而不会

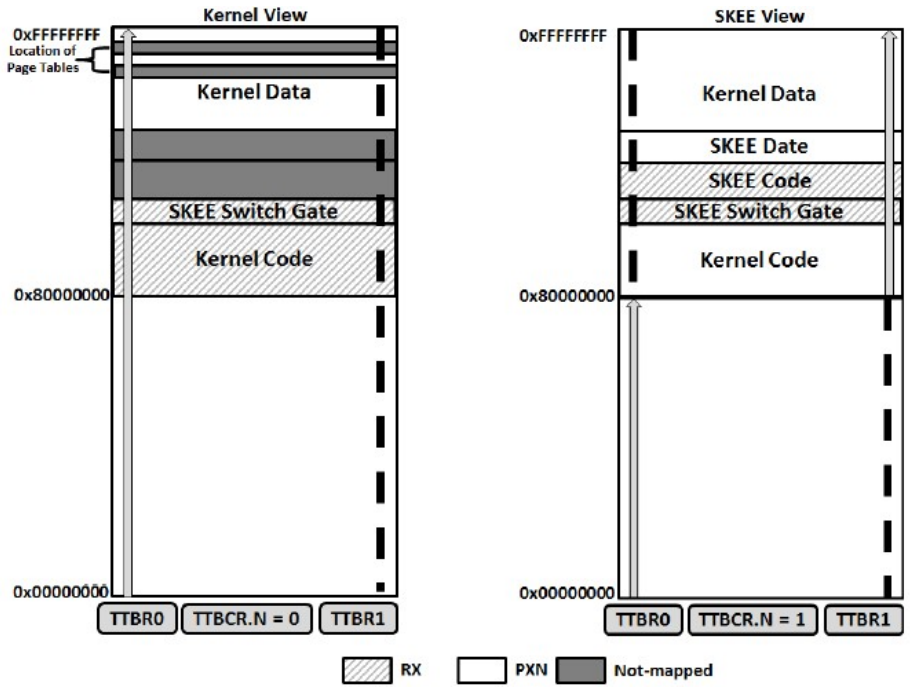

图 1.3　SKEE隔离空间实现

暴露受保护的地址空间。

SKEE专门控制整个操作系统内存的访问权限。因此，它可以防止试图将未验证的代码注入内核的攻击。此外，它还可以很容易地扩展到拦截其他系统事件，以支持各种入侵检测[24],[25]和完整性验证工具。该框架的优点在于基于ARM平台实现对内核的安全防护。

使用同层隔离思想，在同一份页表上创建不同的地址空间，基本实现如下图1.3。能够阻止原系统内核对页表和MMU的特权访问。在实现地址空间切换时保证隔离空间的安全性。

1.2.1.2　ED-Monitor

南京大学提出了一种新的框架ED-Monitor[1]，用于云计算中不受信任的虚拟机监视器（VMMs）的事件驱动监视。与以前的VMM监控方法不同，该框架既不依赖于更高的特权级别，也不需要任何特殊的硬件支持。相反，将受信任的监视器放在与不受信任的VMM相同的特权级别和地址空间中，以实现更高的效率，同时提出一种独特的相互保护机制来确保监视器的完整性。安全性分析表明，框架可以为事件驱动的VMM监视提供高度的保证，即使最高权限的VMM已完全被破坏。实验结果表明，该框架在执行事件驱动的监视策略时只会产生很小的性能开销，与以前的方法相比性能有了很大的提高。

该图1.4是框架的架构图，该框架的主要特点如下：基于事件驱动的ED-Monitor（与VMM同特权级别），不依赖定制的硬件设备，软件方法实现的TEE。该系统能保障

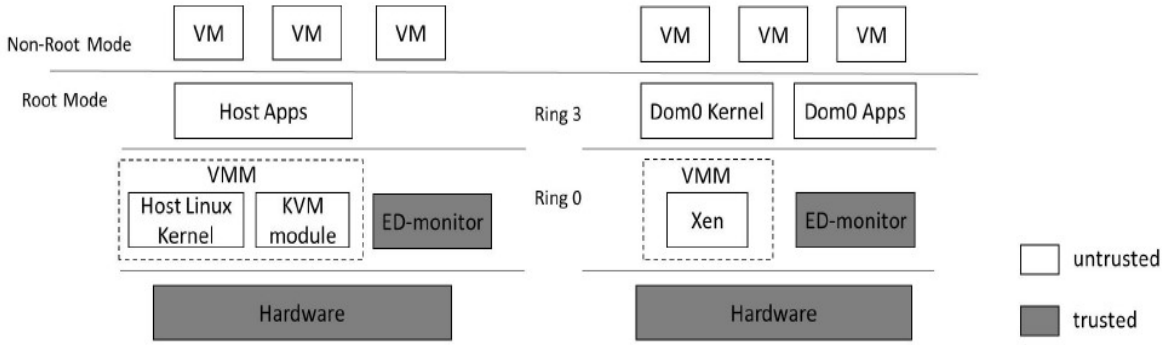

图 1.4 ED-Monitor架构图

内存完整性、控制流完整性。采用IPR（指令保护）和ASR（地址随机化）相互作用的方法，为ED-Monitor提供保护。性能好，对VMM无修改。

1.2.1.3 Secpod

操作系统内核对计算机系统的安全至关重要。许多系统已被提出以提高其安全性。这些系统的一个根本弱点是页表（控制内存保护的数据结构）没有与易受攻击的内核隔离，因此会受到篡改。为了解决这个问题，研究人员依靠虚拟化来实现可靠的内核内存保护。不幸的是，这种内存保护需要监视对客户页表的每次更新。这从根本上与硬件虚拟化支持的最新进展相冲突。本文提出了一种基于虚拟化的安全系统扩展框架Secpod，它既能提供强大的隔离性，又能与现代硬件兼容。Secpod[13]有两个关键技术：分页委派和审核内核对安全空间的分页操作；执行陷阱截获（受损的）内核企图通过误用特权指令来破坏Secpod。已经实现了一个基于KVM的Secpod原型。实验表明，Secpod既有效又高效。

正如图1.5所示，为了保证TEE的存在，在原先的Normal Space上创建了Secure S-pace，这是针对虚拟机进行创建，而不是宿主机内核。

在虚拟机上，安全工具只是运行在虚拟机的用户空间中，为了保证虚拟机空间的隔离性和安全性，需要对空间进行一定的防护。Secure Space 主要是通过在虚拟机的页表上开辟部分用户空间作为安全工具运行所用，这些工具并不在Normal Space中运行。为了保证安全性，当Secure Space活跃时，Normal Space的内核代码不可执行，防止对Secure Space中的数据和代码进行恶意访问（读写操作），如图1.6所示。

本论文学者对Secpod的性能进行了测试，与原系统Linux对比，性能开销相对较低，如图1.7所示。

1.2.2 硬件隔离技术

1.2.2.1 H-SVM

随着对云计算的需求不断增加，保护客户虚拟机（VM）免受恶意攻击者的攻击已

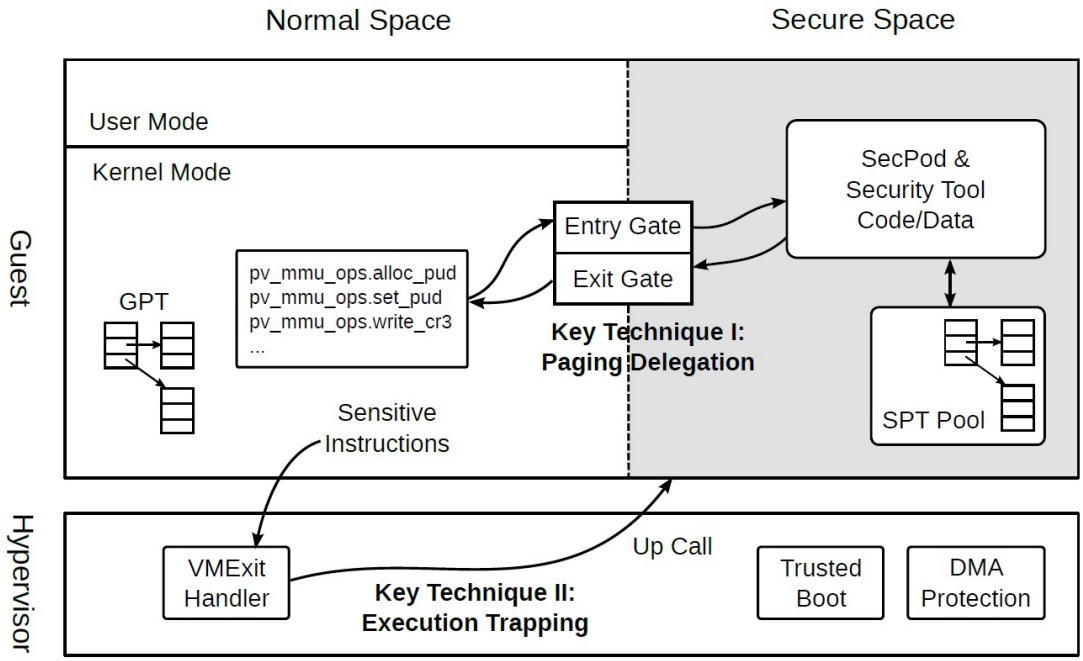

图 1.5　Secpod系统架构图

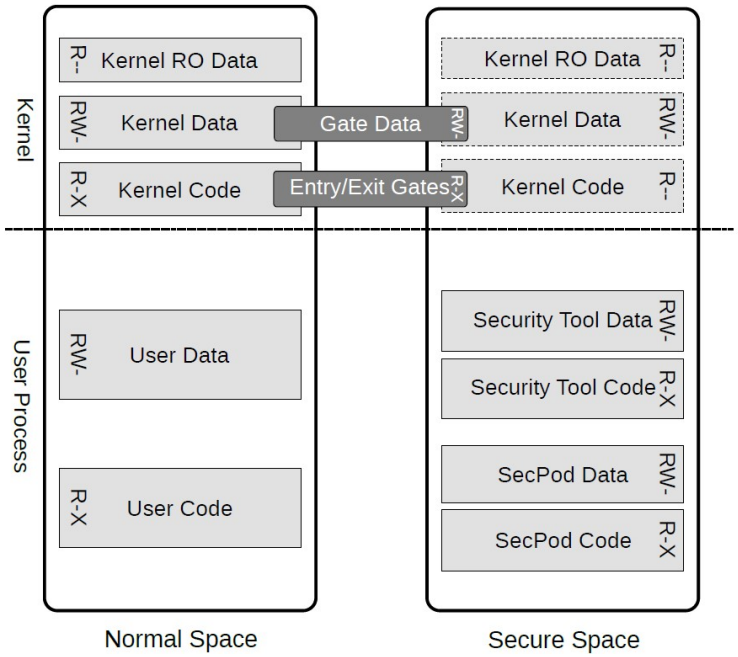

图 1.6　Secpod隔离空间布局图

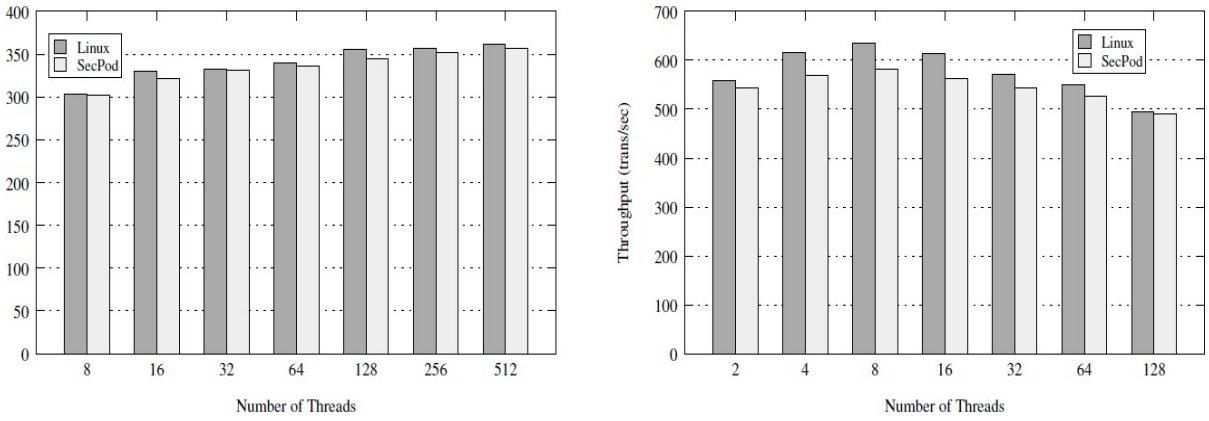

图 1.7 Secpod性能测试结果对比图

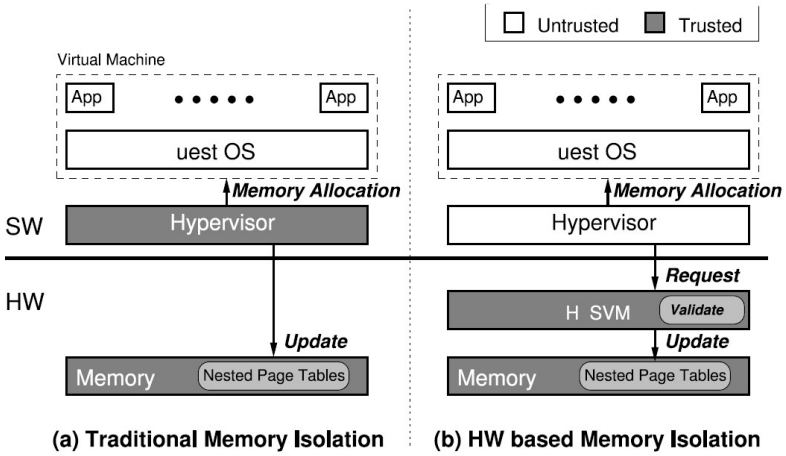

图 1.8 H-SVM系统架构图

成为提供安全服务的关键。当前的基于软件虚拟化的云安全模型依赖于软件管理程序及其具有根权限的值得信赖的管理员的无懈可击性。但是，如果将管理程序与远程攻击或根权限相冲突，则攻击者可以完全访问虚拟机的内存和上下文。本文提出了一种基于硬件的方法H-SVM[15]来保护客户虚拟机，即使在不受信任的管理程序下。通过这种机制，安全硬件提供了内存隔离，这比软件管理程序的脆弱性要小得多。该机制以较小的额外硬件成本扩展了当前基于嵌套分页的内存虚拟化硬件支持。虚拟机监控程序仍然可以灵活地将物理内存页分配给虚拟机，以实现高效的资源管理。除了安全虚拟化的系统设计外，本文还提出了一个使用系统管理模式的原型实现。虽然目前的系统管理模式不适用于安全功能，因而限制了性能和完整的保护，但原型实现证明了所提出设计的可行性。

H-SVM[15]主要是在硬件层和Hypervisor之间添加了定制的硬件，在内存数据访问的过程中，数据需要通过该定制硬件进行验证确保其正确性。访问主要是针对写操作。读操作无法篡改，篡改无意义，若在读取数据后篡改，数据必然会写回到内存，此时

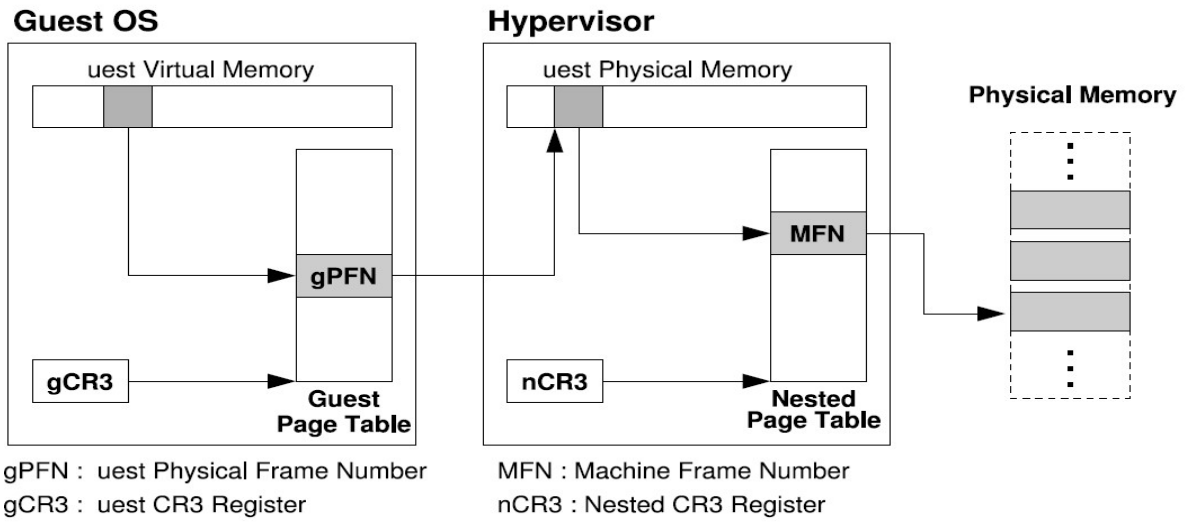

图 1.9　H-SVM地址翻译流程图

的数据验证无法通过，所以读访问篡改无意义。架构图如图1.8。

原先的操作流程是客户机请求访问自身虚拟内存，随后通过内存虚拟化技术，访问到Hypervisor的虚拟内存，最终转换机器页框号来访问位于物理上的内存数据。实现流程如图1.9。在使用H-SVM后，数据写操作流程会发生变化，只是对MFN上的数据提前进行验证，已确保数据的正确性，从而保证安全性。

1.2.3　重组Hypervisor

1.2.3.1　Nexen

该框架Nexen[12]由上海交大并行与系统安全实验室提出。 Hypervisor容易受到攻击。不幸的是，高效强化的Hypervisor具有挑战性，因为它们缺乏特权安全监视和分解策略。在这项工作中，Nexen系统地分析了Xen Security Advisories中的191个Xen虚拟机监控程序漏洞，发现大多数（144）位于核心虚拟机监控程序中，而不是dom0。然后，使用分析将Xen（称为Nexen）分解为一个安全监视器、一个共享服务域和每个VM Xen切片，这些切片由一个最低特权的沙盒框架隔离。使用嵌套内核架构实现Nexen，在xen地址空间内有效地嵌套自己，并通过为任意多个保护域添加服务以及动态分配器、数据隔离和跨域控制流完整性来扩展嵌套内核设计。其效果是，Nexen将基于虚拟机的管理程序限制在单个Xen虚拟机实例上，阻止已知Xen漏洞的74%（107/144），并强制Xen代码完整性（防御所有代码注入危害），同时观察可忽略的开销（平均1.2%）。总的来说，Nexen的独特定位是以最低的性能和实现成本提供管理程序强化的基本需求。

