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研究

作者姓名：刘文清

指导教师：涂碧波 研究员

中国科学院信息工程研究所

学位类别：工学硕士

学科专业：网络空间安全

培养单位：中国科学院信息工程研究所

2019 年 6 月

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

A thesis submitted to the

University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Science

in Cyberspace Security

By

Liu Wenqing

Supervisor: Professor Tu Bibo

Institute of Information Engineering

Chinese Academy of Sciences

June, 2019

中国科学院大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。本人完全意识到本声明的法律结果由本人承担。

作者签名：

日　　期：

中国科学院大学
学位论文授权使用声明

本人完全了解并同意遵守中国科学院大学有关保存和使用学位论文的规定，即中国科学院大学有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：　　导师签名：

日　　期：　　日　　期：

摘 要

当远程或者本地攻击者攻陷 Hypervisor 后，就可以攻陷 Hypervisor 上的客户虚拟机或者恶意访问系统信息。由于用户的信息都保存在虚拟机的内存上，攻击者可以通过 Hypervisor 恶意访问客户机的内存数据，甚至篡改 Hypervisor 与虚拟机交互的上下文，最终泄露敏感关键信息。为了保护虚拟机的信息，一些研究学者采用定制硬件的方法来监控 Hypervisor 对关键信息的访问，该方法对于云平台提供商移植性不强。部分研究学者采用高于 Hypervisor 特权层的软件层方法，即更高层软件层，用来控制对关键数据的访问操作，从而能够阻止攻击者利用 Hypervisor 实施进一步的攻击，这样会带来一定的特权层之间的切换开销。

为了在 Hypervisor 不可信的前提下，实现对虚拟机内存安全和隔离功能，本文提出了 HyperMI 框架，基于同层地址空间隔离创建一个安全可信执行环境，提供对 Hypervisor 关键交互数据的访问监控和虚拟机内存的高强度隔离。本文的主要工作及创新点如下：

- 安全执行环境：通过同层地址空间隔离技术创建与 Hypervisor 位于同一特权层的隔离地址空间，且保证地址空间的安全性和隔离性，阻止不可信 Hypervisor 对该空间数据和代码的恶意访问。
- Hypervisor 关键交互数据监控：通过剥夺 Hypervisor 对关键交互数据（VMCS）的访问功能，实现上下文安全切换和虚拟机退出重定向，保证虚拟机运行时状态安全，最终避免系统信息泄露威胁。
- 虚拟机内存高强度隔离：通过内存动态标记与跟踪关键技术和虚拟机地址映射监控，实现虚拟机内存高强度隔离，当物理页被分配时能够保证虚拟机只能访问属于自身的内存页，阻止恶意的内存越权访问攻击。

HyperMI 框架对平台不过度依赖，可移植性好，适用于商业的云平台提供商。安全分析表明该框架可以提供有效的 Hypervisor 关键交互数据监控和虚拟机内存高强度隔离，性能测试表明该框架具有高效性。

关键词：虚拟化，虚拟机内存防护，虚拟机安全

Abstract

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 Memory Protection, VM Security

目 录

图形列表

表格列表

符号列表

字符

Symbol	Description	Unit
R	the gas constant	$m^2 \cdot s^{-2} \cdot K^{-1}$
C_v	specific heat capacity at constant volume	$m^2 \cdot s^{-2} \cdot K^{-1}$
C_p	specific heat capacity at constant pressure	$m^2 \cdot s^{-2} \cdot K^{-1}$
E	specific total energy	$m^2 \cdot s^{-2}$
e	specific internal energy	$m^2 \cdot s^{-2}$
h_T	specific total enthalpy	$m^2 \cdot s^{-2}$
h	specific enthalpy	$m^2 \cdot s^{-2}$
k	thermal conductivity	$kg \cdot m \cdot s^{-3} \cdot K^{-1}$
S_{ij}	deviatoric stress tensor	$kg \cdot m^{-1} \cdot s^{-2}$
τ_{ij}	viscous stress tensor	$kg \cdot m^{-1} \cdot s^{-2}$
δ_{ij}	Kronecker tensor	1
I_{ij}	identity tensor	1

算子

Symbol	Description
Δ	difference
∇	gradient operator
$\delta^{\pm}$	upwind-biased interpolation scheme

缩写

CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
EOS	Equation of State

JWL	Jones-Wilkins-Lee
WENO	Weighted Essentially Non-oscillatory
ZND	Zel'dovich-von Neumann-Doering

第 1 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Deng 等 (2017) 和 33.6 万行Deng 等 (2017)，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CVE (2018)，参看图1.1，比如，CVE-2017-1087CVE-2017-8106 (2017) 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

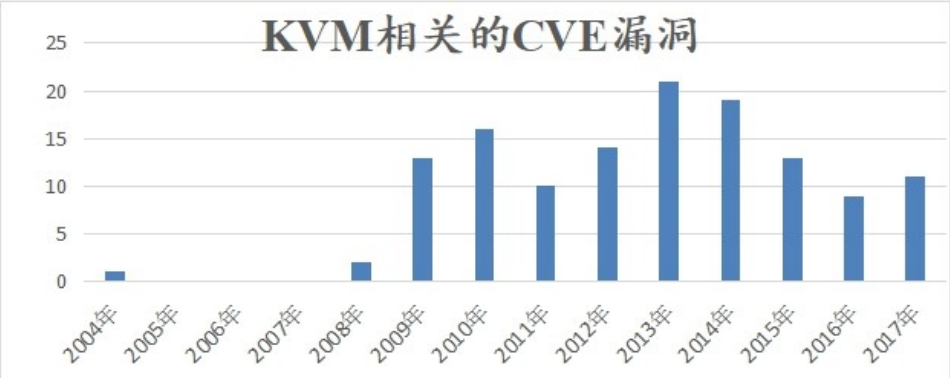

图 1.1 2004-2017 年 KVM 相关漏洞.