通过与其余几个保护系统的方案对比，如图1.10。Nexen针对Xen系统将Hypervisor进行划分，划分成为几个slices。但是对系统进行了改造。

当前，虚拟机最容易受到的攻击是DOS/DDOS攻击，为了避免VM受到攻击后可

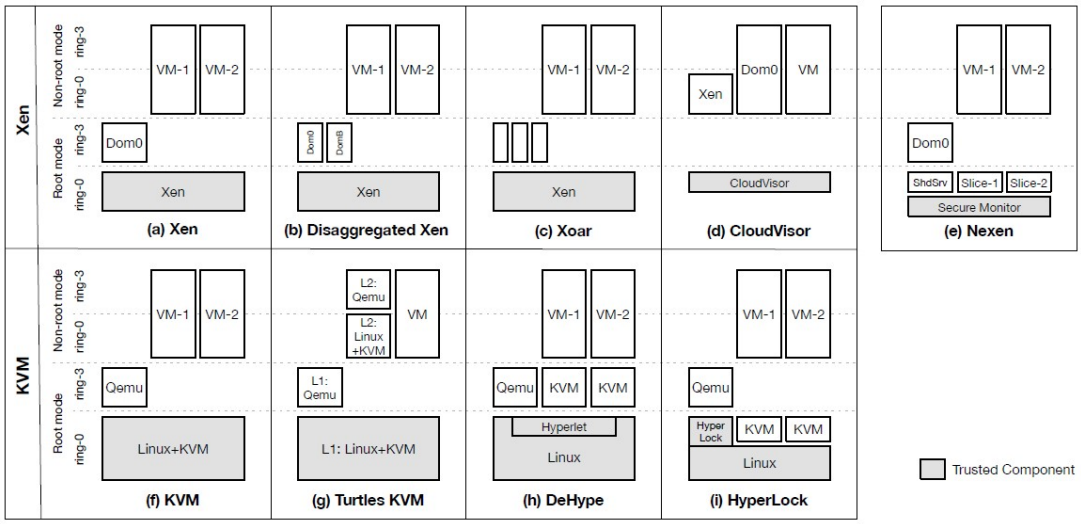

图 1.10 Nexen多方案架构对比图

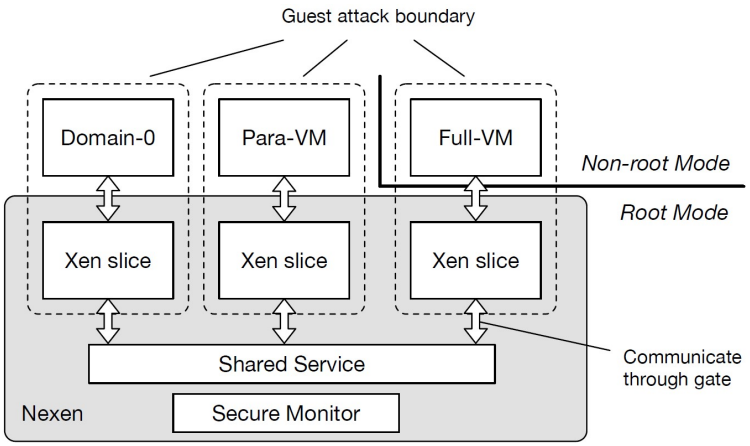

图 1.11 Nexen系统架构图

能对整个Hypervisor或者其余的VM造成威胁，论文学者提出一旦VM受到DOS攻击，则马上关闭该VM，阻止威胁的进一步发生。提出的Nexen框架结构如图1.11，通过对Hypervisor进行改造，将Hypervisor划分为三个部分，监控器、共享服务、以及针对每一个VM的Xen slice，监控器用于监控部分功能和服务，共享服务是所有虚拟机可能用到的共同服务，其余的功能会针对每一台VM进行实现。当攻击发生时，立即关闭该VM，并不影响其余的VM，也不影响Hypervisor为其余VM继续服务。

为了保证区域的安全性，下级不得访问上级区域的内容。各个组件访问方式如图1.12。

实验效果：整个系统可以实现的攻击防护目标如图1.13。主要是DOS攻击，权限提升（访问控制策略）以及信息泄露。

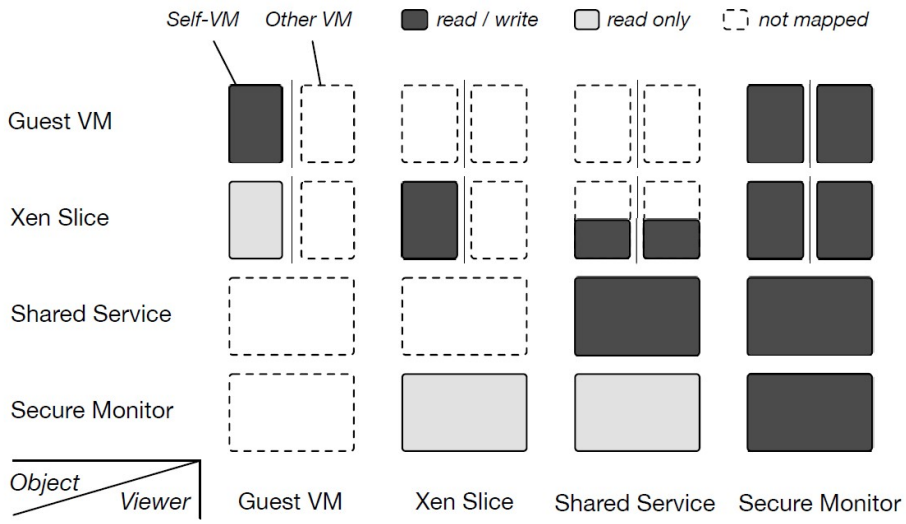

图 1.12　Nexen内部读写访问图

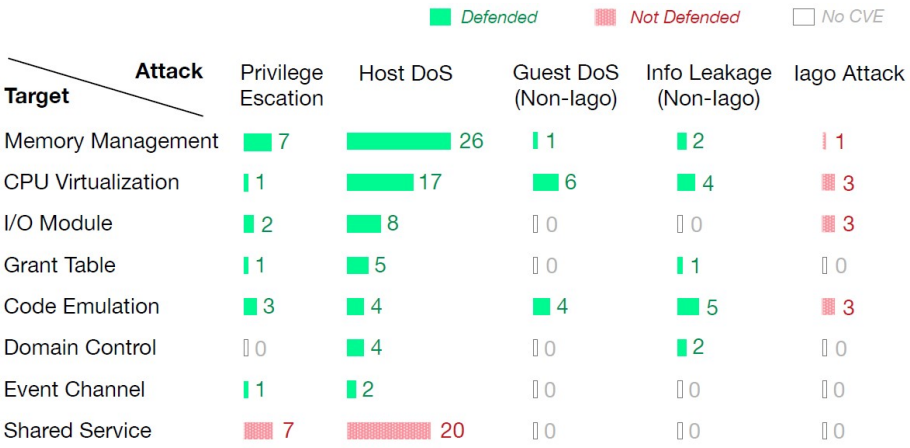

图 1.13　Nexen攻击防御图

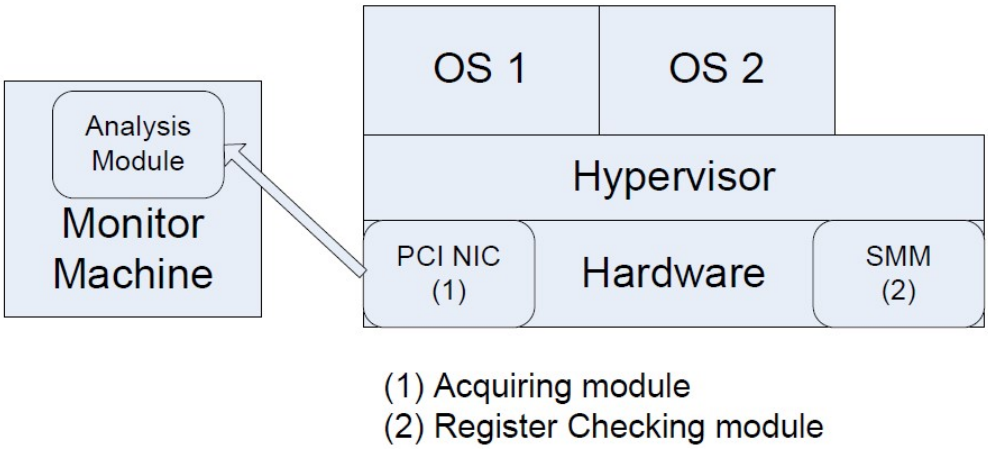

图 1.14 HyperCheck系统架构图

1.2.3.2 Nohype

云计算是一种颠覆性的趋势，它正在改变我们使用计算机的方式。云基础设施中的关键底层技术是虚拟化——很多人认为虚拟化是关键功能之一，而不仅仅是实现细节。不幸的是，虚拟化的使用是一个重要的安全问题的根源。由于多个虚拟机运行在同一台服务器上，并且由于虚拟化层在虚拟机的操作中发挥了相当大的作用，恶意方有机会攻击虚拟化层。成功的攻击会让恶意方控制所有强大的虚拟化层，可能会损害任何虚拟机的软件和数据的机密性和完整性。在本文中，建议删除虚拟化层，同时保留虚拟化所支持的关键特性。Nohype[14]体系结构（命名为表示虚拟机监控程序的删除）处理虚拟化层的每个关键角色：仲裁对CPU、内存和I/O设备的访问，充当网络设备（如以太网交换机），以及管理客户虚拟机的启动和停止。此外，还表明，Nohype 体系结构可能确实是“没有炒作”，因为实现Nohype体系结构所需的几乎所有功能目前都作为处理器和I/O设备的硬件扩展提供。

1.2.3.3 HyperCheck

在过去的几年里，虚拟化已经被应用到从人口密集的云计算集群到家庭桌面计算机的各种环境中。安全研究人员将虚拟机监视器（VMMs）作为一种新的机制来保证对不受信任的软件组件的深度隔离[21]。不幸的是，他们的广泛采用促使虚拟机成为攻击者的主要目标。在本文中，提出了一种硬件辅助篡改检测框架HyperCheck，旨在保护VMM的完整性，并针对某些类型的攻击，提供底层操作系统（OS）。HyperCheck利用x86系统中存在的CPU系统管理模式（SMM）安全地生成受保护机器的完整状态并将其传输到外部服务器。使用HyperCheck，能够找到针对Xen管理程序和传统操作系统完整性的rootkit。此外，对于旨在禁用或阻止其操作的攻击，HyperCheck非常强大。实验结果表明，超检测可以在40毫秒以内产生和传输受保护软件的状态扫描。

为了实现针对Hypervisor的完整性保护，在硬件层获取硬件信息，然后在SMM中进

行寄存器检查，最后在监控机器中进行分析。整个的架构和数据访问过程可以参看系统架构图1.14。

1.2.3.4 HyperSentry

本文介绍了一个新的框架HyperSentry，用于对正在运行的Hypervisor（或系统中任何其他最高特权软件层）进行完整性测量。与现有的保护特权软件的解决方案不同，HyperSentry不会在完整性度量目标下引入更高特权的软件层，也不引入类似于ISO-X[26]中的定制硬件。相反，HyperSentry引入了一个软件组件，该组件与虚拟机监控程序正确隔离，以实现对虚拟机监控器运行时状态完整性的隐藏和上下文测量。以确保受损的管理程序在检测到即将到来的测量时没有机会隐藏攻击痕迹，隐藏测量是必要的。上下文测量是必要的，以验证所有输入数据的的完整性。

HyperSentry使用带外信道（如服务器平台上常用的智能平台管理接口（IPMI））触发隐藏测量，并采用系统管理模式（SMM）保护其代码基和关键数据。HyperSentry的一个关键贡献是克服了SMM的局限性，为完整性度量代理提供了（1）可供管理程序使用的相同上下文信息，（2）Hypervisor完全受保护的执行，以及（3）对其输出的完整性验证。该框架基于Xen实现，性能开销低。

1.2.3.5 HyperSafe

虚拟化在当今的计算系统中被广泛采用。它在隔离和反省作为虚拟机（VM）的商品操作系统方面的独特安全优势使其能够应用广泛。然而，一个常见的基本假设是存在一个值得信赖的管理程序。不幸的是，普通商品Hypervisor的大量代码库和最近成功的Hypervisor攻击（例如，VM逸出）严重质疑了这种假设的有效性[27]。在本文中，提出了一种轻量级的方法，使现有的I型裸机管理程序具有独特的自我保护能力，以提供终身控制流完整性。具体来说，提出了两个关键技术。第一个是不可绕过的内存锁定，它可以可靠地保护虚拟机监控程序的代码和静态数据，即使存在可利用的内存损坏错误（例如缓冲区溢出），也不会受到影响，因此成功地提供了虚拟机监控程序代码的完整性。第二个是受限指针索引，它引入了一个间接层，将控制数据转换为指针索引。这些指针索引受到限制，因此相应的调用sgx/返回目标严格遵循管理程序控制流图，从而扩展保护以控制流完整性。已经构建了一个原型并使用它来保护两个开源的Type-i管理程序：Bitvisor和Xen。通过对虚拟机监控程序的综合开发和基准测试程序的实验结果表明，Hypersafe能够可靠地实现虚拟机监控程序的自我保护，并以较小的性能开销提供完整性保证。

Rootkit主要是通过修改内核的控制结构和钩子来隐藏自身，原先系统中的钩子和控制结构是零散的、分散的，Hooksafe通过将钩子和控制结构进行整合，利用重定向方法，将钩子集中在一起，并验证对钩子的写操作。许多研究工作通过Hypervisor来实现

图 1.15 HyperSafe页表访问流程图

对rootkit的监控[28]。该系统的功能效果如图1.15，对于只读页表，具有写保护的页表能够阻止恶意操作的访问。

1.2.3.6 Hilps

特权分离一直被认为是软件设计中的一个基本原则，以减轻安全攻击的潜在危害，尤其在一体化的操作系统设计中[29]。在开发各种特权分离方案的过程中，已经付出了很大的努力，其中一个单块操作系统或管理程序被划分为两个特权域，其中一个域在逻辑上比另一个域更具特权，即使这两个域在相同的处理器特权级别上运行。如果某个特权级别的软件在没有更多特权软件参与或帮助的情况下实现了特权分离，那么特权分离就是内部级别。一般来说，实现层内特权分离要求开发人员依赖底层硬件的某些安全特性。然而，到目前为止，这些开发工作并没有像英特尔x86系列那样专注于ARM体系结构，主要是因为ARM安全功能的体系结构提供相对不足。与x86不同，因此，不存在可普遍应用于ARM上任何特权级别的完整内部级别方案。然而，随着恶意软件和攻击的增加针对几乎每一级的特权软件，包括操作系统、管理程序，甚至是受TrustZone保护的最高特权软件，开发一种称为Hilps[30]的技术，以在所有这些级别的ARM特权软件中实现真正的级内特权分离。Hilps成功的关键是支持ARM最新64位体系结构（称为TXSZ）的新硬件功能，操纵它来弹性地调整程序的可访问虚拟地址范围。在实验中，应用Hilps将特权分离的核心软件机制改造为现有的系统软件，并评估了结果系统的性能。实验结果表明，该系统的平均开销不到1%，因此，得出结论，Hilps在实际部署中具有很好的应用前景。

该框架主要通过创建两个不同的区域——内部域和外部域，不同的区域处理不同的权限的功能事件，外部域是原先普通的内核区域，内部域用来处理一些有关于敏感信息访问和函数操作的功能，这样不同的区域处理不同特权功能函数，两个区域相互隔离，从而实现特权分离。

基本功能如图1.16，在原外部区域中对某些关键函数进行挂钩，当这些事件发生时，会第一时间跳转到内部区域去处理，这些事件一般包括页表管理、系统控制寄存

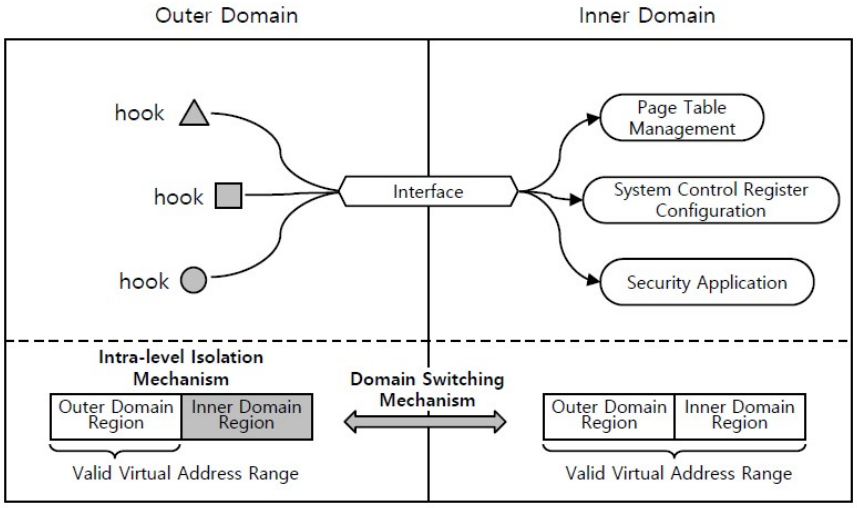

图 1.16　Hilps特权分离原理图

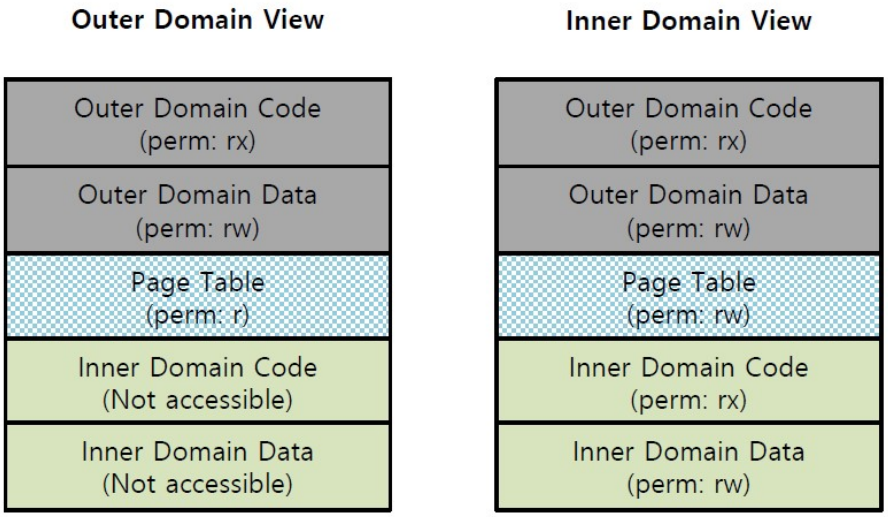

图 1.17　Hilps地址空间布局图

器、安全应用处理函数，处理结束后再跳转回到外部区域。

两个区域的实现主要是创建不同的地址空间，如图1.17。不同地址空间的互访是通过Interface这唯一通道完成。

1.3　研究内容

虚拟化技术发展很快，越来越多的功能逐渐添加到Hypervisor中，其代码量也逐渐增大。到目前为止，内核2.6.36.1 中XEN 和KVM 的代码量分别达到30万行和33600行，当然，越来越多的代码量，也就意味着它会存在着许多有漏洞的地方，更容易被攻击的地方。根据CVE 漏洞网站相关的数据，从2004 年到现在，有357个和XEN 相关的漏洞，180个和KVM 相关的漏洞，比如，CVE-2009-2287在提权后，用于加载恶意的EPT页

表。CVE-2018-1087是个高危漏洞，攻击者可以利用它进行提权来攻击Hypervisor，从而对云平台上的租户进行攻击。因为云平台上的多租户共享物理资源，其中一种主要的资源就是物理内存，当多租户中某一租户被攻击后，那么很可能继而发生跨域攻击。针对此，本文提出了同层地址空间隔离下的虚拟机内存保护技术。本文的研究目标是提供一个在云环境非可信Hypervisor下，通过同层地址空间隔离技术创建TEE，同时能够保证虚拟机内存安全的技术。该技术是一项对平台没有依赖性、对云平台提供商可以广泛使用的技术。

本文的研究内容主要是在X86平台上KVM监控器中实现对虚拟机的运行时关键数据进行监控以及隔离虚拟机之间的内存的框架HyperMI，尽可能减弱性能逃逸攻击、虚拟机跨域攻击等造成的虚拟机信息泄露。该方法是基于软件实现的，相比基于硬件方法实现的虚拟机安全防护[15,31]，其优点在于可移植性强，适用的平台广；相比于基于软件的大规模修改Hypervisor的方法[12,14],本方法对系统的修改小，适用于商业平台，作为内核模块可加载。

该框架主要包含三部分：HyperMI World、VM Monitoring、VM Isolation，即区别于原系统的安全执行环境、虚拟机和Hypervisor交互关键数据监控（简称“交互数据监控”）、虚拟机内存高强度隔离（简称“VM内存隔离”）。

其技术难点如下：

- 提供不依赖系统更高特权层或者定制硬件的安全可信执行环境，通过同层地址空间隔离技术使得该执行环境与Hypervisor处于同一特权级别。
- 提供对虚拟机与Hypervisor关键交互数据的不可绕过监控，主要是VMCS（虚拟机控制结构）和EPT（硬件扩展页表）这两类数据结构，阻止关键数据泄露，为VM内存隔离提供一部分安全防护。
- 提供虚拟机之间、与Hypervisor之间的高强度内存隔离技术，通过物理页跟踪技术阻止虚拟机之间的内存越权访问攻击，阻止内存信息泄露攻击，例如地址重映射攻击、地址多映射攻击。
- 实现在X86平台上针对KVM的系统，实验测评表明其性能开销相对较低，安全性较高。

该系统的组成分为传统的Hypervisor和虚拟机安全套件，以及切换门。切换门，用于两个环境进行切换，能够保障安全性，环境切换过程不可绕过，不可伪造，不可中断。虚拟机安全套件，由安全执行环境，交互关键数据监控，VM内存隔离组成。用于提供对虚拟机运行时状态信息的隔离保护、对虚拟机内存高强度隔离保护，从而阻止用户隐私泄露问题。

1.4 组织结构

文章的基本组织分为七章：第一章绪论简要介绍课题的研究现状、研究背景和研

究内容。第二章主要讲HyperMI的基本设计，第三章、第四章、第五章主要针对系统的三个组件部分进行详细设计和实现描述，第六章对系统的整体性能测评和安全性能进行详细分析。最后一章是研究生生活的结论与展望。

第二章　HyperMI设计

2.1　威胁模型

一个攻击者可以从远程或者本地的方式对Hypervisor进行攻击，针对不同的虚拟机架构实现不同的攻击，可能先攻击宿主机、再攻击Hypervisor[27],[32],[15]。如图2.1，远程攻击者主要是网络攻击者，先通过攻击多租户中的一台虚拟机，攻击方式主要是通过攻击虚拟机本身的系统漏洞或者利用其上应用软件的漏洞攻陷虚拟机，随后通过虚拟机跨域攻击的方式提权到Hypervisor。针对本地攻击者，攻击方式主要是通过利用Hypervisor或者宿主机本身的系统漏洞，随后拿到Hypervisor的处理权限。整个Hypervisor被攻击的过程基本是这样的。当Hypervisor被攻击后，攻击者则可以对交互关键数据和虚拟机进行攻击，从而获取相应的敏感信息。

攻击者主要从本地或远程的方式攻陷Hypervisor，随后可以进行各种攻击，主要有两种攻击，一种是攻击者篡改虚拟机和宿主机进行上下文切换时的交互关键数据，主要包含系统控制寄存器等，会造成系统安全性低，系统敏感数据泄露攻击威胁。另一种攻击是针对虚拟机的内存越界访问攻击，例如多映射攻击和双映射攻击，造成内存中敏感信息泄露。攻击方式主要是在实现虚拟机逃逸后进行虚拟机同驻攻击，最终可能被利用造成规模更大的DDOS攻击[33]。详细的攻击方式如下介绍。

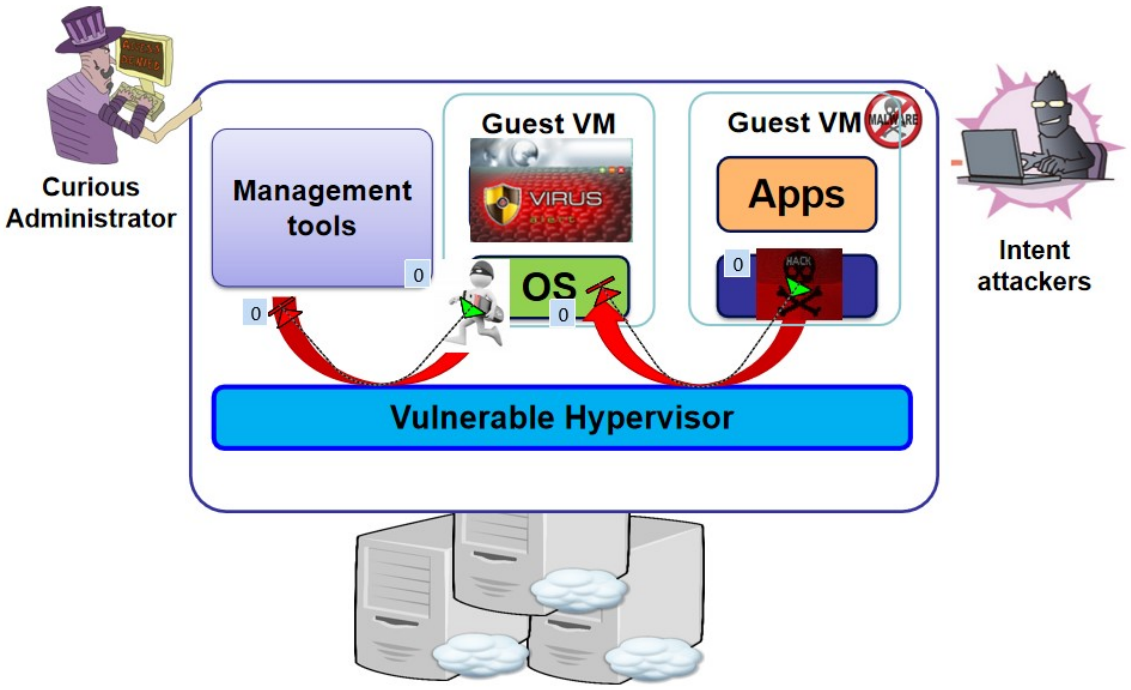

图 2.1　威胁模型

2.1.1 Hypervisor完整性攻击

远程攻击者和本地攻击者都可以攻击Hypervisor。远程攻击者可以通过同驻攻击和跨域攻击的方式攻陷Hypervisor，主要是利用虚拟机上软件漏洞先攻击其中的一台虚拟机，随后再通过逃逸的方法攻击Hypervisor[34],[35]。本地攻击者可以直接利用Hypervisor本身的漏洞直接攻击Hypervisor。

2.1.2 交互数据攻击

当虚拟机和宿主机进行交互，产生虚拟机退出和虚拟机进入。采用分时策略，交互最终可以实现一个处理器上运行虚拟机或者宿主机，主要是通过虚拟机上下文切换实现。其中，上下文主要保存在VMCS，VMCS中包含虚拟机和宿主机的重要信息，以便上下文正常切换，处理器获取这些重要信息后能够正常运行虚拟机/宿主机。虚拟机退出时，虚拟机的特权寄存器等信息重新保存到VMCS，宿主机的信息被从VMCS中加载，随后宿主机正常运行。虚拟机进入时，宿主机的特权寄存器等信息重新保存到VMCS，虚拟机的信息被从VMCS中加载，随后虚拟机正常运行。攻击者篡改交互数据，篡改虚拟机地址映射页表地址（EPTP），导致加载恶意页表；篡改虚拟机/宿主机的RIP，导致控制流劫持攻击等；篡改其它数据，导致加载错误的信息；泄露VMCS中的系统关键信息。

2.1.3 内存越权访问攻击

虚拟机的地址映射需要两套页表，自身系统中的一套页表和宿主机管理的扩展页表（EPT）。EPT作为一套页表，包含GPA与HPA的地址映射关系。攻击者攻陷Hypervisor之后，可以通过访问VMCS获得EPT的地址EPTP，随后可以直接访问EPT，更改EPT中的地址映射，造成内存恶意访问攻击，例如多重映射、双映射。

多重映射（remap）：参见图2.2,假设两台虚拟机VM1和VM2，VM1作为远程攻击者使用的虚拟机，VM2作为受害者虚拟机。当VM2使用的物理页A被使用完释放后，VM1会重新映射到这个物理页A，那么会导致VM1可以访问物理内存页A上的信息，造成信息泄露攻击。

双映射（double map）：假设两台虚拟机VM1和VM2，VM1作为远程攻击者使用的虚拟机，VM2作为受害者虚拟机。VM1使用虚拟地址C1、C2，对应的物理地址是D1、D2，VM2使用虚拟地址C3、C4，对应的物理地址是D3、D4。攻击者去修改C1对应的物理地址，通过更改C1对应VM1的EPT页表的最后一级，使得最后一级指向物理页D3，那么最终会造成虚拟地址C1对应的物理内存页是D3，而不是D1。这样可以访问D3上的数据，从而实现了用户数据泄露攻击。

2.2 假设

第一，假设使用的物理资源是可信的，包含处理器、总线、IO传输等用于数据传

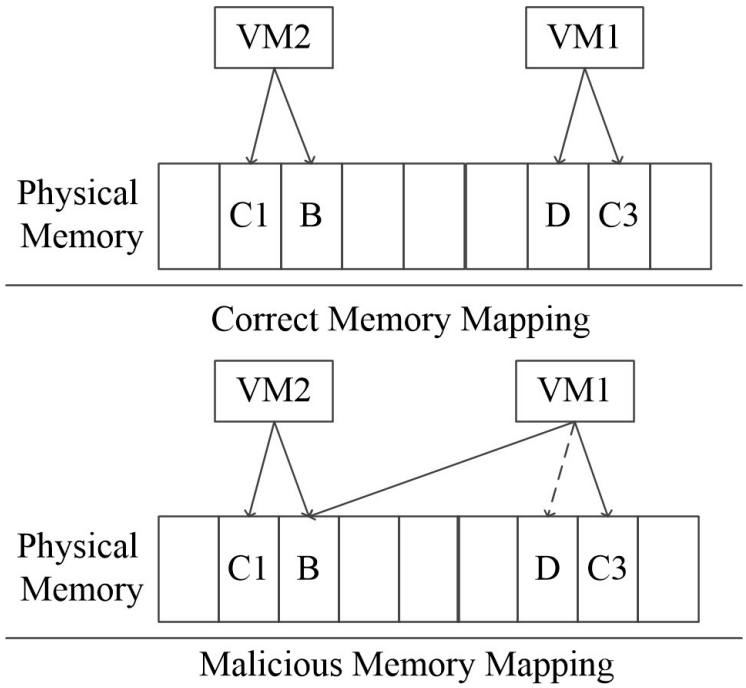

图 2.2 重映射攻击

输的硬件是安全的。

第二，假设存在可信的启动，在启动过程中整个操作系统并不受到恶意的攻击。

第三，假设虚拟机自身的软件漏洞导致的内存数据泄露问题并不考虑。

第四，不考虑攻击者实施DOS攻击，侧信道攻击，基于硬件的攻击（冷启动攻击或者Rowhammer攻击）。针对Hypervisor的DOS攻击会导致Hypervisor无法正常运行，VM所依赖的Hypervisor无法给VM提供资源管理服务，整个系统都会停止运行和服务。侧信道攻击可以导致内存信息泄露，不考虑这样的攻击方式。

2.3 架构

在介绍了威胁模型和假设后，本小节将开始描述HyperMI在X86平台上的总体架构设计，虚拟机监控器主要以KVM为主，一种全虚拟化平台。

2.3.1 系统架构

HyperMI作为一个虚拟机内存隔离技术的系统实现，它基于同层地址空间隔离技术创建的可信执行环境，结合挂钩技术，虚拟机内存物理页的跟踪技术，实现对虚拟机内存的高强度隔离，实现对虚拟机和Hypervisor关键交互数据的监控，实现对虚拟机上敏感信息的安全保护。

如图2.3所示，HyperMI的整体布局分为两个部分。在传统的系统架构中，主要有三层，硬件层、Hypervisor虚拟化层（root层）以及虚拟机层（非root层）。在HyperMI系统中，与传统系统的区别在于root层，root层主要由三部分组成，原Hypervisor（简

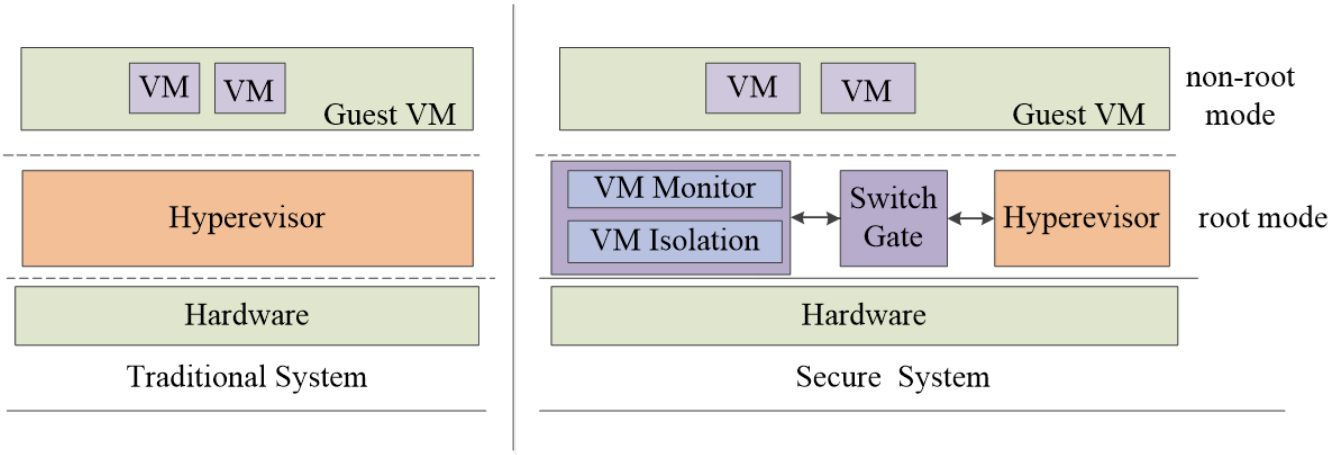

图 2.3　系统总体架构图

称“Hypervisor”)、同层地址空间隔离技术创建的可信执行环境HyperMI World（简称“HW”)、两组件的交互通道Switch Gate（简称“SG”)。

HW：在实际中，云平台提供商对系统中性能开销低最难以接受，系统的性能开销低会影响用户的流畅使用体验，从而影响系统的销售和用户的实际使用。已有部分技术通过定制的硬件来对敏感数据进行保护，或者使用系统中的高特权级别的软件层实现对敏感数据的保护，阻止恶意部件的访问。受SKEE等研究的影响，创建同层地址空间隔离的技术来避免定制硬件和高于Hypervisor特权级别的弊端，该技术可以创建TEE，暂时称为HW。其与Hypervisor是处于不同的地址空间，它们之间是相互隔离的。其能够保证一些关键的数据和数据操作函数访问的机密性和完整性。如图所示，将VM Monitor和VM Isolation放在HW中执行。

SG：如图所示，两个地址空间是相互隔离的，但是总有一些数据是需要进行交互的，那么HW与Hypervisor进行交互的唯一通道就是SG。从Hypervisor跳到HW和从HW返回到Hypervisor都是通过SG，为了阻止恶意切换、恶意绕过、恶意破坏HW的隔离性，SG能够保证两个不同地址空间的安全切换，保证切换的原子性、准确性、不可绕过性。

VM Monitor：虚拟机监控模块的主要功能是监控Hypervisor与虚拟机的交互关键数据访问过程的正确性，该监控是事件驱动的，不是事件轮询的（每隔一定的时间周期对需要监控的时间进行验证监控)，能够避免轮询间隔中检测不到的攻击威胁（攻击者可以在轮询时刻之外的时间段进行攻击)。监控的关键数据是VMCS和EPT，EPT可用于客户机物理地址（GPA）到宿主机物理地址（HPA）的转换，实际上是一套页表。EPTP可以存放EPT的页表入口地址，类似于CR3寄存器，攻击者可以伪造EPTP、EPT，甚至篡改EPT中的地址映射从而导致攻击。VMCS是在VMX操作使用来管理Hypervisor和虚拟机的行为。尤其当VMCS中的一些特殊数据被篡改，例如客户机状态信息、虚拟机执行域中的关键寄存器信息，虚拟机会受到不可控的威胁。

VM Isolation：虚拟机内存隔离技术可以阻止恶意的虚拟机访问和Hypervisor越界访问攻击，尤其是地址重映射、地址多映射攻击。首先，HyperMI使用页标记技术标记了每一个物理页，保证每一个物理页只能被一台设备访问，一台虚拟机或者Hypervisor。其次，它剥夺了Hypervisor原先的地址翻译能功能，来确保物理页被分配或者映射时同时确定物理页的属主。最后，为了避免内存越界访问攻击，当发生缺页情况时，进行EPT更新时，确定物理页的属主的唯一性防止发生多映射攻击。为了避免地址重映射攻击，当物理页被使用完清除时，通过清除该物理页上的信息内容，保证物理页上的信息不会被在页释放后恶意访问，完全阻止地址重映射攻击。

HyperMI系统的主要特点是通过监控虚拟机与宿主机唯一交互通道，监控虚拟机的地址映射，将交互和地址映射放在创建的隔离地址空间中进行处理，从而使得非可信的Hypervisor无法直接访问重要的交互数据、无法直接访问虚拟机地址映射使用到的页表，阻止攻击者进一步实施破坏虚拟机运行时状态数据攻击和内存多映射、双映射攻击。基本实现方案是通过对原Hypervisor系统VM与HOS交互、VM地址映射设置挂钩，进而实现监控。代码是通过使用LKM模块方式实现。

系统执行流程：加载虚拟机安全套件系统模块（LKM方式），开启虚拟机；通过挂钩的方式监控原Hypervisor系统中VM与HOS的交互操作，监控VM地址映射；当所监控事件发生时，判断是VM地址映射还是交互操作，随后切换到新的地址空间中进行VM地址映射或者交互操作；当处理完成后，需进行环境切换跳回到原Hypervisor去执行；此过程可以不断循环，如若关闭系统，先关闭虚拟机，随后卸载虚拟机安全套件系统模块，结束系统运行。

2.3.2 子系统间关系

虚拟机安全套件系统包含三个模块，地址空间隔离、交互关键数据监控、VM内存隔离，为虚拟机的创建、运行、销毁提供安全服务。

地址空间隔离，采用同层隔离的方法实现了地址空间隔离，创建与Hypervisor同层的隔离执行环境，目的是减少性能开销，同时增强安全性。当虚拟机安全套件运行在一个相对安全的隔离的执行环境中，可以免受非可信虚拟化层的攻击威胁，提供对虚拟机安全组件的保护。针对隔离执行环境中使用到的特权寄存器，MMU和DMA，当然要设置一些安全策略对安全隔离空间进行保护，包括监控特权寄存器访问、监控对MMU的访问，监控I/O访问地址，避免安全隔离地址空间受到攻击，从而绕过安全监控甚至破坏安全隔离空间的完整性。

VM内存隔离，为了使得VM的内存资源隔离，需要在地址映射的时候对内存资源进行隔离。同时对地址映射进行监控，保护关键的数据结构，防止跨域攻击从而泄露免敏感数据。

交互关键数据监控，监控的主要内容是Hypervisor 和VM 之间的交互关键数据，即

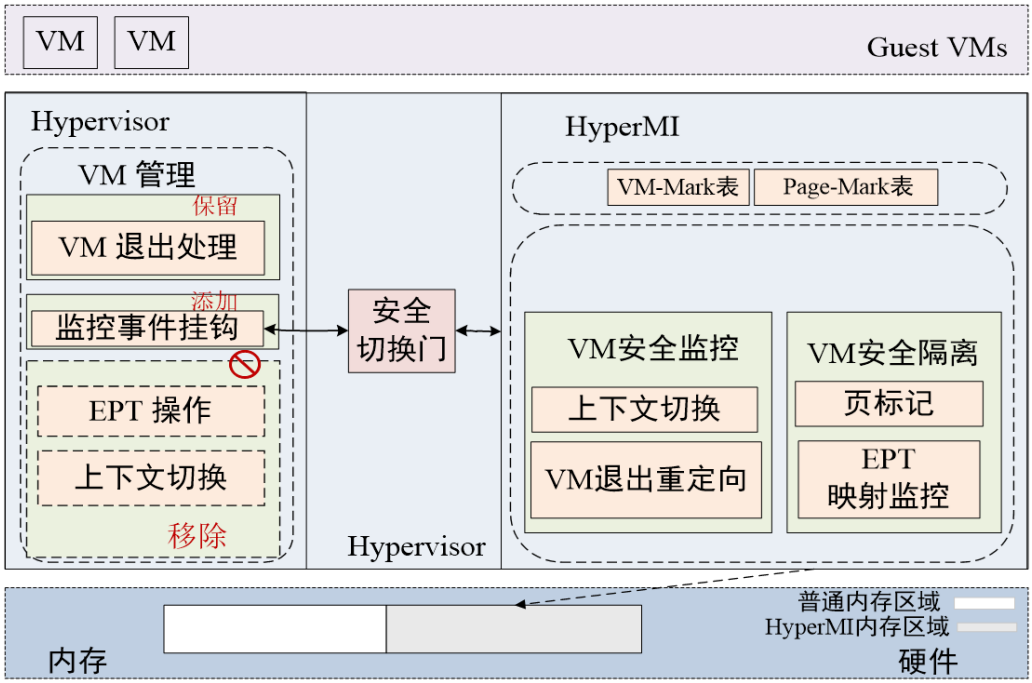

图 2.4　系统架构图

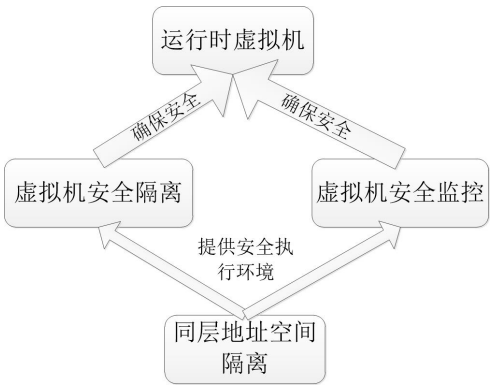

图 2.5　子系统交互关系图

切换上下文（VMCS），分为监控虚拟机上下文切换和虚拟机退出处理模块。

系统基本架构：在宿主机内核层创建新的地址空间（HyperMI World），与原Hypervisor（称为Hypervisor）通过安全门进行交互，安全切换门作为交互的唯一通道，不可绕过，且有一定的安全性。 Hypervisor中部分功能会被保留、添加或者移除。其中，虚拟机退出事件处理功能被保留，挂钩监控功能被添加，将虚拟机地址映射（EPT操作）和交互操作移除。Hypervisor中添加挂钩来监控VM与HOS的切换、监控VM的地址映射。在HyperMI World中有虚拟机安全套件系统使用的关键数据（Mark表），交互监控包含上下文切换和退出重定向，地址映射监控包含物理页动态标记与跟踪和EPT地址映射监控。该隔离的地址空间运行过程中产生的代码段和数据段在HyperMI内存区域中，该区域不可被Hypervisor随意访问，参见整个系统架构图2.4。