Figure 1.1 The related vulnerabilities about KVM from 2014 to 2017.

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。

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，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。

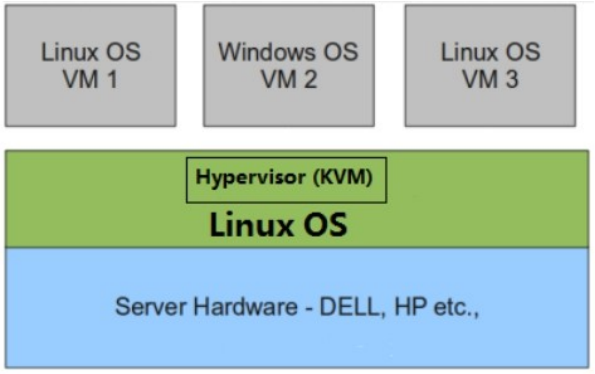

图 1.2 KVM 架构图.

Figure 1.2 The architecture of KVM.

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。

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，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 Trust Zone、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 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Atamli-Reineh 和 Martin (2015)。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，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Baumann 等 (2014)，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Azab 等 (2010)，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Jang 等 (2016)，将虚拟机交互数据和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 CloudVisorZhang 等 (2011) 和 TinycheckerCheng 等 (2012)，都是使用更高特权级别的软件控制对低特权级的访问Sharif 等 (2009)。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-MetCipresso 等 (2014) 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 NexenShi 等 (2017)，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 SecpodWang 等 (2015b)，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 NoHypeKeller 等 (2010) 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-SVMJin 等 (2015) 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。可信执行环境包含隔离空间实现Cho 等 (2016); Mckeen 等 (2013); Hoekstra 等 (2013)，定制硬件实现隔离空间Moon 等 (2012); Lee 等 (2013) 等。对 VM 的防护主要是重组 Hypervisor，实现对 Hypervisor 的完整性保护Hoekstra 等 (2013); Wang 等 (2010); Petroni 和 Hicks (2007)，实现不同组件和功能。

1.2.1 隔离空间实现

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，它是基于 ARM 平台的，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。SKEEAzab 等 (2016) 提供了一套保证 ARM 平台上 Kernel 地址隔离的新技术。它创建了一个恶意 Kernel 无法访问的受保护地址空间 (SK)，并且 Kernel 和 SK 共享相同的特权级别，即同层隔离思想。SKEE 通过阻止 Kernel 管理 MMU（内存管理单元）来实现地址空间隔离，那么 Kernel 必须请求 SKEE 来管理 MMU，于是 SKEE 会验证有关 MMU 管理和访问的所有操作，从而保证 SK 这个地址空间不会被恶意的 Kernel 访问。从 Kernel 到 SKEE 的切换只通过一个安全开关门，切换过程是原子性和确定性的，能够保证

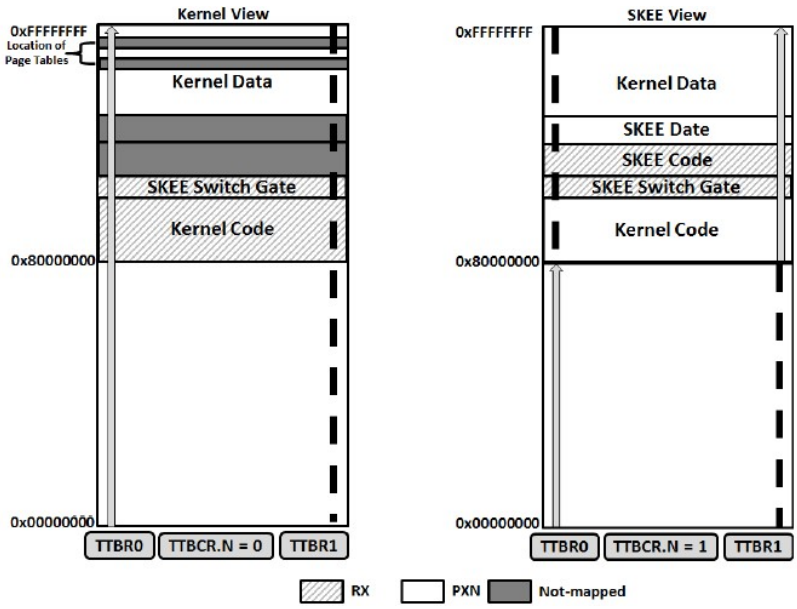

图 1.3 SKEE 隔离空间实现.

Figure 1.3 The achievement of isolation space of SKEE.