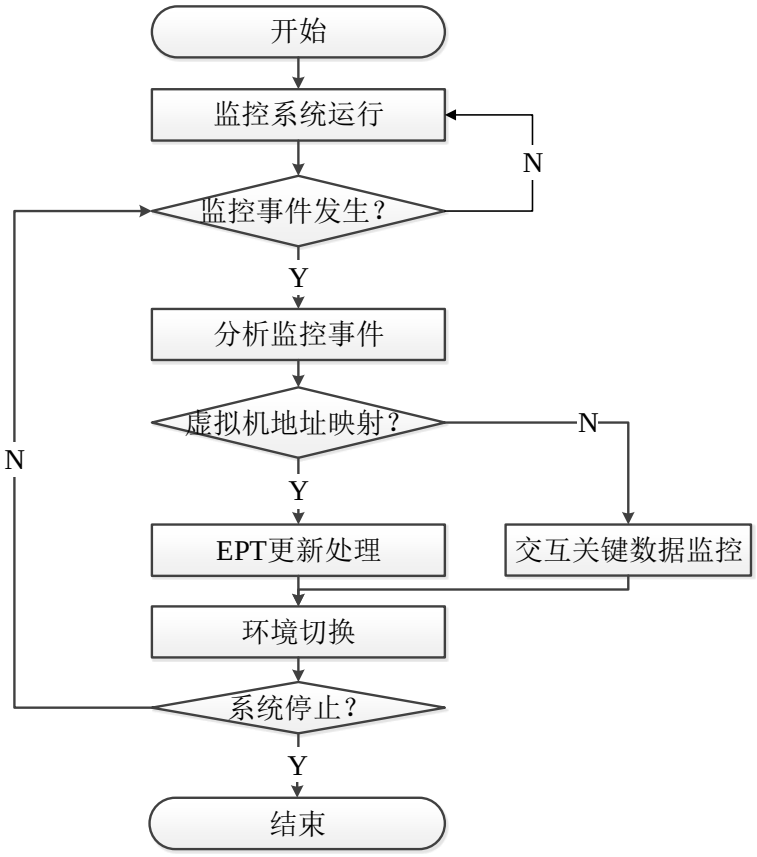

图 2.6 控制流程图

2.3.3 系统流程

地址空间隔离提供了一个安全的隔离的地址空间，虚拟机安全映射和虚拟机与宿主机安全切换是运行在隔离的地址空间中，切换门是用来使得原来Hypervisor和虚拟机安全套件进行切换的接口。这三个子系统存在一定的关系，地址空间隔离模块为交互监控模块和虚拟机地址映射监控模块提供安全隔离的执行环境。交互模块和虚拟机地址映射模块为虚拟机的运行提供安全保护。其中，交互模块为虚拟机运行时状态信息提供保护，保证虚拟机的正常安全运行；虚拟机地址映射模块为虚拟机内存提供安全隔离和访问控制，三个子系统模块之间的关系如图2.5所示。

加载虚拟机安全套件系统模块（LKM方式），开启虚拟机；通过挂钩的方式监控原Hypervisor系统中VM与HOS的交互操作，监控VM地址映射；当所监控事件发生时，判断是VM地址映射还是交互操作，随后切换到新的地址空间中进行VM地址映射或者交互操作；当处理完成后，需进行环境切换跳回到原Hypervisor去执行；此过程可以不断循环，如若关闭系统，先关闭虚拟机，随后卸载虚拟机安全套件系统模块，结束系统运行。控制流程如图2.6。

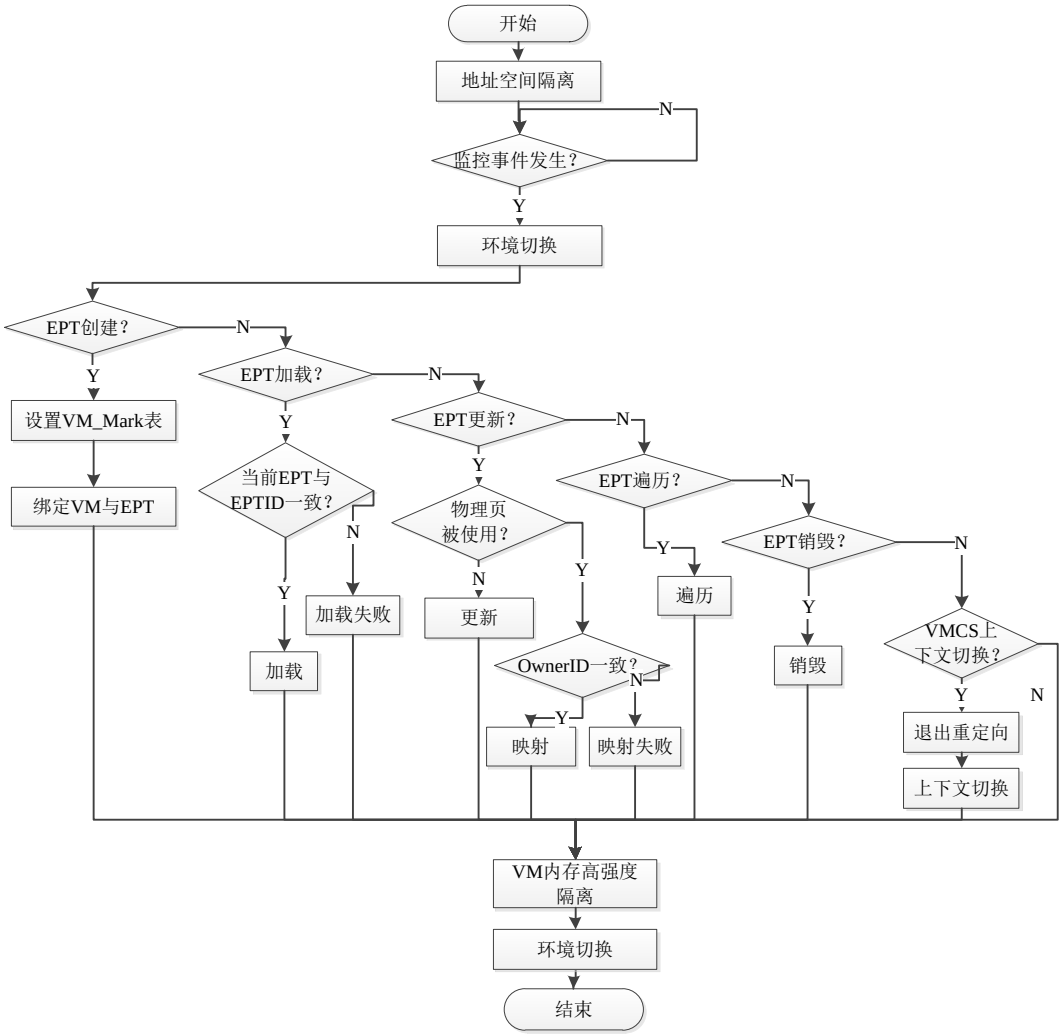

图 2.7　监控事件执行流程图

2.3.3.1 监控事件

通过挂钩的方式在Hypervisor监控交互过程和虚拟机地址映射中使用。交互关键数据主要是指VMCS，需要将VMCS放在新地址空间HyperMI World（HW）中；因Hypervisor会有一些函数访问到VMCS，同样这些函数会被移到新地址空间HyperMI World中，这样监控内容会增加；那么有关VMCS主要监控包括VMCS读，写，清除过程。虚拟机地址映射监控的主要是关键数据EPT，EPT是虚拟机地址映射表，其中包含了虚拟机物理地址到宿主机物理地址的映射。为了保护EPT，需要将EPT放在HW中；同样类似于VMCS，EPT作为地址映射中的关键数据，Hypervisor中会有许多函数直接访问EPT去完成系统功能，当EPT被放在HW中时，这些函数就无法直接访问，所以需要将EPT访问的相关函数防在HW中执行，这样监控的内容增加；那么有关EPT主要监控的关键数据包括EPT本身、EPT创建函数、EPT遍历函数、EPT销毁函数、EPT更新函数。这些关键数据的监控方式是通过在Hypervisor中设置hook，在函数入口设置跳转指令，导致这些函数一旦被执行，会通过安全切换门跳转到HW中去执行另外一些函数（功能相同）。从而达到实时监控的目的。

2.3.3.2 分析监控事件的过程

整个操作系统安全启动后，整个系统会被划分为两个区域，原执行环境和隔离的执行环境。监控系统中的关键事件，当这些事件发生的时候，会到隔离的地址空间中运行。分别对事件进行判断，判断依据是EPT是否创建、加载、更新、遍历、销毁、缺页、VMCS上下文环境是否切换，随后做出相应的事件处理。处理结束之后，切换回到原运行系统，当模块被卸载的时候，整个系统运行结束。实现流程如图2.7。

2.4 小结

本章首先介绍了HyperMI的威胁模型，随后提出了该框架的基本设计原则与要求，定义了一些不同于其它现有的虚拟机内存隔离技术而需要解决的一些新的问题。这些问题主要是解决云平台提供商来考虑到的性能开销问题、系统可移植性问题、平台不依赖性问题。随后，介绍了假设和框架架构。这一框架分为两个模块，它们分别是HW、SG，在HW中，又会进行VM 监控和VM隔离，HW与监控隔离各自实现自己的功能，又相互依赖，共同合作，最终构成一个可信的虚拟机内存安全防护框架HyperMI。

这一基本框架的设计遵循在之前提出的基本原则和要求，为了能更好地理解HyperMI的架构和设计，将在第三章到第五章进行详细的实现介绍，第六章将对性能测试结果进行评估以及对安全防护功能进行安全测试。

第三章　安全隔离执行环境

当前，国内外学者在软件隔离做了大量的研究[？],[36],[37],[38]，用来实现软件层完整性验证、入侵检测等[28],[39],[40],[41]。我们的基于同层隔离机制的地址空间隔离技术，可以为一些安全工具创建安全隔离的执行环境。在虚拟化层创建隔离执行环境，实现的方法有多种，根据国内外研究，主要有两种，嵌套虚拟化和微型Hypervisor，嵌套虚拟化的方法会导致大量的性能开销，微型Hypervisor在安全性和性能上有所缺陷，故需要安全隔离的环境，控制恶意Hypervisor随意访问该隔离的安全环境，阻止对安全隔离环境的破坏，能够以相对较低的性能开销运行。

采用同层隔离的思想创建安全隔离的地址空间。即创建与Hypervisor处于同一特权级别的地址空间，可以减小环境切换带来的性能开销。与微型Hypervisor比较，同层隔离地址空间可以防御针对Hypervisor的攻击，同时还能抵御来自恶意Hypervisor的攻击。两个地址空间的切换要求是安全的，安全切换门可以保证两个地址空间之间切换的安全性。为了保证隔离地址空间的安全性，需要对隔离地址空间进行安全保护。

综上所述，基本技术点包含同层隔离技术、安全切换门技术、对隔离地址空间的安全防护技术。

3.1　执行环境的创建

同层隔离技术的基本原理是创建的地址空间位于与Hypervisor相同特权级别，不依赖于更高特权级别，也不依赖于额外定制的硬件，能够提供安全隔离的执行环境，避免来自非可信Hypervisor的攻击。创建地址空间的实现方法是在虚拟化层创建另一套页表，该页表包含了原虚拟化层的地址空间，同时包含了新的地址空间，新的地址空间主要是用于保护虚拟机与宿主机交互的关键系统数据、用于虚拟机地址映射监控功能，保护这些进程运行时的代码段和数据段的安全性。

首先通过创建新的页表，64位系统上创建适合64位系统的页表，高虚拟地址空间表示内核空间，低地址空间表示用户空间。地址空间包含原地址空间的代码段和数据段。新的页表入口地址被保存在安全的地方。

3.2　安全切换门

安全切换门的目的是保证两个地址空间的切换是安全的，从原地址空间切换到新的隔离地址空间，或者从新隔离地址空间切换到原地址空间，该切换过程必须是安全的，才能避免给新隔离地址空间带来安全威胁。

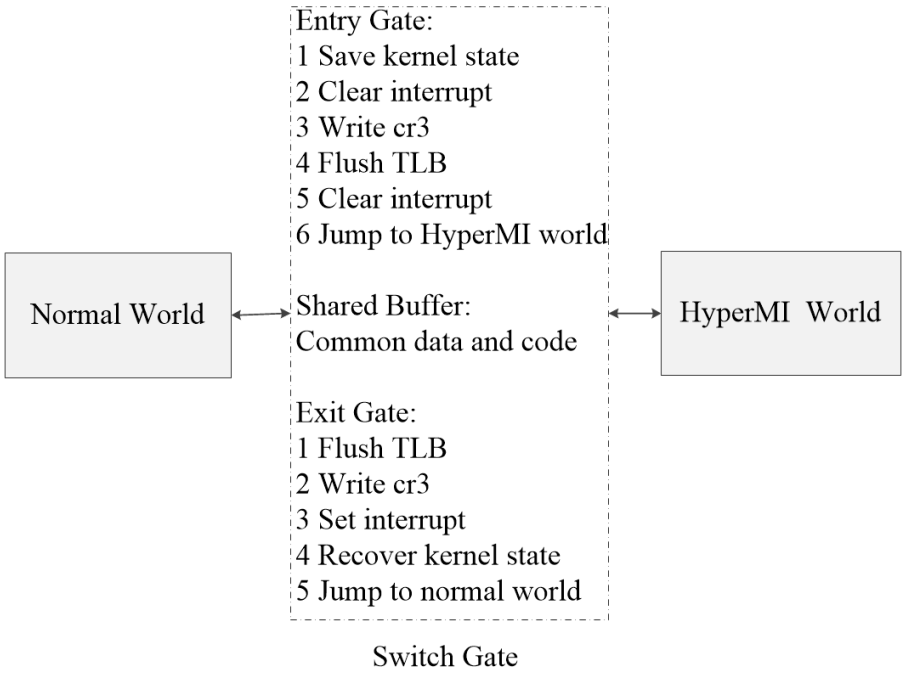

图 3.1　安全切换门

切换门包含进入门、退出门和共享缓冲区，进入门是用于从原地址空间切换到新隔离的地址空间，退出门是用于从新隔离地址空间退出到原地址空间。共享缓冲区包含公共的代码段和数据段，公共的代码段是指切换的代码段，公共数据段是指切换需要的地址空间入口地址。这些切换的地址空间入口地址需要被保护，以防攻击者获取这些信息切换的基本过程是：1）读取新页表的入口地址，2）跳转到新页表所在地址，3）切换到新的隔离地址空间完成，相关进程运行，4）读取原页表的入口地址，5）跳转到原地址空间，6）切换回原地址空间完成。

为了保证切换过程的安全性，在进入和退出过程中，使用原子操作技术保证切换的原子性，地址空间切换不可绕过性。添加原子操作后的进入过程如图3.1：1）保存内核信息状态（特权寄存器、中断状态等）到栈中，2）关中断，3）加载新页表的入口地址到CR3寄存器中，刷新TLB，4）再次关中断，5）跳转到新隔离地址空间中。反之，退出的操作相反。在该过程中，两次中断操作可以防止攻击者跳过第一次中断获取入口与地址再次跳到新隔离地址空间中。

3.3　执行环境的安全防护

在通过同层隔离技术创建隔离的地址空间，使用安全切换门进行地址空间切换，那么需要对地址空间的隔离性和安全性进行防护[42]，第三个技术点是对隔离地址空间进行安全防护。隔离地址主要使用新的页表创建新地址空间，特权寄存器存放页表入口地址，那么就需要对新页表、特权寄存器的访问进行安全防护。此外，通过DMA访问可以直接访问物理内存空间，并不经过页表，还需要对DMA访问进行控制。综上，

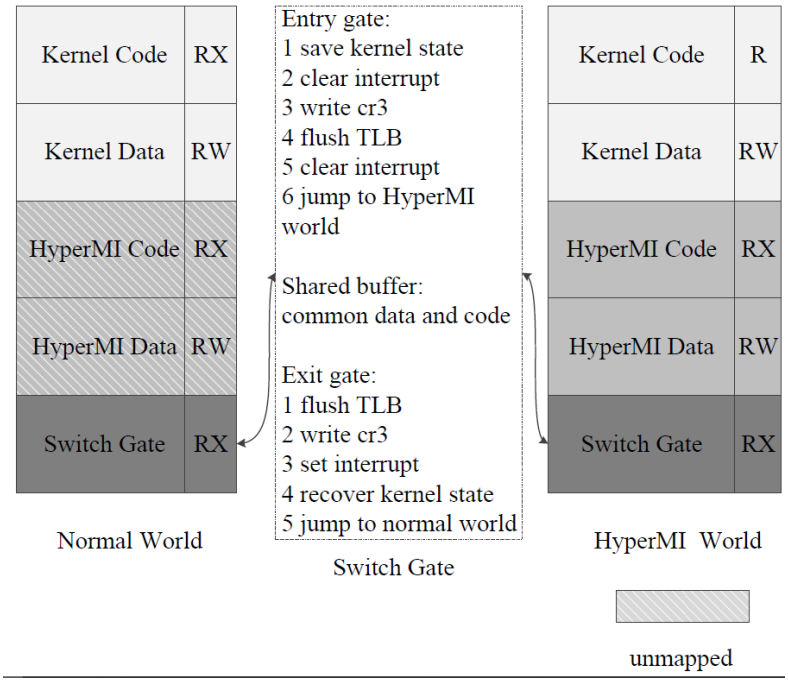

图 3.2　页表安全防护

防护措施主要是三点：对新页表的访问控制、特权寄存器访问控制、DMA访问控制。

3.3.1　新页表访问控制

虚拟化层Hypervisor在当前系统都拥有一定的权限，拥有对页表访问的权限，一旦Hypervisor被攻击，攻击者就可以通过破坏页表对新隔离地址空间进行破坏。可能的攻击如下：1）在切换的过程中，攻击者可以加载恶意的页表到CR3寄存器中，或者篡改原地址空间中的页表来映射新隔离地址空间中的地址（敏感数据的地址），最终恶意访问敏感数据。2）因新页表中包含原地址空间中的代码段和数据段，当新隔离地址空间在进行时，原地址空间中的代码依旧可以运行，一旦这些代码含有一定的漏洞，被攻击后，可能会破坏新地址空间中的一些进程。

针对如上攻击，防护措施如图3.2：1）针对第一种攻击，虚拟机安全套件系统的代码和数据段在原地址空间（页表）中并没有映射，为了保护新页表入口地址不被泄露，从原系统页表上移除了新页表的入口地址，剥夺了访问CR3寄存器的能力，这样就可以避免加载非法页表攻击和绕过安全隔离地址空间攻击。2）针对第二种攻击，在新页表中对原地址空间中的代码段进行访问控制，使得代码段不具备运行权限。对页表写权限进行控制，防止攻击者直接篡改页表内容更改代码段的访问权限，或者直接篡改页表映射，导致错误映射并泄露敏感信息。

3.3.2　特权寄存器访问控制

需要对一些特权寄存器进行访问控制，分别是CR0、CR3、CR4寄存器。通过剥夺这些特权寄存器的访问权限，让这些寄存器的访问在新的隔离地址空间中运行。

3.3.3 DMA访问控制

DMA的特点是直接访问物理内存，为了避免攻击者直接通过DMA方式访问敏感数据所在内存，采用IOMMU方式对映射的地址进行访问控制，对IOMMU中的页表删除映射到隔离空间的地址映射，同时在映射发生时验证映射地址的合法性。

3.4 小结

同层隔离技术是一种软件的方法，可以提供安全隔离可执行环境（TEE），保证系统运行的安全性，相比较其余的硬件方法和软件方法，其移植性相对较好，性能开销相对较低。对于云平台提供商，硬件需要定制，移植性相对较差，可广泛适用性相对较差；其余软件方法主要包含嵌套虚拟化方法，嵌套虚拟化通过创建比Hypervisor层更高层的软件层，但是频繁的特权级别切换会导致性能开销增大。综上，同层隔离技术移植性较好，适用于云平台提供商，性能开销相对较低。

第四章　Hypervisor关键数据监控

由于当前的底层的物理资源是被软件层的Hypervisor和各个客户虚拟机共享使用，各个虚拟机之间缺失资源的完全隔离性，同时虚拟机（VM）和宿主机（HOS）交互方式是通过在同一CPU上进行上下文切换，关键交互数据主要是VMCS结构体。所以存在这样的威胁，恶意的攻击者在攻破Hypervisor后，恶意访问上下文切换过程中使用的VMCS结构体中的数据信息，篡改数据，导致宿主机或者虚拟机的特权寄存器（CR0、CR3、CR4）等关键信息被篡改，导致系统被进一步攻击，系统数据被泄露等严重问题。为了阻止攻击，提出了虚拟机与宿主机交互监控技术，该技术主要包含两个技术点，隐藏VMCS结构体在隔离地址空间技术和剥夺宿主机的VMCS结构体访问功能技术。

4.1　上下文安全切换

宿主机和虚拟机交互的方式是在同一个虚拟CPU（vCPU）使用分时策略，当vCPU上运行虚拟机时，需要从宿主机状态切换到虚拟机状态，基本过程是：1）保存宿主机各信息到VMCS结构体中的宿主机部分，2）将VMCS结构体中虚拟机部分写到当前vCPU中的各个特权寄存器等，3）运行vCPU。当vCPU运行宿主机时，过程相反。那么，VMCS结构体十分重要，是两者交互的关键系统数据，必须要阻止攻击者恶意访问VMCS结构体。

为了达到这样的目的，避免攻击者在Hypervisor被攻击后恶意访问VMCS结构体，虚拟机安全套件系统将VMCS结构体的地址进行了隐藏，将其地址隐藏在新的安全隔离地址空间中，这样可以避免攻击者恶意访问VMCS结构体，从而避免攻击者进行进一步的攻击，阻止系统信息的泄露。那么，为了更充分地了解虚拟机安全监控，本文对一些背景知识进行介绍。

4.1.1　上下文切换

4.1.1.1　上下文

上下文是指程序（进程/中断）运行过程中使用的寄存器内容的最小集合[43]。这些寄存器代表着程序运行和处理器运行的状态信息，例如CR3寄存器指向页表的入口地址，页表是用于虚拟地址到物理地址的翻译。以下是X86平台上的寄存器分类。

1）通用寄存器

2）段相关寄存器组

3）标志寄存器

4）程序指针寄存器

5） GDT基地址

6） LDT段选择符

7） IDT基地址

8）控制寄存器组

9）浮点相关寄存器组

10）特殊用途的寄存器

一个程序的上下文可能是上面列出内容的一个子集，也可能是全部。在后面的部分，提到的上下文都是指上下文切换时必须更改的寄存器集合。

4.1.1.2 上下文切换原因

上下文切换是程序从一种行为切换到另一种行为，主要是将相关的寄存器信息进行保存，以备处理器正常运行。在系统中，我们讨论三种切换，用户态切换、进程切换、中断切换。

4.1.1.3 硬件虚拟化中的上下文切换

英特尔中该技术的基本思想如下：

引入两种操作模式，VMX操作模式。

根操作模式（ROOT模式）：虚拟机监控器运行的模式。

非根模式（NON-ROOT模式）：客户机运行模式。

这两种操作模式都有相应的特权级0-3特权级，即虚拟机监控器和客户机的0 3级。引入这两种操作模式的原因很简单，虚拟化是通过“陷入再模拟”的方式实现，IA32架构上有19条敏感指令不能被模拟，导致虚拟化漏洞的发生。解决办法是通过触发异常解决这些敏感指令，这种方法能改变指令的语义，导致与现有软件不兼容，无法在商业平台上使用。主要的解决方案是，重新定义非根模式下的敏感指令的执行，可以不再使用陷入模拟的方式执行；对于根模式下，不需要做任何更改。

虚拟机监控器和虚拟机采用分时的策略，在同一CPU上运行，就会出现两者进行交互切换过程。非根模式下敏感指令导致的陷入再模拟过程被称为VM EXIT，即虚拟机退出。从非根模式到根模式。相反的操作被称为VM Entry。

为了支持CPU虚拟化，完成处理上述切换过程，VMCS（虚拟机控制结构）被引入，它保存了虚拟CPU相关的状态，包括虚拟机监控器和虚拟机的特权寄存器值，指令指针地址等。VMCS主要是用于CPU使用，在CPU进行切换时，从根模式到非根模式或者从非根到根模式，会自动查询和更新VMCS。

Hypervisor是通过一些指令访问VMCS，VMREAD/VMWRITE读写VMCS；通过VMLAUCH/VMRESUME从根模式到非根模式。这些指令的执行被包含在一些函数中。

4.1.2 VMCS

VMCS域的6大组成部分。

1）客户机状态域：保存客户机在非根模式下的关键数据结构信息，在虚拟机退出/进入时，用于将这些信息保存/读取到该域。

2）宿主机状态域：保存Hypervisor的运行状态，用于在CPU在虚拟机退出/进入时，加载/读取该域，保证程序的正常运行。

3）虚拟机进入控制域：控制虚拟机的进入过程。

4）虚拟机执行控制域：控制处理器在非根模式下的执行流程。

5）虚拟机退出控制域：控制虚拟机退出的执行流程。

6）虚拟机退出信息域：标记虚拟机退出原因及其额外信息。

其中最重要的是客户机状态域和宿主机状态域。

客户机状态域信息：包括客户机寄存器状态内容(Guest Register State)和非寄存器状态内容。

宿主机状态域信息：存储Hypervisor的寄存器信息, 在发生VMExit事件时候恢复到相应寄存器。

控制寄存器：CR0, CR3, CR4,Esp,Eip。CS, SS, DS, ES, FS, GS和TR寄存器，段选择子，FS, GS, TR, GDTR, IDTR 信息, 基址。

一些MSR寄存器, IA32_SYSENTER_CS, IA32_SYSENTER_ESP,IA32_SYSENTER_EIP, IA32_PERF_GLOBAL_CTRL,IA32_PAT, IA32_EFER。

4.1.3 VMX操作模式

4.1.3.1 VMX操作执行流

当虚拟机监控器需要该功能时，可以通过虚拟化技术中的两条指令来打开/关闭该操作模式。这两条指令主要是VMXON/VMXOFF，分别表示打开/关闭VMX操作。VMX操作模式执行流如下：

1）虚拟机监控器执行VMXON指令进入VMX模式，CPU处于根操作模式。虚拟化操作被启动。

2）执行VMLAUNCH或者VMRESUME指令，前者是第一次加载虚拟机需要执行的，后者是第二次及以后需要执行的指令，及虚拟机开启和虚拟化重启动。这样会产生虚拟机进入，客户虚拟机运行，CPU处于非根模式，在该过程中将宿主机的信息写到VMCS，从VMCS中加载虚拟机的信息到各个寄存器，从而保证CPU的运行。

3）虚拟机退出发生条件：当客户虚拟机执行特权指令时，或者其执行了一些中断异常的事件，虚拟机退出被触发，从而CPU需要切换到虚拟机监控器，即从非根模式切换到根模式。在该过程中，CPU将客户虚拟机中寄存器信息存放到VMCS结构体中，随后再从VMCS中读取宿主机的各种信息将其写入到当前系统中运行的寄存器中。当

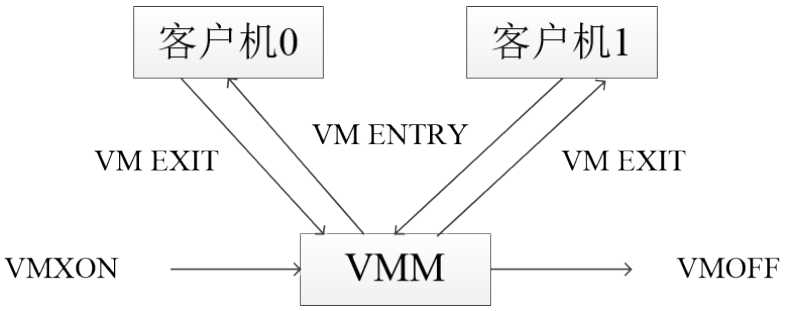

图 4.1　VMX操作模式

切换到根模式之后，需要处理一些退出之后的事件，包括中断异常的事件处理或者特权指令的执行等问题，根据VMCS中虚拟机退出原因的记录继而处理相应的事件。

4）当处理结束后，会重新执行VMRESUME指令重新回到非根模式，继续运行客户虚拟机。

5）如若决定停止虚拟机监控器的操作，执行VMOFF指令关闭VMM操作模式，结束操作，但一般并不关闭，默认开启操作。

4.1.3.2　虚拟机退出/进入

虚拟机监控器在机器加电后，系统引导，会进行类似操作系统相似的初始化过程，在所有操作准备好之后，会通过前述已经讲过的VMXON指令开启虚拟机监控器运行，从而使得CPU进入根模式，在创建虚拟机的时候，虚拟机监控器会通过VMRESUME或VMLAUNCH指令切换到非根模式运行虚拟机，虚拟机引起虚拟机退出后，CPU会切换到根模式运行宿主机。接下来，会详细介绍虚拟机进入和虚拟机退出。

1）虚拟机进入

虚拟机进入是CPU从根模式切换到非根模式的过程，也就是从虚拟机监控器运行到虚拟机运行状态的过程。这个操作是虚拟机监控器发起，在发起之前保存了VMCS的信息。

2) 虚拟机进入执行过程

- CPU使用VMLAUNCH/VMRESUME使得虚拟机进入。
- 执行基本系统检查确保虚拟机进入可以进行。
- 对VMCS中的宿主机状态域进行数据信息检查，确保下一次虚拟机退出时访问VMCS中该域信息是正确的。
- 检查VMCS中虚拟机状态域的数据正确性，通过虚拟机状态域信息装载CPU处理器。
- 装载MSR寄存器。
- 根据事件注入控制的配置，注入一个事件到虚拟机中。

- 如果前六步无法正确执行，那么虚拟机进入操作执行失败。否则的话，执行成功，切换到了非根模式，开始执行虚拟机指令。

3）虚拟机退出

虚拟机退出时从非根模式到根模式，从虚拟机到虚拟机监控器的操作。虚拟机退出是在虚拟机中引发的，在非根模式下指令敏感指令、中断、异常等都会引发虚拟机退出。虚拟机退出事件处理是由虚拟机监控器进行。具体退出流程如下：

- CPU将虚拟机退出原因信息记录到VMCS相应的信息域中，虚拟机进入的中断信息段的第31位被清零。
- CPU状态信息被保存到VMCS虚拟机状态域中。包含MSR的设置。
- 根据VMCS中宿主机状态域和虚拟机退出控制域中的设置，将宿主机的状态信息加载到CPU相应的寄存器中。加载MSR。
- CPU从非根模式切换到根模式。从RIP中指定的指令开始执行入口函数。
- 处理虚拟机退出事件函数。
- 退出处理结束之后，通过VMLAUNCH/VMRESUME指令发起虚拟机进入重新运行客户机。
- 虚拟机退出和虚拟机进入操作循环执行。

4.1.4 攻击威胁

虚拟机监控器与虚拟机的交互通道只有一种方式，即虚拟机进入和虚拟机退出。在该过程中，如上所示，所有的操作都会写入VMCS结构体。那么直接攻击VMCS，就可能对整个系统的运行产生很大的攻击，其目的是访问VMCS，更改VMCS虚拟机客户信息域和宿主机信息域。一般有如下攻击威胁。

1）获取VMCS地址，更改虚拟机信息域中的CR3 CR0 CR4等寄存器。

2）加载恶意的页表、关闭DEP机制、关闭SMEP机制。

3）更改宿主机信息域中的RIP CR0 CR3 CR4等信息。

4）加载恶意RIP地址，导致控制流劫持攻击；关闭DEP机制，加载恶意页表、关闭SMEP机制。为进一步的攻击做准备。

5）直接泄露VMCS结构体中的信息。

4.1.5 防御方案

4.1.5.1 VMCS隐藏

因为攻击者的目的是访问VMCS，通过将VMCS结构体隐藏在HW中，可以阻止攻击者直接访问VMCS结构体。VMCS结构体主要有软件和硬件两类。软件的主要是针对开发人员方便使用的接口，硬件上的VMCS在软件层一般是无法访问的。因为将VMCS存放在HW中，攻击者通过非可信的虚拟机监控器层在软件接口上是无法访问到VMCS。

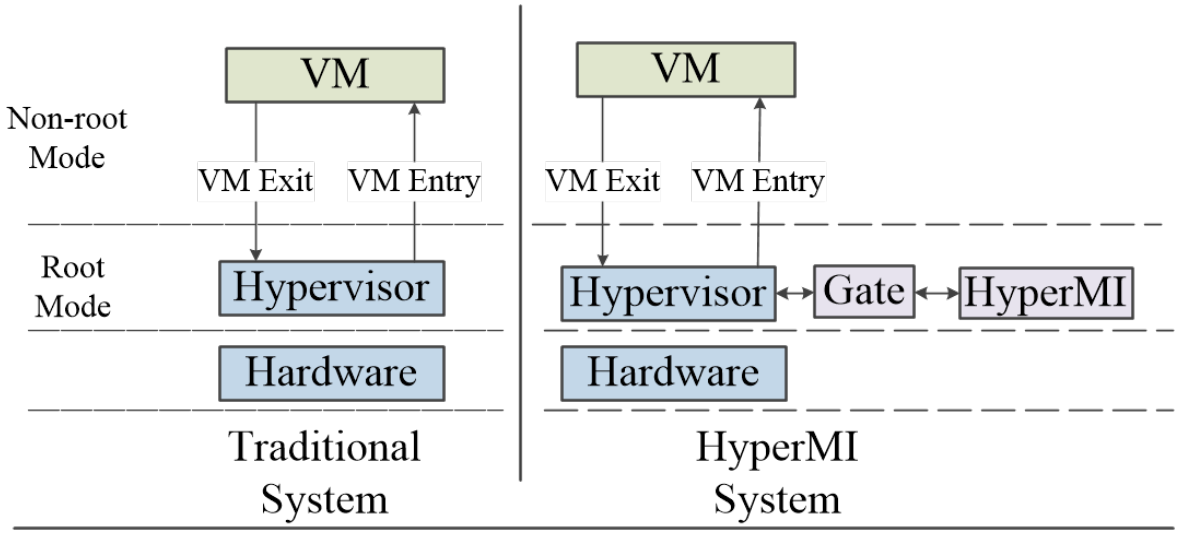

图 4.2　虚拟性与宿主机交互监控过程示意图

4.1.5.2　VMCS访问操作管理

因VMCS被隐藏在HW区域中，那么Hypervisor本身无法在自己的地址空间中访问到VMCS，从而造成程序无法运行，系统终止的问题。为了使得系统能够正常地运行，HyperMI通过将VMCS访问的相关函数挂钩到安全执行环境中运行，这样就可以保证系统的正常运行。接下来，需要将访问VMCS地址的函数找到，并统计。通过对内核代码的阅读，总结出如下访问VMCS结构体的函数，主要是创建、访问、销毁VMCS函数，主要在mmu_audit.c文件中。

访问VMCS结构体的控制流过程发生了变化，详细说明见图4.2。

4.1.5.3　挂钩方式

原函数A，新函数B。将A挂钩到B，基本的算法操作如下。

1) 使得函数A地址代码段所在物理页可写。

2) 确保函数A 的汇编代码长度大于16字节，否则在该函数的起始处添加NOP指令，共16条NOP指令。

3) 拷贝函数A 的前16字节到另一数据空间C。

4) 预分配地址空间D共16字节。

5) 在D中写入两条hook指令，mov b,eax ; jmp eax. 剩余空间使用NOP进行填充。其含义是将函数B的入口地址b赋值给寄存器EAX，随后跳转到EAX寄存器中地址值。

6) 在函数A入口地址处写入D的数据。

7) 在安全执行环境中执行函数B需要的操作，能够确保函数A的功能执行，并且保证VMCS的地址不被泄露。

8) 函数B被执行完成后，必须跳转回到函数A。由于X86体系架构中汇编指令，call和return的关系，根据编译原理知识。当函数B被执行完成后，控制流会执行到调用

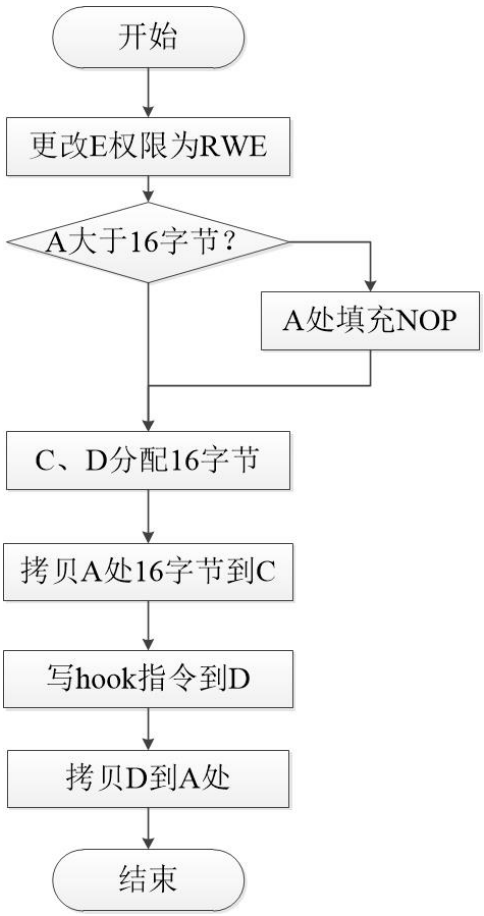

图 4.3 HOOK算法执行流程图

函数A的指令处，因RIP并没有发生变化。整个过程并不会对控制流产生影响，能够保障程序的正常运行。

9) 挂钩函数结束后，将C拷贝到A的起始16字节处。

10) 函数A所在代码段不可写。

假设如下术语，函数A代码段: E；预分配数据缓冲空间D 16字节；预分配数据缓冲空间C 16字节；挂钩到的函数：B。整个程序流程图如图4.3。

通过对这些函数的挂钩操作，使得当这些函数在NW中执行的时候，作为环境切换的条件，跳转到安全执行环境HW中执行。当这些函数执行完后，会从HW切换回到NW中，主要的切换方式是通过切换门进行，其伪代码参见第三章。通过将NW的页表入口地址写到CR3寄存器中，完成切换。

4.2 退出重定向

上一个技术点介绍了隐藏VMCS结构体的地址，避免攻击者恶意访问VMCS结构体，但是正常程序的运行也需要访问VMCS结构体，例如，VMCS创建，VMCS访问，VMCS销毁，VMCS加载。为了使得系统能够正常运行，虚拟机安全套件系统

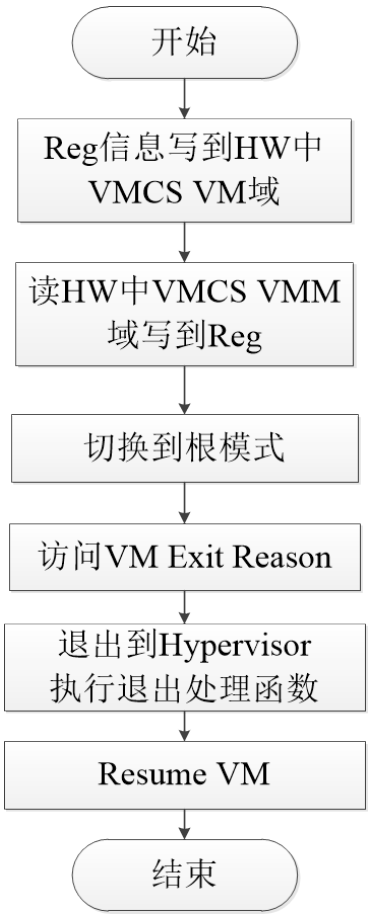

图 4.4　虚拟机退出重定向流程图

将VMCS结构体访问功能剥夺了，并将所有的VMCS访问功能在新的安全隔离地址空间中运行，实现技术上主要使用hook挂钩技术。

将有关VMCS访问的函数在隔离地址空间中执行，或者对某些访问函数（不需要VMCS结构体地址的函数）只返回信号信息，并不真实返回VMCS的地址。防止VMCS被恶意访问和篡改。