了潜在的非可信 Kernel 不能利用交换序列来破坏 SK 的隔离。如果恶意的 Kernel 试图通过破坏切换过程来破坏 SK 的隔离性，系统会识别并阻止切换。

SKEE 可以控制 Kernel 对内存的访问权限。因此，它可以防止未验证的代码注入攻击，并且在 SK 中拦截其他系统事件，以支持各种入侵检测Kourai 和 Chiba (2005); Bahram 等 (2010) 和完整性验证工具。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-MonitorDeng 等 (2017)，用于云计算中对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。

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

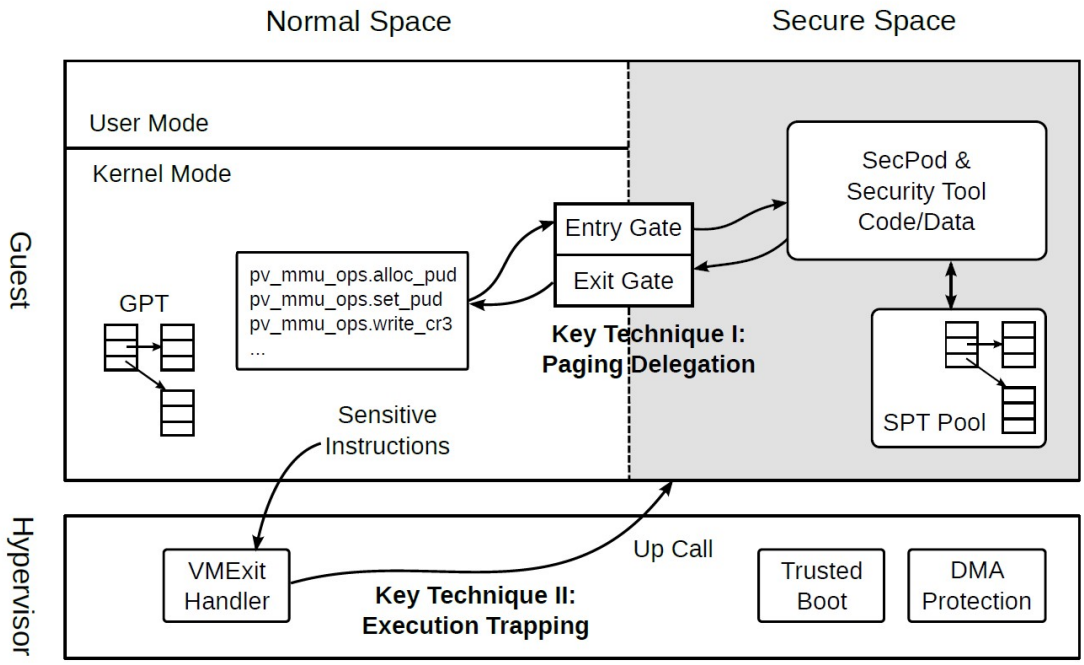

图 1.4 Secpod 系统架构图.

Figure 1.4 The architecture of Secpod.

ED-Monitor 提供保护。

1.2.1.3 Secpod

操作系统内核安全对计算机系统的安全至关重要。先前许多系统已被提出以提高内核的安全性，但是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，因此页表会受到篡改。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。然而这种内存保护需要监视对客户机页表的每次更新。这从根本上与硬件虚拟化支持的最新进展相冲突。论文学者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 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）内核使用的一些特权指令。

正如图1.4所示，共两个地址空间，普通空间和安全空间（Normal Space 和 Secure Space）。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）。

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主要依赖于软件管理程序及其具有 Root 权限的系统管理员，很容易被攻击。但是，如果管理程序（Hypervisor）被攻击，则攻击者可以完全访问虚拟机的内存和上下文。学者提出了一种基于硬件的方法（H-SVMJin 等 (2015)）来保护客户虚拟机。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，该机制基于额外硬件成本扩展和嵌套分页的内存虚拟化硬件。

H-SVMJin 等 (2015) 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，此时的数据验证无法通过，所以读访问篡改无意义。

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。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

NeXen 框架由上海交大并行与系统安全实验室提出。由于 Hypervisor 缺乏特权安全监视和分解策略（代码基过于庞大），它很容易受到攻击。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 个）位于核心虚拟机监控程序中，而不是 Dom0。然后，将 Xen 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。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 Xen 漏洞的 74%（107/144），并保证 Xen 代码完整性（防御所有的代码注入危害）。

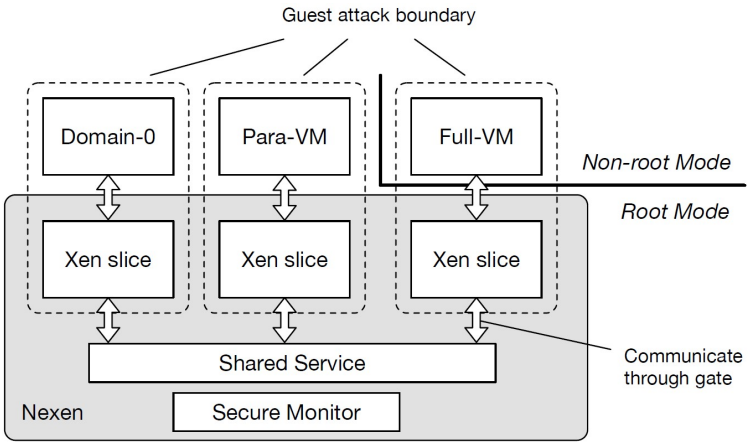

图 1.5 Nexen 系统架构图.

Figure 1.5 The architecture of Nexen.