虚拟机退出和进入的操作在4.1节中已经介绍，其基本流程也被详细介绍，在虚拟机退出的过程中，会存在访问VMCS结构体获取退出原因的操作，该操作是在根模式下进行运行。

4.2.1　虚拟机退出

虚拟机退出流程：

- 将寄存器信息写入到VMCS中虚拟机信息域。
- 将VMCS中虚拟机监控器信息域中信息写到寄存器中。
- CPU从非根模式切换到根模式。
- 访问VMCS中虚拟机退出原因，执行处理函数。
- 重新启动虚拟机。

根据虚拟机退出流程进行分析，不同的虚拟机退出原因会对应不同的虚拟机退出事件处理。一般包含EPT访问地址缺页处理、IO访问处理等。基本的虚拟机退出重定向处理如上。由于虚拟机退出事件处理代码和函数相对庞大，考虑实际的编程问题，不可能将所有的事件处理函数都通过挂钩的方式放在HW中，这样一来会导致HW中大量的代码基以及增大程序量。

4.2.2 退出重定向

当切换到根模式的时候，会访问VMCS获取虚拟机退出原因，随后进行退出函数处理操作，在这个过程中会访问VMCS，该操作先是软件操作直接访问VMCS，随后CPU会执行硬件操作。由于VMCS被隐藏在HW中，根模式下运行的是Hypervisor，Hypervisor会因访问不到VMCS从而造成系统中止或者崩溃。为了避免这一状况的发生，于是HyperMI设计了虚拟机退出重定向模块，主要是HyperMI处理虚拟机退出，处理结束后，将虚拟机退出处理操作重新定向给Hypervisor进行处理。也就是说，虚拟机退出的整个过程会被划分为两个步骤，第一个步骤是HyperMI处理虚拟机退出过程，直至切换到根模式，未处理虚拟机退出事件；第二个步骤是将退出处理事件重定向到Hypervisor，切换到Hypervisor来处理。可参见流程图4.4。

不同的虚拟机退出原因会对应不同的虚拟机退出事件处理。一般包含EPT访问地址缺页处理、IO访问处理等。基本的虚拟机退出重定向处理如上。由于虚拟机退出事件处理代码和函数相对庞大，考虑实际的编程问题，不可能将所有的事件处理函数都通过挂钩的方式放在HW中，这样一来会导致HW中大量的代码基以及增大程序量。通过详细的分析和比较，将退出事件处理函数重定向到Hypervisor，在获取退出原因后，Hypervisor得到退出处理事件入口代码时，执行虚拟机退出处理事件。那么，这样就不需要挂钩所有的退出处理事件，减小HW的代码量，减小被恶意攻击的危险，减少程序编写的复杂性。

4.3 小结

虚拟机与宿主机交互关键数据监控技术包含两个技术点，隐藏上下文切换中VMCS结构体地址技术、剥夺宿主机访问VMCS结构体功能技术以及虚拟机退出事件处理重定向技术。

其独到之处在于虚拟机与宿主机交互的唯一关键数据和方式被严格监控，同时为了保障系统的正常运行，还剥夺了宿主机对VMCS结构体的访问功能，整个交互过程被监控，所以攻击者无法通过非可信Hypervisor恶意篡改虚拟机的运行时状态信息，同时可以保护EPTP（VMCS结构体中的一个关键数据）的安全访问。该技术可以保证虚拟机的运行时关键系统数据不被篡改，CR3、CR4、CR0等，可以保证DEP、SEMP机制的正常运行，阻止系统被进一步攻击。

这个虚拟机退出过程被分为两个步骤，HyperMI处理虚拟机退出访问VMCS过程，Hypervisor处理虚拟机退出事件。在HW中访问VMCS结构体完成虚拟机退出过程，CPU从非根模式切换到根模式，获取了虚拟机退出原因和事件处理入口地址后，虚拟机退出事件的处理不归HyperMI处理，而是归Hypervisor进行处理。主要考虑到的原因如下：1）此时不再需要访问VMCS；2）若通过挂钩大量的退出事件处理函数到HW中，造成HW中代码量过大，威胁性增加，编程代码量大。通过虚拟机退出重定向成功地解决了该问题，使得整体的虚拟机与宿主机交互关键数据监控模块被完成。

第五章　虚拟机内存高强度隔离

因为虚拟机监控器和虚拟机共享底层的物理资源，同时各个虚拟机之间没有完全的隔离性[44]，尤其是内存这样的硬件，所以存在这样的威胁，恶意的虚拟机或者恶意的Hypervisor还可以对受害虚拟机的物理资源随意访问，实现跨域攻击，导致信息泄露等严重问题[45]，在这些攻击过程中可能使用针对内存映射的多重映射攻击和双映射攻击。一些学者采用隔离商业Hypervisor产品的方法[46]，本文通过使用动态标记与跟踪技术，使得所有虚拟机之间的内存实现高强度隔离，如图5.1。

5.1　地址映射监控

在支持硬件虚拟化的平台上，虚拟机的地址映射需要两套页表，一套虚拟机自身的页表，另一套是归Hypervisor管理的EPT（扩展页表）。虚拟机上的虚拟地址翻译过程是：通过自身页表将GVA（客户机虚拟地址）映射到GPA（客户机物理地址），通过EPT页表将GPA映射到HPA（宿主机物理地址）。

首先，本文详细介绍一下客户机物理地址GPA和扩展页表EPT。充分了解虚拟机地址映射的整个过程。

客户机物理地址空间GPA 对一个操作系统来说，内存是物理地址从0开始的连续的地址空间。在虚拟化环境下，真正拥有物理内存的是虚拟机监控器Hypervisor。物理内存只有一份，Hypervisor需要在宿主机上为每个客户机操作系统模拟出可以当作物理内存一样使用的虚拟内存。Hypervisor模拟了一层新的地址空间：客户机物理地址空间。

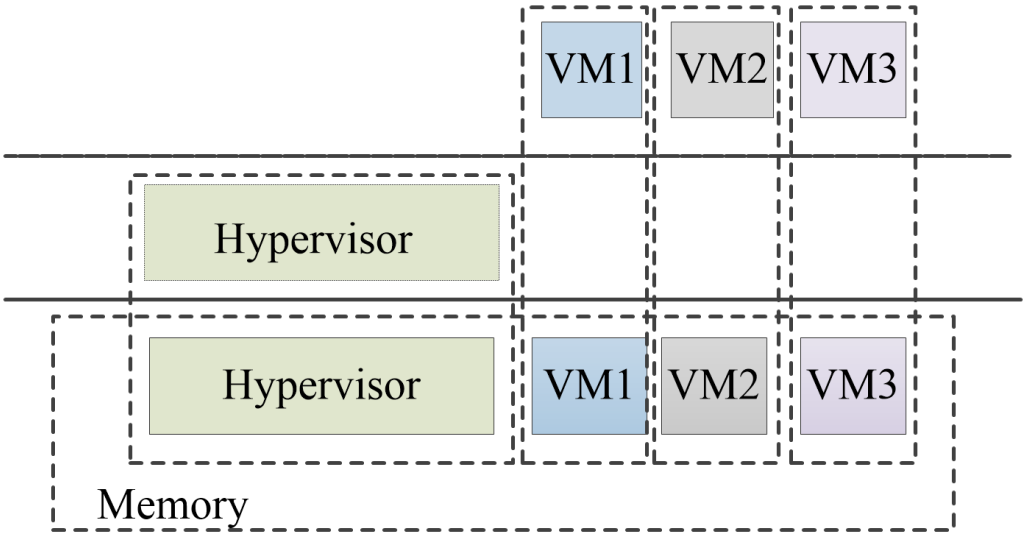

图 5.1　内存高强度隔离效果图

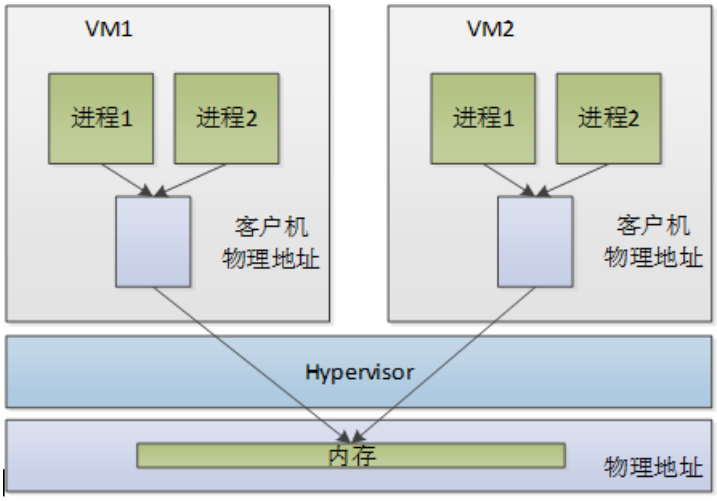

图 5.2　客户机虚拟地址翻译过程

```
int kvm_mmu_create(struct kvm_vcpu *vcpu)
{
        ASSERT(vcpu);

        vcpu->arch.walk_mmu = &vcpu->arch.mmu;
        vcpu->arch.mmu.root_hpa = INVALID_PAGE;
        vcpu->arch.mmu.translate_gpa = translate_gpa;
        vcpu->arch.nested_mmu.translate_gpa = translate_nested_gpa;

        return alloc_mmu_pages(vcpu);
}
```

图 5.3　EPT创建函数说明

该机制对客户机是透明的，GPA不是宿主机最终的物理地址空间，客户机虚拟地址翻译过程如图5.2。

扩展页表EPT 每一台虚拟机都有对应各自的EPT,用于将GPA转换为HPA，详细的映射过程如下。

由于引入了客户机物理地址空间，内存地址转换过程变为：从客户机虚拟地址GVA转换到客户机物理地址GPA；再从客户机物理地址GPA转换到宿主机物理地址HPA。其中，GVA到GPA的转换由客户机操作系统决定，客户机操作系统通过VMCS中Guest状态域CR3寄存器指向的页表来指定；GPA到HPA的转换是由Hypervisor决定，Hypervisor在将物理内存分配给客户机时确定GPA到HPA的转换，这个映射关系往往由Hypervisor中的内部数据结构记录，Hypervisor为每个虚拟机动态地维护了一张客户机物理地址与宿主机物理地址映射表，基于硬件创建的表叫做扩展页表（EPT），如下两张图5.3和5.4分别是EPT创建代码和原理图。

通过对上述两种关键数据结构，本文详细虚拟机地址映射过程基本流程。可以发现数据结构EPT在地址映射过程中的重要性，因EPT是由Hypervisor管理的，我们

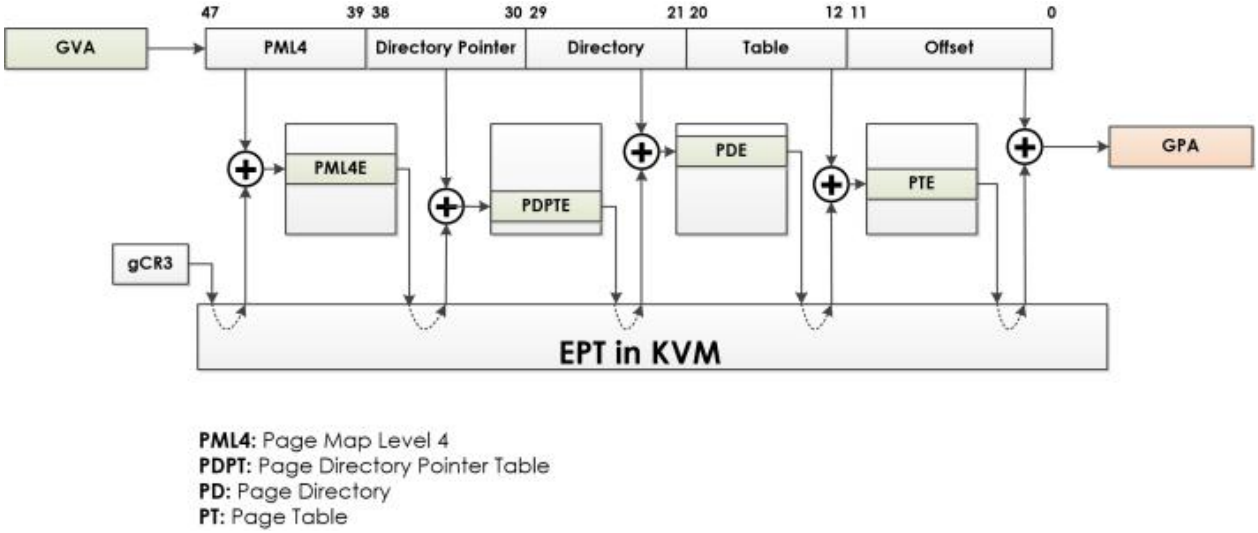

图 5.4 EPT翻译原理图

在本文中假设Hypervisor是不可信的，那么EPT的保密性和完整性需要进行考虑。攻击者可以实施如下的攻击：1）恶意虚拟机VM2得到虚拟机VM1的EPT地址，加载其地址就可以访问到VM1内存上的各种信息，从而成功地实现跨域攻击。2）直接访问VM1的EPT并且篡改其上的地址映射，从而实现内存越界访问攻击，例如：多映射攻击和重映射攻击等。这两种攻击在第二章威胁模型一节中被详细介绍，这些攻击会造成内存数据泄露。对这些攻击进行总结，我们发现攻击者的目标是通过访问EPT实施攻击继而访问内存上的敏感信息，根据这种攻击方式，只要控制Hypervisor对EPT的访问就可以避免这些攻击，于是制定了对应的防御策略。

本文采用了隐藏EPT在HW技术和EPT操作监控技术。应对如上的攻击，通过将EPT进行隐藏，避免来自非可信Hypervisor的访问操作（读写操作），进而能避免进一步访问内存信息的攻击。

为了确保VM加载自身的EPT，本文提出了EPT隔离技术，通过将VM与EPT进行了绑定，使得每台VM只能访问自身对应的EPT。通过创建的VM Mark表(表5.1)实现两者的绑定，表的内容如下。该表主要包含两个关键数据结构，VMID和EPT_Address，这两个关键的数据结构是随着虚拟机的创建而创建，随着虚拟机的销毁而销毁。VMID记录虚拟机的ID号，EPT_Address记录EPT的地址信息，这样就可以成功阻止第一种攻击。

表 5.1 VM Mark Table.

标签	VMID	EPTID	EPT_Address
描述	VM标识	EPT标识	EPT入口地址

当EPT地址被隐藏后，Hypervisor中原有的EPT操作函数（EPT创建、加载、遍历、

销毁）都会访问EPT的地址，由于这些地址Hypervisor根本访问不到，就会引发系统运行中止或者系统崩溃，为了避免这一现象的发生，本文设计了EPT操作监控技术。该技术的主要关键点在于将EPT访问的这些函数全部通过挂钩的方式[47]放在HW中执行。一旦这些函数执行，作为触发条件，控制流会跳转到HW中去执行。当这些函数执行结束后，控制流会跳转回到NW中，RIP中的值并不改变。

虚拟机地址映射监控，需要监控关键数据结构EPTP。将其地址进行隐藏，同时将访问EPTP数据相关的函数放在隔离地址空间中执行，或者返回信号信息，这些主要是通过挂钩的方式实现，并不直接返回EPTP的地址，目的是防止攻击者直接恶意访问EPTP，从而篡改EPTP加载恶意的EPT（扩展页表）。为了阻止第二种攻击，本文采用内存动态标记与跟踪技术实现阻止内存越界访问攻击。

5.2 内存动态标记与跟踪

内存动态标记与跟踪技术，主要包含两种技术，普通情况下的内存标记与跟踪技术和内存复用情况下的内存动态标记跟踪技术，这两种技术都是在安全隔离的地址空间中运行。

内存标记技术主要是通过对真实的物理内存，即虚拟机或者宿主机使用的物理内存进行标记，标记每一个物理内存页的使用属主。当内存被系统分配使用的时候，能够对内存的使用进行分辨，对当前正在被使用中的内存，不再给宿主机或者其余是虚拟机进行分配，可以防止内存双映射攻击；对于未被使用内存直接进行分配。对于即将被释放的内存，先清空其物理页上的信息，随后进行释放，目的是防止其余恶意虚拟机或者被攻陷的宿主机重新映射到物理页上进行内存多重映射攻击。

5.2.1 内存分配与跟踪

内存需要在被分配的时候被打上标签，并在使用的过程中，实时地被跟踪其使用情况，该过程可以阻止双映射攻击，内存双映射攻击（remap）的基本执行过程在威胁模型中已被详细介绍。假设两台虚拟机VM1和VM2，VM1作为远程攻击者使用的虚拟机，VM2作为受害者虚拟机。VM1使用虚拟地址C1、C2，对应的物理地址是D1、D2，VM2使用虚拟地址C3、C4，对应的物理地址是D3、D4。攻击者去修改C1对应的物理地址，通过更改C1对应VM1的EPT页表的最后一级，使得最后一级指向物理页D3，那么最终会造成虚拟地址C1对应的物理内存页是D3，而不是D1。这样可以访问D3上的数据，从而实现了用户数据泄露攻击。通过设置Page Mark表(表5.3)，攻击者在更改EPT时，由于EPT被保存在HW中，EPTP被保存在HW中，在访问EPT的时候会出现阻止；由于VM1和VM2可以访问的物理页并不相同，若攻击者更改EPT成功后，VM1去访问物理地址D3，由于D3并不存在内存中，此时会引发EPT更新，那么在更新的过程中会访问Page Mark表来验证D3的属主，由于属主是VM2，此时拦截到运行并

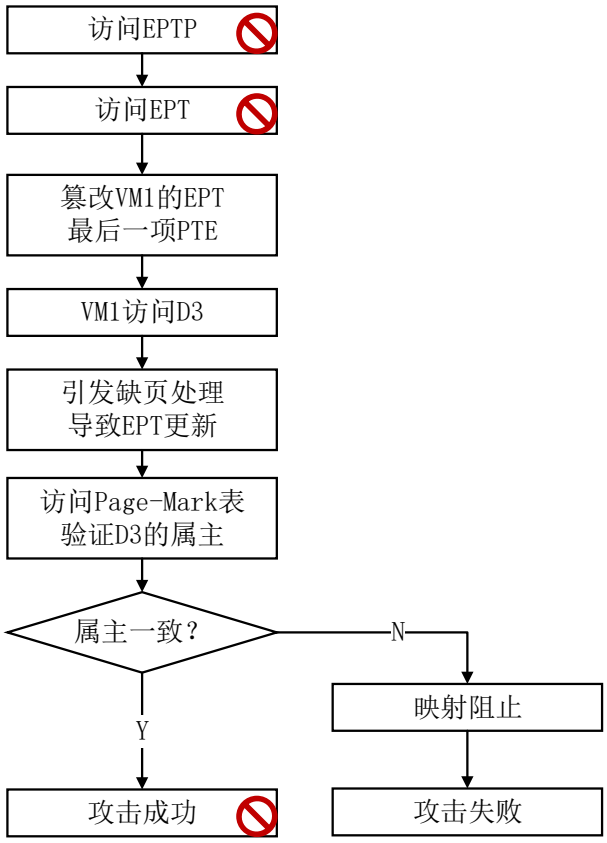

图 5.5 内存双映射攻击阻止说明图

访问的虚拟机是VM1，那么这种映射并加载物理页到内存中的进程会被成功地阻止，图5.5描述了攻击路径，其中这些攻击路径会被多次阻止。

5.2.2 内存安全释放

页释放的一般流程是判断当前在active列表中的页是否可以被释放，如果可以则放到inactive列表中，依次释放，否则，将这些不同情况的页进行不同的处理。shrink_page_list是页面回收中最重要的函数之一，图5.6说明了处理流程，具体的处理流程如下：