当前，虚拟机最容易受到的攻击是 DOS/DDOS 攻击，为了避免 VM 受到攻击后可能对整个 Hypervisor 或者其余的 VM 造成威胁，论文中学者提出一旦 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1.5，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。

1.2.3.2 Nohype

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，恶意攻击者就有机会攻击虚拟化层。成功的攻击会让恶意攻击者控制虚拟化层，可能会损害任何虚拟机的软件和数据的机密性和完整性。学者建议删除虚拟化层，同时保留虚拟化所支持的关键特性。NohypeKeller 等 (2010) 处理虚拟化层的每个关键功能，CPU、IO、网络访问等。

1.2.3.3 HyperCheck

之前的研究工作中，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恶意的软件组件的深度隔离Wang 等 (2010)。然而广泛采用 VMMs 促使虚拟机成为攻击者的主要目标。HyperCheck 旨在保护 VMM 的完整性，并

针对某些类型的攻击，提供底层微型操作系统（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。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。

1.2.3.4 HyperSentry

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-XEvtyushkin 等 (2015) 中的定制硬件。相反，HyperSentry 引入了一个软件组件，该组件与 Hypervisor 隔离，以实现对 Hypervisor 运行时状态完整性的隐藏和上下文监测，保证在检测到即将到来的代码监测时 Hypervisor 没有机会隐藏攻击痕迹。

1.2.3.5 HyperSafe

HyperSafeWang 和 Jiang (2010) 通过提供控制流完整性，使现有的 I 型裸机管理程序具有自我保护能力。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，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，从而扩展保护以控制流完整性。

Rootkit 主要是通过修改内核的控制结构和钩子来隐藏自身。之前的研究工作是通过 Hypervisor 来实现对 Rootkit 的监控Rhee 等 (2009)，由于系统中的钩子和控制结构是零散的、分散的，HyperSafe 采用另一种方法———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，验证对钩子的写操作。

1.2.3.6 Hilps

对于一体化的操作系统，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Saltzer (1973)。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专注于 ARM 体系结构，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，随着针对每一级特权软件的恶意软件和攻击的增加，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

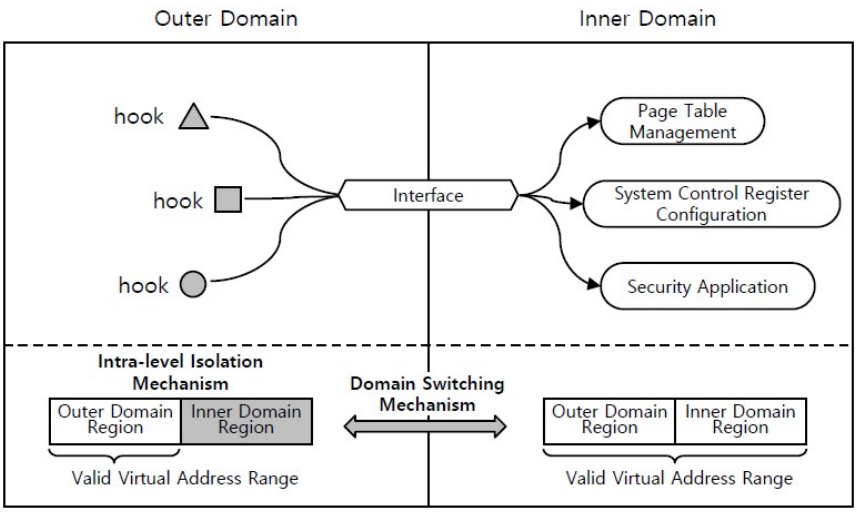

图 1.6 Hilps 特权分离原理图.

Figure 1.6 The privilege separation principle of Hilps.

HilpsRhee 等 (2009) 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。

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1.6，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、系统控制寄存器、安全应用处理函数，处理结束后再跳转回到外部区域。

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Jin 等 (2015); Chhabra 等 (2011)，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Shi 等 (2017); Keller 等 (2010), 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 HyperMI，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。HyperMI 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 2 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 HypervisorWang 和 Jiang (2010); Dautenhahn 等 (2015); Jin 等 (2015)。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Wang 等 (2015a)。详细的攻击方式如下介绍。

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 HypervisorSteinberg 和 Kauer (2010),Ben-Yehuda 等 (2007)。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

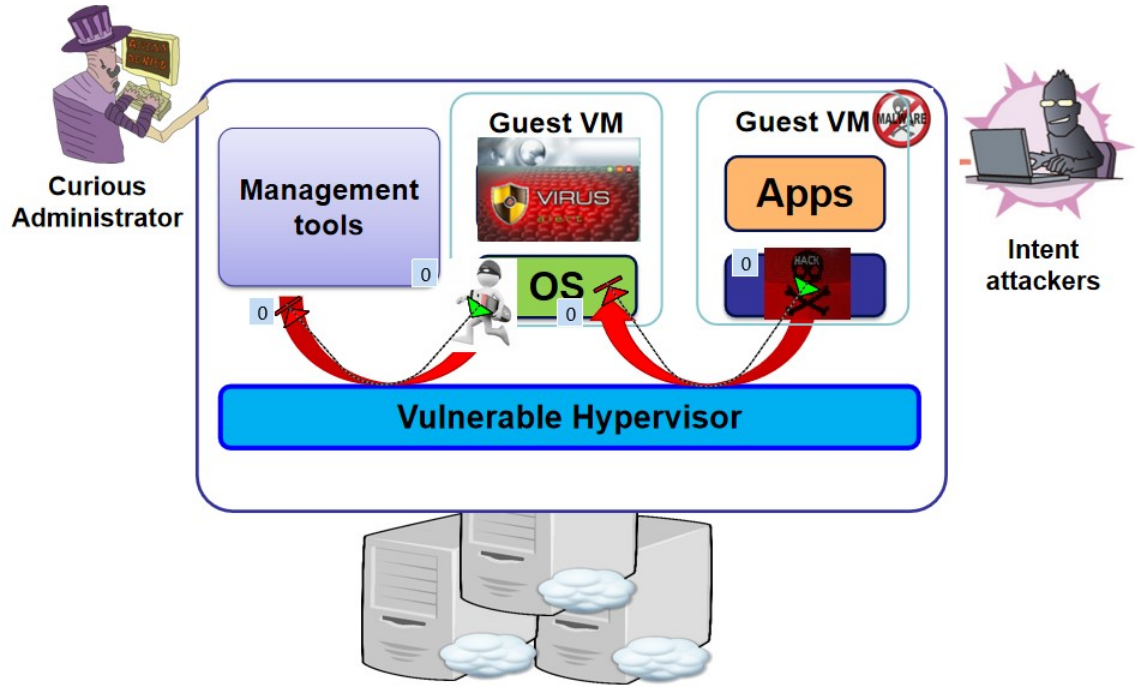

图 2.1 威胁模型.

Figure 2.1 Threat model.

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

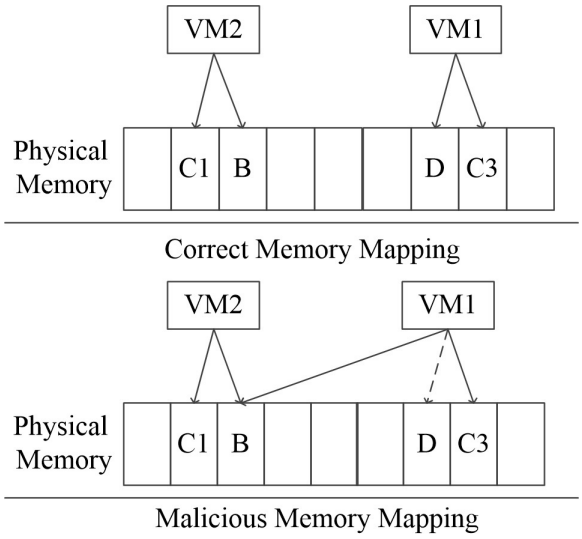

图 2.2 重映射攻击.

Figure 2.2 Remapping attack.

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

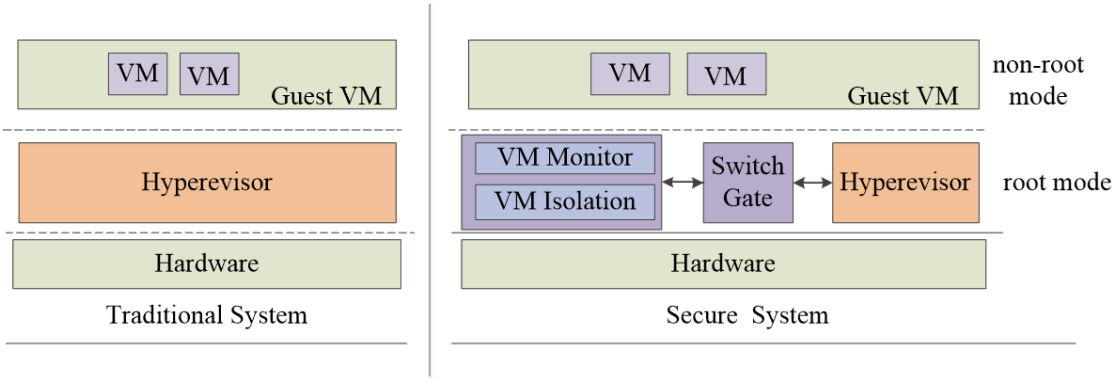

图 2.3 系统概述图.

Figure 2.3 The overview of system.

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。

系统执行流程：加载 HyperMI 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 HyperMI 系统模块，结束系统运行。

2.3.2 子系统间关系

HyperMI 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 HyperMI 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，监控的内容是 Hypervisor 和 VM 之间的交互关键数据，即切换上下文（VMCS），分为监控虚拟机上下文切换和 VM 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

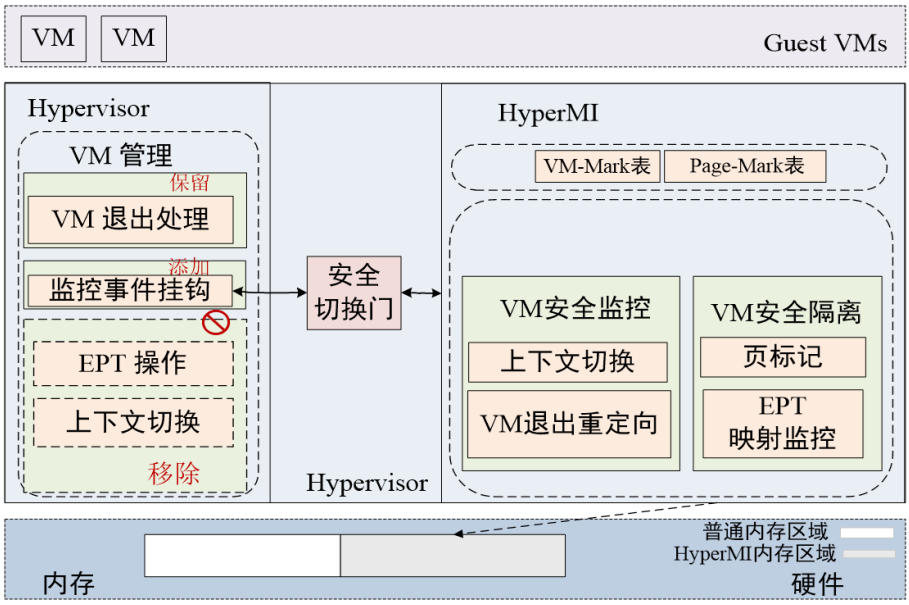

图 2.4 系统架构图.