1、如果页面被锁住了，将页面保留在inactive list中，再次扫描试图回收这些页。

2、如果在回写控制结构体标记了不允许进行unmap操作，把仍有映射的页面保留在inactive list中。

3、如果页面正在回写中，对于同步操作，等待回写完成。对于异步操作，继续留在inactive list中，等待再次扫描再回收释放。

4、如果页面被再次访问，则有机会回到active list链表中。必须满足一下条件

a、检查page_referenced确认页面被访问；

b、检查order，如果order小于3，系统趋向回收大页面；对于较小页面，趋向保留

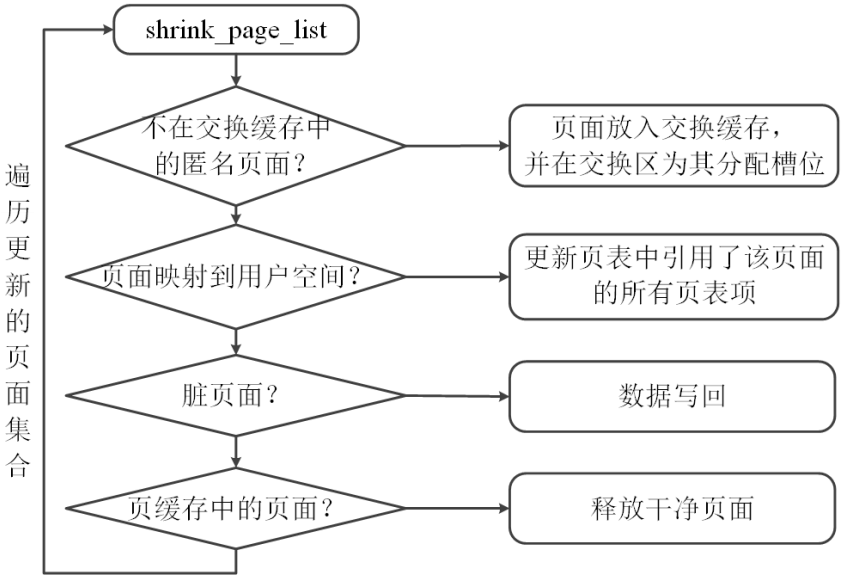

图 5.6　页面回收流程图

在active list中；

c、检查page_mapping_inuse。

5、如果是匿名页面，并且不在swap缓冲区中，把页放到swap的缓冲区中。

6、如果页被重新映射了，调用unmap函数释放页面。

7、如果页面是脏页，调用pageout将页内容写回。

8、如果页面和缓存相关联，调用try_to_release_page函数将缓存释放掉。

9、最后调用__remove_mapping函数把页面回收归还伙伴系统。

如上是页面回收的基本算法流程，为了使得内存页面安全释放，需要在页面最后释放的时候，删除该页面的内容并设置为零，删除该页面对应的PageFlag结构体，再彻底释放该页面到内存池，算法流程如表5.2。当页面上的内容被清除后，攻击者在物理页A（物理地址为B）被释放后，更改EPT页表映射，将虚拟地址C对应的GPA指向物理地址B，攻击详细过程参见威胁模型一节中重映射攻击。因页面回收算法设计，当物理页B被第一次释放后，其上的内容为零，此时C再次指向B时，访问C则得到的数据全部为零，这样就成功组织了攻击。

5.2.3　共享页接口设定

内存动态标记跟踪技术是针对KSM机制和Balloon机制对内存标记跟踪技术进行了扩展，KSM机制和Balloon机制可以使得虚拟机所拥有的内存随时发生变化，也就是在系统运行的过程中，内存会被分配、释放、重新分配，在内存释放之前还可能会被重

表 5.2 内存页面安全释放.

内存页面安全释放
1、在页面回收函数Shrink_page_list（）操作中判断所有的页面情况
2、回收函数中插入页面内容清空函数 clean_content_page(struct page* page)
3、在LKM模块代码中对clean_content_page进行挂钩并跳转到new_clean_content_page函数
4、对page上的内容进行清空，访问其地址并置值为零
5、释放该page对应的PageFlag结构体

表 5.3 Page Mark Table.

标签	OwnerID	Used
描述	属主标识	未使用或者使用

新分配给其他虚拟机，所以可能会造成内存标记的属主发生变化，与内存标记与跟踪功能中内存属主唯一性相互矛盾，导致内存标记跟踪功能失败。为了防止这种情况发生，通过对KSM机制和Balloon机制中内存易主进行监控，动态地变化内存的属主标识，保证内存动态标记跟踪功能的正常运行。

因为虚拟机监控器和虚拟机共享底层的物理资源，同时各个虚拟机之间没有完全的隔离性，所以存在这样的威胁，恶意的虚拟机或者恶意的Hypervisor还可以对受害虚拟机的物理资源随意访问，实现跨域攻击，导致信息泄露等严重问题。系统中存在KSM机制和Balloon机制，KSM机制会导致多个VM使用同一份物理内存页，这样就和物理页属主唯一性冲突了；Balloon机制中，经常会有物理内存页属主发生变化的情况，那么应该随属主变化而更改物理页的标签信息，否则会造成系统的崩溃。

对虚拟机的保护，主要体现在对虚拟机的物理资源隔离，主要方法是内存动态标记和跟踪，各个虚拟机以及Hypervisor只能对自己的资源进行访问，可阻止跨域攻击和信息泄露，阻止针对内存的重、多映射攻击。同时因系统可能会开启共享内存页功能，故为了兼容该功能，添加内存页共享页接口设置模块。

首先，针对KSM机制可能导致原代码不兼容问题。添加共享接口处理功能来解决，在gatefunction.h中ksm_page_modify()函数对物理内存页进行标记，标记shared属性，针对内核挂钩的函数是ksm_do_scan()。详细的标记方法是，根据KSM机制的请求，随后切换到安全执行空间中对页进行合并，包含两棵树，稳定树和非稳定树，首先查找稳定树，如若存在则合并，否则在非稳定树中进行查询，存在则合并，放到稳定树中，非稳定树销毁，不存在则加入到非稳定树中。

整个的合并操作在安全隔离的地址空间进行，shared标识标记，随后将合并后的结果返回到原系统中，切换回到原系统。针对KSM机制的物理页验证方法如下：当虚拟机地址映射的时候，关键函数是tdp_page_fault()，当映射物理页时，对于shared标记的物理页忽略属主检查，非shared标记的物理页检查属主情况，不一致则报警恶意访问。

其次，针对Balloon机制可能导致属主经常发生变化，需要及时更改物理页的属主信息，在tdp_page_fault()函数中对物理页进行属主标记，当balloon机制发生作用时，hook该函数，更改物理页的属主信息。

5.3 小结

虚拟机地址映射监控技术主要包含两个技术点，虚拟机地址映射监控和内存动态标记与跟踪技术。首先在隔离地址空间中监控虚拟机地址映射，在系统运行时，虚拟机的数据存放在内存中，所以内存是重要硬件之一，虚拟机的内存映射包含两类页表，一个是自身的页表，另一个是由宿主机管理的EPT（扩展页表）。那么为了统一地监控地址映射，虚拟机安全套件剥夺了宿主机管理所有EPT的功能，能够监控内存的分配。其次，在监控内存分配时，对内存进行属主标记并进行跟踪。这两个技术点监控了内存映射时的关键点，卡住了内存映射关键过程，同时考虑内存动态变化属主的问题，针对KSM和Balloon机制设置了新的Page Mark表，适应内存复用机制，对内存运行时过程进行了全面的考虑。

第六章　评估

6.1　性能评估

6.1.1　性能需求

由于系统的性能开销对云平台提供商很重要，系统的时间特性对于用户使用体验来讲，特别重要，本文在性能整体损耗和时间特性上对整个系统进行了性能分析，确保性能分析上能达到一定的要求。本文对系统在时间损耗相对较大的功能进行了测试，确保时间对于用户是可接受的。

A. 性能损耗 针对地址空间隔离、虚拟机映射监控和虚拟机上下文切换的性能损耗都要小于10%。

B. 时间特性 系统包含三大模块，安全执行环境的创建，虚拟机与Hypervisor交互关键数据监控，以及虚拟机内存隔离。安全执行环境的创建理论上并不需要大量的操作，也不会带来大量的性能开销，但是环境切换的过程会因为挂钩函数数量的多少，挂钩函数的使用频率受到影响。虚拟机与Hypervisor交互关键数据的监控模块中由于VMCS和EPT是被存储在HW中，访问这两个关键数据结构体的操作越频繁，那么会导致过多的环境切换，从而带来一定性能开销。同时访问VMCS和EPT的操作主要是虚拟机进入和退出，虚拟机内存缺页访问操作。虚拟机内存隔离功能模块会在物理页被分配时访问Page Mark 表和VM Mark表，过多的内存分配会导致过度地访问这两个表，从而带来一定的性能开销。根据上述分析，对环境切换、虚拟机与Hypervisor的安全切换、内存分配三个角度进行性能分析。由于这三种功能无法被直接测试，本文通过测试虚拟机启动来测试HyperMI在这三个功能上引入的性能开销。

- 环境切换对于隔离的新地址空间和原先的Hypervisor之间切换的时间要相对较小。
- 虚拟机与Hypervisor的安全切换 虚拟机在处理敏感指令的时候，会退出到Hypervisor中处理，从非root模式到root模式的过程被称为虚拟机退出，它的反过程被称为虚拟机进入，需要确保切换的过程时间开销相对较小。
- 虚拟机启动 需要保证虚拟机启动时间影响较小，保证用户可接受，并不影响用户的使用体验。

6.1.2　测试环境与配置

6.1.2.1　基准工具测试

测试环境：2台ubuntu14.04 LTS虚拟机， Linux-4.4.0，1G内存，磁盘大小20G。宿

主机 Linux4.4.1系统，32G内存，硬盘大小256G。
测试工具：SPEC CPU2006微观测试集和Bonnie++。
测试步骤：
1）使用测试工具测50次，将结果进行平均化。
2）编译运行测试代码，详细如下。
cd /root/cpu2006/
./install.sh
echo ”starting SPECCPU2006 at (*date*)”
source./*shrc*
bin/*runspec* − −*action* = *validate* − *oall* − *r*4 − *c Example* − *linux*64 − *amd*64 − *gcc*43.*cfgall*
echo”*SPECCPU*2006*endsat*(date)”

本次示例中runspec脚本用到的参数中，–action=validate表示执行validate这个测试行为（包括编译、执行、结果检查、生成报告等步骤），-o all表示输出测试报告的文件格式为尽可能多的格式（包括html、pdf、text、csv、raw等），-r 4（等价于–rate –copies 4）表示本次将会使用4个并发进程执行rate类型的测试（这样可以最大限度地消耗分配的4个CPU线程资源），–config xx.cfg表示使用xx.cfg配置文件来运行本次测试，最后的all表示执行整型（int）和浮点型（fp）两种测试类型。runspec的参数比较多也比较复杂，可以参考其官方网站的文档了解各个参数的细节。

3）在result目录下将HTML格式的CINT2006.001.ref.html（对整型的测试报告）和CFP2006.001.ref.html（对浮点型的测试报告）两个文件进行处理。获取最后的性能测试结果。

6.1.2.2 虚拟机启动关闭时间测试

测试一台虚拟机的启动时间和关闭时间，测试主要分为两部分，1）在没有加载HyperMI的KVM上进行测试。2）在加载了HyperMI的KVM上进行测试。

测试环境：1台ubuntu14.04 LTS虚拟机， Linux-4.4.0，1G内存，磁盘大小20G。宿主机 Linux4.4.1系统，32G内存，硬盘大小256G。

测试工具：无。

测试步骤：在没加载HyperMI的KVM上开启虚拟机，记录开启时间，随后1min后关闭虚拟机，记录关闭时间。在加载HyperMI的KVM上开启虚拟机，记录开启时间，随后1min后关闭虚拟机，记录关闭时间。

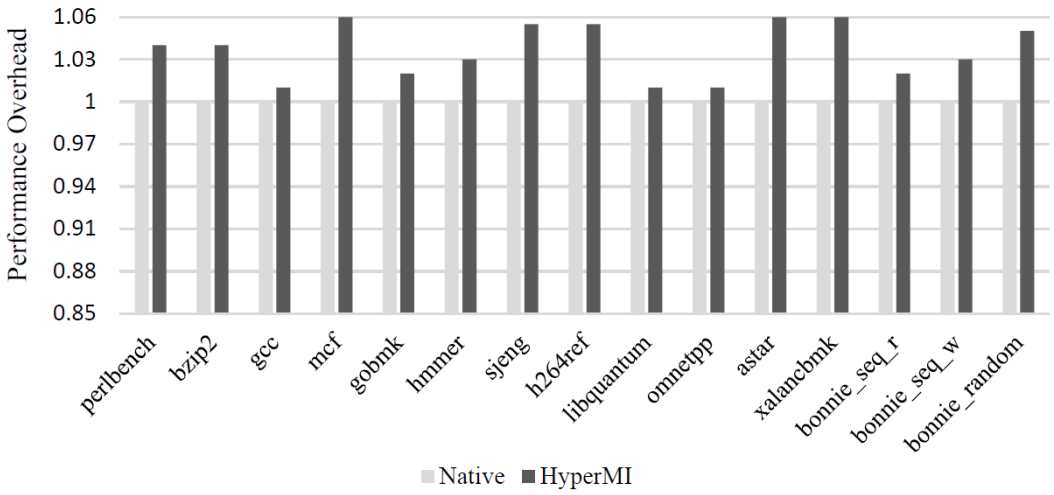

图 6.1 性能测试结果分析图

表 6.1 虚拟机启动关闭时间测试.

描述	启动(s)	关闭(s)
不启动HyperMI	11.79	1.75
启动HyperMI	12.97	1.89
效率	1.1	1.08

6.1.3 测试结果分析

6.1.3.1 基准工具测试

性能测试结果图6.1结果显示，与内存操作相关的性能开销大，与I/O 操作相关的性能开销小。分析知与内存操作监控较多，I/O监控操作较少，环境切换需刷新TLB、内存页频繁加载、EPT更新时验证Page信息（延迟内存EPT访问时间），这些都与内存访问相关。根据上图的测试结果，最高的性能开销是原KVM系统的1.06倍，主要是Mcf, astart, xalancbmk，这三种测试集主要的功能是大量分配内存，从而在EPT更新时访问Page Mark表，验证地址映射的正确性；访问Page Mark会导致环境切换；EPT更新会导致虚拟机退出/进入,并频繁访问VMCS这三个部分发生性能损耗。总体的性能是低于110%，满足性能需求。上图中的后三项数据是bonnie++的测试结果，分别进行了顺序读、顺序写以及随机写操作测试，这些测试是预先分配内存再进行读或者写操作。根据结果分析，性能开销不大于无HyperMI组件的KVM系统的105%，性能开销相对较低，低于110%，可以接受。

6.1.3.2 虚拟机启动关闭时间测试

测试结果如表6.1，通过程序运行过程分析，当宿主机安全启动后，HyperMI会创建新的一套页表完成安全执行环境的创建。随后，创建虚拟机，在启动虚拟机的过程

中，会分配大量的内存，引发虚拟机退出和虚拟机进入操作；同时在内存分配，页表更新的过程中访问Page Mark 表和VM Mark表，验证EPT更新的正确性和页更新的正确性；由于访问在虚拟机退出/进入和页表分配中，访问了位于HW中的VMCS和EPT，导致大量的环境切换，引发一定开销。综上，虚拟机启动的过程中会引发大量的环境切换、Page验证、虚拟机与Hypervisor的交互操作。

根据上述测试结果分析，当使用HyperMI时，给虚拟机启动和关闭带来的时间开销分别是1.1倍和1.08倍，相对较小，这种开销是可以接受的。

6.1.3.3 性能开销局限性及优化

环境切换是随着挂钩函数的多少变化，随着虚拟机退出/进入的次数变化而变化。必须控制挂钩函数的数量，并控制虚拟机退出发生的次数。根据系统设计分析，挂钩函数一般不需要再增加，当前的设计已经能够保证系统的安全运行。控制虚拟机退出发生的次数可以通过设置VMCS虚拟机退出控制域实现，减少不必要的虚拟机退出事件的发生。

6.2 安全评估

安全执行环境

创建一个在非可信虚拟化层环境下的相对安全的隔离的地址空间，该空间可以为后续的两个功能提供执行环境。地址隔离空间实现的目的是，当虚拟化层不安全，受到攻击者攻击的时候，另外的2类功能可以安全地运行在隔离的地址空间中，同时提供对系统中的关键数据的保护。

交互关键数据监控

Hypervisor在虚拟化底层为上层的虚拟机提供资源管理和资源分配的功能，所以Hypervisor与上层的虚拟机需要大量的交互过程，并且一个物理核每次只能运行一个系统，依据分时的方法，Hypervisor和虚拟机会在不同时间段使用物理核，所以它们之间需要一个安全的切换，防止在切换的过程中受到恶意攻击，避免不必要的危害。

虚拟机内存高强度隔离

上层虚拟机需要运行，其地址映射过程需要Hypervisor的参与。因为Hypervisor在虚拟化底层管理资源的分配，其中就包括内存资源的分配，那么地址映射的过程中会有Hypervisor的参与，最终上层虚拟机的虚拟地址会映射到真实的物理内存上。

功能需求

将虚拟机安全套件主要分为3个部分，分别提供对应的功能，保障其系统的安全性。三个功能分别是安全执行环境、交互关键数据监控、虚拟机内存高强度隔离。根据用户的需求，设计虚拟化安全套件的功能，功能要达到相对的完善性，能覆盖用户的所有需求。达到虚拟机隔离的目的，以及能够抵御用户需求相关领域的黑客攻击

6.2.1 安全性测试目标

此处安全测试即功能测试，涉及三个部分：安全执行环境、交互关键数据监控、虚拟机内存高强度隔离。需要对着三个功能模块进行测试，达到一定的功能和安全性需求。

安全执行环境功能测试要求在新的隔离的地址空间的数据不可被其余地址空间的进程访问，保证隔离地址空间的绝对隔离性，阻止攻击者破坏隔离地址空间，无法创建安全隔离的可执行环境。

交互关键数据监控功能测试要求虚拟机与宿主机的交互的关键数据在隔离地址空间中不被攻击者恶意访问、篡改。其中，交互关键数据包括虚拟机与宿主机在VCPU切换时的上下文信息，包含客户机和宿主机的特权寄存器、状态信息等。功能要求能够阻止泄露虚拟机和宿主机的关键信息，阻止控制流攻击等。根据测试要求，设计了攻击步骤，如表6.2。

表 6.2 篡改VMCS攻击步骤.

篡改VMCS攻击步骤
1、被hooked的函数vmcs_load，hooked后跳转到函数vmcs_load_attack
2、复制hook指令(mov vmcs_load_attack,eax;jmp eax)
3、覆盖vmcs_load函数的前8 bytes
4、在hooked后的函数中访问实参 *((*vmcs_signal).data)
5、实现攻击

虚拟机内存高强度隔离功能测试要求保证虚拟机的地址映射相关信息不被恶意访问、篡改等，能够保证虚拟机内存的高强度隔离，阻止攻击者通过其余被攻陷的虚拟机进行攻击，或者阻止攻击者通过在Hypervisor层上直接篡改地址映射数据（EPT），随后进行内存重映射或双映射攻击。表6.3描述了攻击EPT的步骤，主要是通过更改控制流进而访问EPT数据。

表 6.3 篡改EPT攻击步骤.

篡改EPT攻击步骤
1、被hooked的函数vmcs_load，hooked后跳转到函数vmcs_load_attack
2、复制hook指令(mov mmu_spte_walk_attack,eax;jmp eax)
3、覆盖mmu_spte_walk函数的前8 bytes
4、在hooked后的函数中访问实参(*vcpu).arch.mmu.root_level
5、实现攻击

表 6.4　测试环境.

资源名称/类型	配置
宿主机KVM	服务器，主频2.0GHZ，硬盘1T，内存32G，内核版本linux-4.4.1，系统版本ubutnu-14.05
客户机	内存8G，磁盘大小30G，系统版本ubuntu-12.04.5，2台

表 6.5　测试方法.

攻击	描述	测试功能	测试程序
DMA攻击	DMA方式访问安全区域	安全执行环境	Test_Isolation_DMA.c
代码注入攻击	注入代码，覆盖hooked的函数	安全执行环境	Test_Isolation_injection.c
CVE-2009-2287	加载恶意的客户机CR3寄存器	关键交互数据监控	Test_VMCS_CR3.c
CVE-2017-8106	加载恶意EPTP	虚拟机关键交互数据监控、虚拟机内存高强度隔离功能	Test_VMCS_EPTP.c

6.2.2　测试环境与配置

测试环境包括硬件环境和软件配置环境，概括如表6.4。

测试方法

分别实现表6.5中的攻击，攻击代码实现方式是通过LKM方式。

测试步骤

A.按照测试方法，分别执行表6.6中的4类攻击，主要方式是使用加载卸载LKM模块命令。

B.在卸载模块命令执行之后，使用dmesg命令查看攻击成功或失败等信息，测试结果详细分析如表6.7。

6.2.3　测试结果分析

从三个方面对整个HyperMI系统的安全功能进行了测试，针对安全执行环境模块。通过DMA攻击来测试IOMMU页表是否剔除掉关键数据的映射，是否严格监控了IO访问操作中地址验证功能；通过代码注入攻击能够检测在HW活跃时内核代码段是否是不

表 6.6　测试步骤.