Figure 2.4 The system architecture.

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 中有系统使用的关键数据（Page-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。

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 HyperMI 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

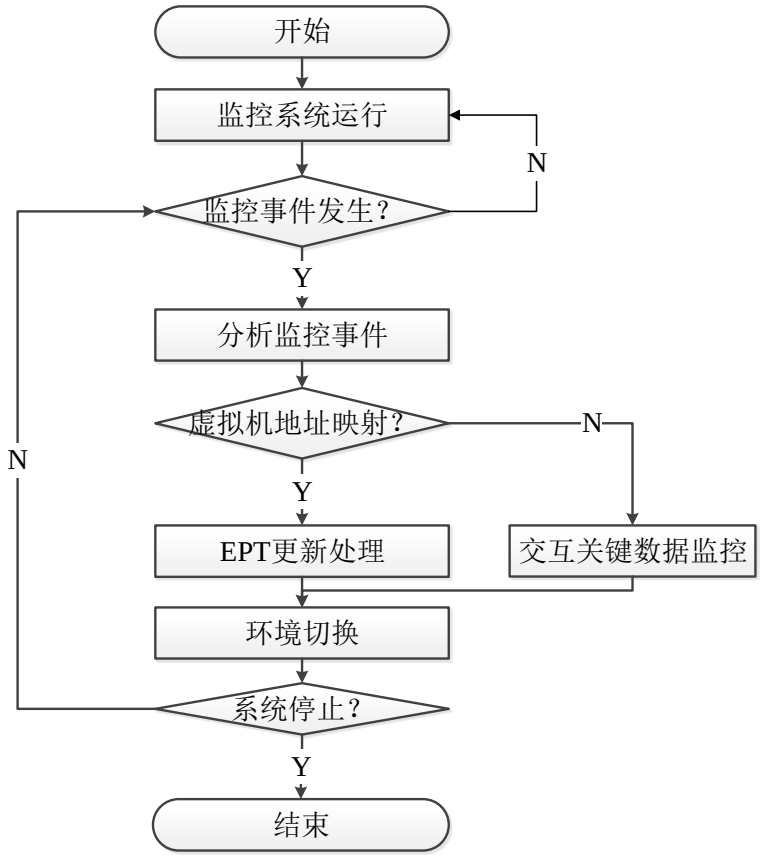

图 2.5 控制流程图.

Figure 2.5 The control process.

射模块为虚拟机内存提供安全隔离和访问控制。

加载 HyperMI 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 HyperMI 系统模块，结束系统运行。控制流程如图2.5。

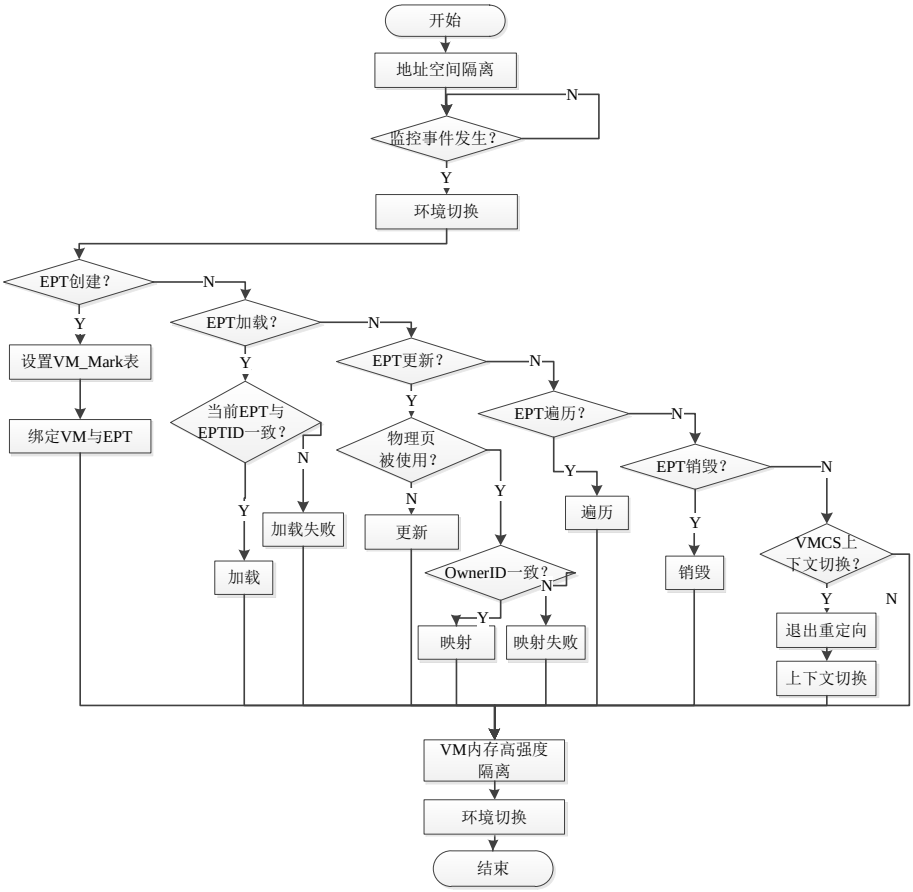

图 2.6 监控事件执行流程图.

Figure 2.6 The monitor event execution process.

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2.6。

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 3 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Criswell 等 (2007); Garfinkel 等 (2003); Jiang 等 (2007)，用来实现软件层完整性验证、入侵检测等Rhee 等 (2009); Garfinkel (2003); Jones 等 (2006); Seshadri 等 (2007)。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，产生相对较低的性能开销。

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，基本技术点包含执行环境创建、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

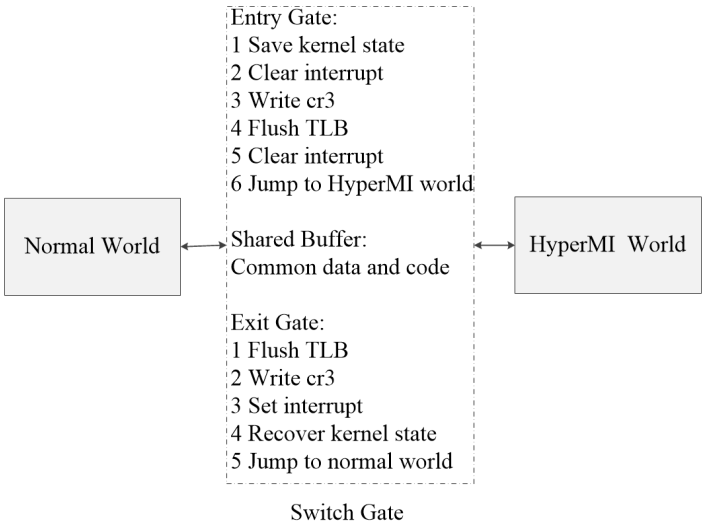

图 3.1 安全切换门.

Figure 3.1 The switch gate.

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。

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（控制寄存器、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Wang 等 (2014)，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，控制寄存器存放页表入口地址，那么就需要对新页表、控制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，防护措施主要是三点：对新页表的访问控制、控制寄存器访问控制、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，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 控制寄存器访问控制

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

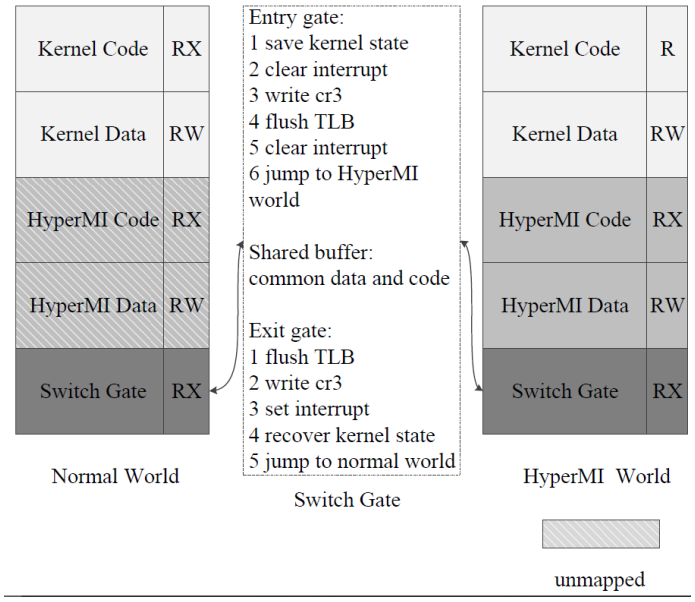

图 3.2 页表安全防护.

Figure 3.2 The protection for page table.

中运行。

3.3.3 DMA 访问控制

DMA 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地址，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 4 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200(2009)。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 VM Exit 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

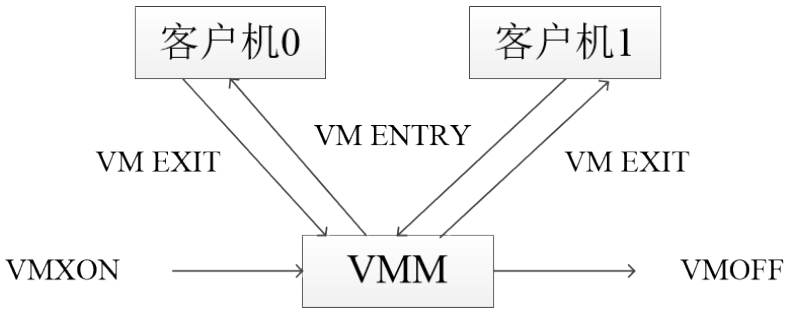

图 4.1 VMX 操作模式.

Figure 4.1 VMX operation mode.

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 Guest 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。

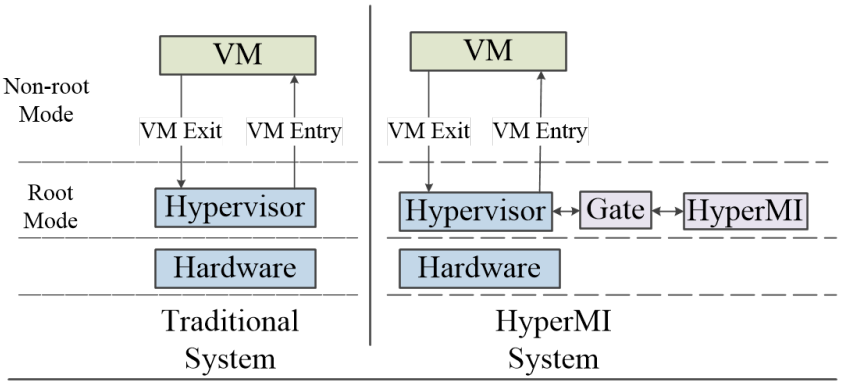

图 4.2 虚拟机与宿主机交互监控过程示意图.