攻击	加载模块	卸载模块	*dmesg*查看
DMA攻击	insmod DMA.ko	rmmod DMA.ko	dmesg grep ‘attackDMA’
代码注入攻击	insmod injection.ko	rmmod injection.ko	dmesg grep ‘attackInject’
CVE-2009-2287	insmod CR3.ko	rmmod cr3.ko	dmesg grep ‘attackCR3’
CVE-2017-8106	insmod EPTP.ko	rmmod EPTP.ko	dmesg grep ‘attackEPTP’

表 6.7 攻击案例.

攻击	描述	防护	测试
DMA攻击	DMA方式访问安全区域	DMA映射验证	安全执行环境
代码注入攻击	注入代码，覆盖hooked函数	页表访问防护	安全执行环境
CVE20092287	加载恶意CR3	特权寄存器监控	关键交互监控功能
CVE20178106	加载恶意EPTP	EPT、VMCS隐藏	关键交互监控功能、虚拟机内存高强度隔离功能

可执行的，并且能够保证代码段是不可写的；通过两个特殊的CVE攻击来检测交互功能和虚拟机内存隔离功能的安全性。经过分析，系统能够保证自身的安全性（隔离空间的安全性），能够保证交互数据安全性，能够保证内存的高强度隔离，满足系统的安全性需求。

6.3 小结

本章从两个方面对HyperMI框架所实现的系统进行了评估，性能测试评估和安全分析评估。性能测试主要是通过使用基准测试工具SPEC CPU2006进行整体性能测试，对虚拟机启动和关闭时间进行测试，安全测试和评估是通过四个案例充分测试并验证了HyperMI的安全性。这种方法相比较在系统上使用额外硬件方法的效率更高，更加方便[31],[26]，[48],[34]。

第七章　结论与展望

本文提出的一种基于同层地址空间隔离技术实现的虚拟机内存高强度隔离与防护框架HyperMI，并通过设计实验实现了对应的系统。将可信、可移植、平台实适用性作为基本属性添加到了虚拟机内存安全防护技术中，修补了之前各种研究方案不可移植的弊端。从原理上讲，HyeprMI以X86平台为实验平台，KVM作为虚拟机监控器，通过同层地址空间隔离技术创建了安全隔离的可信执行环境，该安全隔离执行环境与虚拟机监控器处于同一特权级别上，但是在不同的地址空间上；通过创建的隔离机制来保证虚拟机正常数据流和控制流的完整性以及使用的虚拟机高强度内存隔离中的Page-Mark表等关键数据的私密性，利用页表创建新的地址空间，利用对特权控制器寄存器的控制访问、DMA访问控制实现了对新隔离地址空间的安全防护；通过挂钩对虚拟机与虚拟机监控器交互函数进行监控，保证交互数据的保密性和完整性；通过页跟踪技术实现物理页的分类，并随时跟踪物理页的使用信息，保证物理页不被恶意访问，保证虚拟机内存高强度隔离。

为了更好、更详尽地把HyperMI的立意体现出来，体现其在商业云平台上的利用价值，本文对HyperMI的研究背景与意义、总体设计和详细设计、实现和评估进行了科学而严谨的证明与阐述，最终验证了HyperMI部署在当前云环境中的使用价值。本文介绍了云环境中日益严峻的用户敏感数据泄露问题和当前通用虚拟机隔离技术在安全脆弱的云环境中存在的严重的技术挑战和安全隐患。随后对现有虚拟机隔离技术和虚拟机监控器完整性防护的原理、发展和应用分支进行了详细介绍，总结出“对于云平台提供商，如何在安全脆弱的云环境中对虚拟机内存进行安全防护”这一课题研究和实现还不完善这一观点。其次，本文对当前国内外在虚拟机隔离技术和对虚拟机监控器完整性保护技术领域的发展进行了调查和研究，提出了本文的方案，随后对本文的方案进行了相应的比较和分析，总结出在云平台提供商角度上本文技术和方案设计的优点和长处。同时，本文通过对当前云平台上虚拟机监控器所面对的攻击威胁、同一物理平台上的虚拟机之间的攻击调研，分析当前云平台提供商使用的虚拟机隔离和安全防护方法，总结出虚拟机在内存方面可能受到攻击威胁。最后结合云环境下的基本安全要求和性能开销要求，提出了在非可信云环境下基于同层地址空间隔离技术实现的虚拟机内存高强度隔离的方案。在相关技术介绍部分，本文讲述了基本的虚拟机监控器的架构、虚拟机退出/进入事件等相关技术、虚拟机地址映射等相关背景知识。

在HyperMI的核心部分，设计和实现部分，本文首先讲述了HyperMI框架面对的整体威胁模型，涉及到虚拟机面对的威胁模型和HyperMI本身可能面对的威胁；随后讲述了HyperMI的设计在性能开销、代码设计、可移植性等方面需要遵循和满足的基本条

件和要求。在分析完威胁模型、明确性能和安全要求后，开始讲述了HyperMI将如何使用同层地址空间隔离技术实现虚拟机内存的安全防护。在系统总体架构和设计方案明确后，开始对框架进行系统实现描述。依次概述了HyperMI框架的三部分，同层地址空间隔离技术实现的安全可信执行环HyperMI World（HW）、在HW下实现对虚拟机和虚拟机监控器交互关键数据的监控模块（VM Monitor），在HW下实现对虚拟机内存的高强度隔离模块（VM Isolation）。在整个系统实现中，本文讲述了怎样使用同层隔离技术实现新的安全隔离地址空间HW，并且能够保证两个不同地址空间的安全切换，保证HW的安全性不被非可信Hypervisor破坏；讲述了如何对需要监控的事件在HW中进行挂钩，如何对VMCS进行保护和隐藏；讲述了如何在HW中实现对物理页的分类，如何对EPT进行保护和隐藏，如何抵御内存越权访问攻击。

在评估部分，本文通过对HyperMI的两个方面进行了评估分析，分别是性能评估和安全分析，通过多个测试案例对HyperMI的有效性进行了分析和验证。首先进行了安全分析，从对HyperMI自身的安全防护（HW的隔离和安全性分析）、HyperMI对虚拟机和虚拟机监控器交互数据的监控（交互数据的保密性和完整性）、虚拟机之间内存安全隔离三个方面进行分析。通过测试系统整体性能开销、虚拟机启动加载时间、虚拟机进入/退出性能开销三个角度考察HyperMI的性能。最后对HyperMI的不足进行了讨论。最后，本文对HyperMI的总体设计与实现进行总结，发现HyperMI不同与以往的虚拟机隔离技术和虚拟机监控器保护技术，HyperMI在在性能开销和平台适用性的应用场景更具有普适性：在虚拟机监控器不可信的条件下，能够保障虚拟机内存数据安全；在能够保证虚拟机内存高强度隔离和虚拟机运行状态数据安全下，对硬件平台没有依赖性，系统的可移植性强。这对虚拟机安全研究有一定的意义。

从长远角度来看，结合当前脆弱云环境下在安全方面的防护机制不完善，以及云平台提供商的使用需求，HyperMI在云环境中的部署是极有前景的。在非可信的虚拟化环境中，可以部署一套不依赖硬件平台和虚拟机监控器的可信虚拟机内存监控系统成为了可能。

参考文献

[1] DENG L, LIU P, XU J, et al. Dancing with Wolves: Towards Practical Event-driven VMM Monitoring[J]. Acm Sigplan Notices, 2017, 52(7):83–96.

[2] CVE. 2018. Accessed March 1, 2019.http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=cve.

[3] CVE-2017-8106. 2017. Accessed March 1, 2019.http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-8106.

[4] ATAMLI-REINEH A, MARTIN A. Securing Application with Software Partitioning: A Case Study Using SGX[C]//International Conference on Security and Privacy in Communication Systems. .[S.l.]: [s.n.] , 2015.

[5] BAUMANN A, PEINADO M, HUNT G. Shielding Applications from an Untrusted Cloud with Haven[J]. Acm Transactions on Computer Systems, 2014, 33(3):1–26.

[6] AZAB A M, PENG N, ZHI W, et al. HyperSentry:enabling stealthy in-context measurement of hypervisor integrity[C]//. .[S.l.]: [s.n.] , 2010.

[7] JANG J, CHOI C, LEE J, et al. PrivateZone: Providing a Private Execution Environment using ARM TrustZone[J]. IEEE Transactions on Dependable and Secure Computing, 2016, PP(99):1–1.

[8] ZHANG F, CHEN J, CHEN H, et al. CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization[C]//ACM Symposium on Operating Systems Principles. .[S.l.]: [s.n.] , 2011:203–216.

[9] CHENG T, XIA Y, CHEN H, et al. TinyChecker: Transparent protection of VMs against hypervisor failures with nested virtualization[C]//IEEE/IFIP International Conference on Dependable Systems and Networks Workshops. .[S.l.]: [s.n.] , 2012.

[10] SHARIF M I, LEE W, CUI W, et al. Secure in-VM monitoring using hardware virtualization[C]//Acm Conference on Computer and Communications Security. .[S.l.]: [s.n.] , 2009.

[11] CIPRESSO P, ALBANI G, SERINO S, et al. Virtual multiple errands test (VMET): a virtual reality-based tool to detect early executive functions deficit in Parkinson’s disease[J]. Frontiers in Behavioral Neuroscience, 2014, 8(405):1–11.

[12] SHI L, WU Y, XIA Y, et al. Deconstructing Xen[C]//Network and Distributed System Security Symposium. .[S.l.]: [s.n.] , 2017.

[13] WANG X, CHEN Y, WANG Z, et al. SecPod: a framework for virtualization-based security systems[C]//Usenix Conference on Usenix Technical Conference. .[S.l.]: [s.n.] , 2015:347–360.

[14] KELLER E, SZEFER J, REXFORD J, et al. NoHype:virtualized cloud infrastructure without the virtualization[J]. Acm Sigarch Computer Architecture News, 2010, 38(3):350–361.

[15] JIN S, AHN J, SEOL J, et al. H-SVM: Hardware-Assisted Secure Virtual Machines under a Vulnerable Hypervisor[J]. IEEE Transactions on Computers, 2015, 64(10):2833–2846.

[16] CHO Y, SHIN J, KWON D, et al. Hardware-assisted on-demand hypervisor activation for efficient security critical code execution on mobile devices[C]//Usenix Conference on Usenix Technical Conference. .[S.l.]: [s.n.] , 2016:565–578.

[17] MCKEEN F, ALEXANDROVICH I, BERENZON A, et al. Innovative instructions and software model for isolated execution[C]//International Workshop on Hardware and Architectural Support for Security and Privacy. .[S.l.]: [s.n.] , 2013:1–1.

[18] HOEKSTRA M, LAL R, ROZAS C, et al. Cuvillo, "Using Innovative Instructions to Create Trustworthy Software Solutions," in Hardware and Architectural Support for Security and Privacy[C]//6 IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. .[S.l.]: [s.n.] , 2013.

[19] MOON H, LEE H, LEE J, et al. Vigilare: toward snoop-based kernel integrity monitor[C]//ACM Conference on Computer and Communications Security. .[S.l.]: [s.n.] , 2012:28–37.

[20] LEE H, MOON H, JANG D, et al. KI-Mon: a hardware-assisted event-triggered monitoring platform for mutable kernel object[C]//Usenix Conference on Security. .[S.l.]: [s.n.] , 2013:511–526.

[21] WANG J, STAVROU A, GHOSH A. HyperCheck: a hardware-assisted integrity monitor[C]//International Conference on Recent Advances in Intrusion Detection. .[S.l.]: [s.n.] , 2010:158–177.

[22] PETRONI N L, HICKS M. Automated detection of persistent kernel control-flow attacks[C]//ACM Conference on Computer and Communications Security. .[S.l.]: [s.n.] , 2007:103–115.

[23] AZAB A, SWIDOWSKI K, BHUTKAR R, et al. SKEE: A Lightweight Secure Kernel-level Execution Environment for ARM[C]//Network and Distributed System Security Symposium. .[S.l.]: [s.n.] , 2016.

[24] KOURAI K, CHIBA S. HyperSpector:virtual distributed monitoring environments for secure intrusion detection[C]//Proc of Acm/usenix International Conference on Virtual Execution Environments. .[S.l.]: [s.n.] , 2005:197–207.

[25] BAHRAM S, JIANG X, ZHI W, et al. DKSM: Subverting Virtual Machine Introspection for Fun and Profit[C]//IEEE Symposium on Reliable Distributed Systems. .[S.l.]: [s.n.] , 2010.

[26] EVTYUSHKIN D, ELWELL J, OZSOY M, et al. Iso-X:A Flexible Architecture for Hardware-Managed Isolated Execution[C]//Ieee/acm International Symposium on Microarchitecture. .[S.l.]: [s.n.] , 2015:190–202.

[27] WANG Z, JIANG X. HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity[C]//Security and Privacy. .[S.l.]: [s.n.] , 2010:380–395.

[28] RHEE J, RILEY R, XU D, et al. Defeating Dynamic Data Kernel Rootkit Attacks via VMM-Based Guest-Transparent Monitoring[C]//International Conference on Availability, Reliability and Security. .[S.l.]: [s.n.] , 2009:74–81.

[29] SALTZER J H. Protection and the Control of Information Sharing in MULTICS[C]//Acm Symposium on Operating System Principles. .[S.l.]: [s.n.] , 1973:119.

[30] CHO Y, KWON D, YI H, et al. Dynamic Virtual Address Range Adjustment for Intra-Level Privilege Separation on ARM[C]//24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017. .[S.l.]: [s.n.] , 2017, https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/dynamic-virtual-address-range-adjustment-intra-level-privilege-separation-arm/.

[31] CHHABRA S, ROGERS B, YAN S, et al. SecureME:a hardware-software approach to full system security[C]//International Conference on Supercomputing. .[S.l.]: [s.n.] , 2011:108–119.

[32] DAUTENHAHN N, KASAMPALIS T, DIETZ W, et al. Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation[J]. Acm Sigplan Notices, 2015, 50(4):191–206.

[33] WANG B, ZHENG Y, LOU W, et al. DDoS attack protection in the era of cloud computing and Software-Defined Networking[J]. Computer Networks the International Journal of Computer & Telecommunications Networking, 2015, 81(C):308–319.

[34] STEINBERG U, KAUER B. NOVA:a microhypervisor-based secure virtualization architecture[C]//European Conference on Computer Systems, Proceedings of the European Conference on Computer Systems, EUROSYS 2010, Paris, France, April. .[S.l.]: [s.n.] , 2010:209–222.

[35] BEN-YEHUDA M, DAY M, DUBITZKY Z, et al. The turtles project: Design and implementation of nested virtualization[J]. Yehuda, 2007:1–6.

[36] CRISWELL J, LENHARTH A, DHURJATI D, et al. Secure virtual architecture:a safe execution environment for commodity operating systems[J]. Acm Sigops Operating Systems Review, 2007, 41(6):351–366.

[37] GARFINKEL T, PFAFF B, CHOW J, et al. Terra:a virtual machine-based platform for trusted computing[C]//Nineteenth Acm Symposium on Operating Systems Principles. .[S.l.]: [s.n.] , 2003:193–206.

[38] JIANG X, WANG X, XU D. Stealthy malware detection through vmm-based ”out-of-the-box” semantic view reconstruction[C]//ACM Conference on Computer and Communications Security. .[S.l.]: [s.n.] , 2007:128–138.

[39] GARFINKEL T. A Virtual Machine Introspection Based Architecture for Intrusion Detection[J]. Proc.network & Distributed Systems Security Symp, 2003:191–206.

[40] JONES S T, ARPACI-DUSSEAU A C, ARPACI-DUSSEAU R H. Antfarm: Tracking Processes in a Virtual Machine Environment.[J]. Proc.annual Usenix Tech.conf.usenix Assoc, 2006, 56:1–14.

[41] SESHADRI A, LUK M, QU N, et al. SecVisor:a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes[J]. ACM SIGOPS Operating Systems Review, 2007, 41(6):335–350.

[42] WANG X, QI Y, DAI Y, et al. TrustOSV: Building Trustworthy Executing Environment with Commodity Hardware for a Safe Cloud[J]. Journal of Computers, 2014, 9(10).

[43] 系统虚拟化 : 原理与实现[M].[S.l.]: [s.n.] , 2009.

[44] ZHU M, TU B, WEI W, et al. HA-VMSI: A Lightweight Virtual Machine Isolation Approach with Commodity Hardware for ARM[C]//ACM Sigplan/sigops International Conference on Virtual Execution Environments. .[S.l.]: [s.n.] , 2017:242–256.

[45] MCCUNE J M, LI Y, QU N, et al. TrustVisor: Efficient TCB Reduction and Attestation[J]. Cylab, 2010, 41(3):143–158.

[46] WANG Z, WU C, GRACE M, et al. Isolating commodity hosted hypervisors with HyperLock[C]//Proceedings of the 7th ACM european conference on Computer Systems. .[S.l.]: [s.n.] , 2012:127–140.

[47] PAYNE B D, CARBONE M, SHARIF M, et al. Lares: An Architecture for Secure Active Monitoring Using Virtualization[J]. 2008:233–247.

[48] CHAMPAGNE D, LEE R B. Scalable architectural support for trusted software[C]//HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture. .[S.l.]: [s.n.] , 2010:1–12.

致　谢

在整个研究生生涯中，感谢我的导师在学业上对我的教导，不断的督促和诲人不倦的谆谆教导。感谢实验室的同学给予的学术和生活的帮助，感谢父母对自己的支持。

当毕业论文最后一个字符敲定，硕士研究生学习阶段即将完成，细细品味读书三年所得，在专业知识，云计算领域，操作系统内核内存管理方向上稍微了解，还需要继续深究。

三载一瞬，虽不是学富五车，捆载而归，但终究得益于重书累牍之中，领略到网络空间安全这个专业领域中有趣的一面，上到势态感知、应用安全、威胁攻击溯源追踪、安卓软件安全等应用层面，下到操作系统安全、虚拟化安全、安卓系统安全、内核漏洞挖掘。当前各大公司这些领域发生的各种安全问题以及对应的解决方案，无不体现了计算机安全的发展趋势和重要性。有幸能选择这样的专业进行学习。回顾整个研究生历程，首先在中国科学院大学雁栖湖校区进行了为期一年的学习，主要学习专业知识，从大数据分析、深度学习与安全，到云计算安全、操作系统安全、网络系统安全，再到网络溯源与跟踪、恶意代码分析，涉及大数据、系统安全、恶意代码分析等各个方面，为后两年进实验室的研究学习进行准备和知识扩充，扩展自身的学习知识面并加深对计算机安全的了解。其次，在进实验室后，有幸选择了虚拟化安全方向，主要在操作系统内存管理方向进行研究，完成了基本项目，涉猎了内存中的部分内容，其余的还待继续深究。在整个研究生生涯中，首先感谢我的导师涂碧波老师，感谢其诲人不倦和精益求精的精神，从研究生研究方向选择、项目指导、组会教导、研究生论文选题，到学术论文指导，都细心指导。

感谢课题组的所有同学们，在项目上、学术方面的指导，在生活上的帮助，感谢父母和妹妹在生活、工作上的指导和帮助。

作者简介

姓名：刘文清　　性别：女　　出生日期：1993.11.24　　籍贯：山西

2016.9 – 现在　　中国科学院大学信息工程研究所攻读硕士研究生
2012.9 – 2016.7　　同济大学本科生

【攻读硕士学位期间发表的专利】

[1] 实现虚拟机安全隔离的方法与装置（已申请）

【攻读硕士学位期间参加的科研项目】

[1] 参与国家重点研发计划项目(课题号2016YFB0801002)，完成子项目“虚拟机安全套件” 2017.10至今

【攻读硕士学位期间的获奖情况】

[1] 2018年中国科学院大学“三好学生”
[2] 2017年第十四届“华为杯”全国研究生数学建模竞赛三等奖