Figure 4.2 The interaction monitor between VM and Hypervisor.

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。

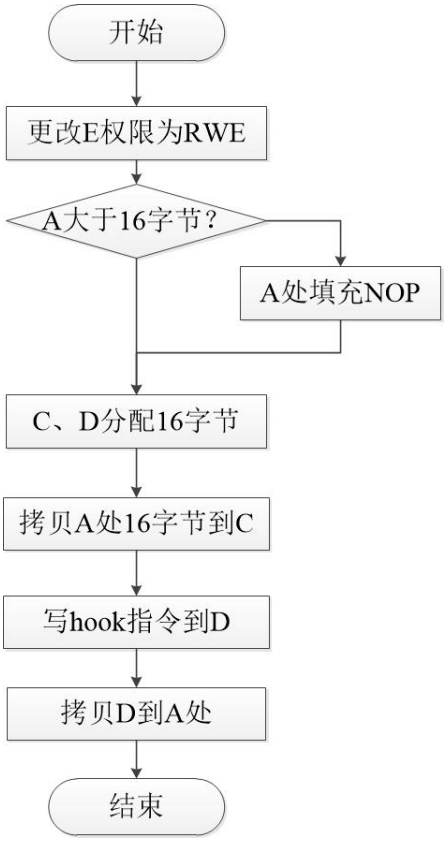

图 4.3 HOOK 算法图.

Figure 4.3 The HOOK algorithm.

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。

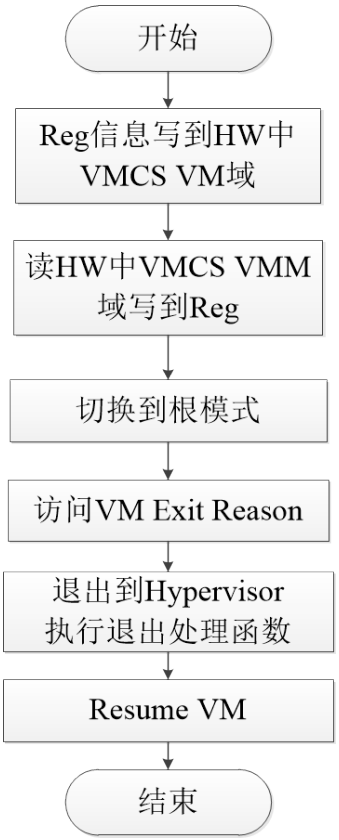

图 4.4 虚拟机退出重定向流程图.

Figure 4.4 VM Exit redirection.

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 5 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Zhu 等 (2017)，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McCune 等 (2010)，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Wang 等 (2012)，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。

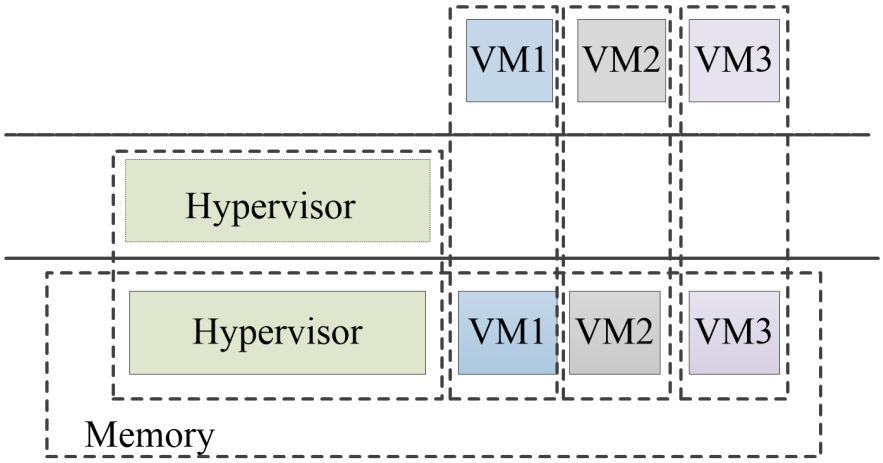

图 5.1 内存高强度隔离效果图.

Figure 5.1 Memory isolation.

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

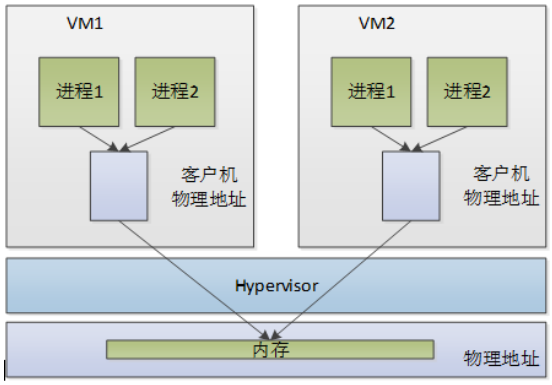

图 5.2 客户机虚拟地址翻译过程.

Figure 5.2 The virtual address translation of VM.

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。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。

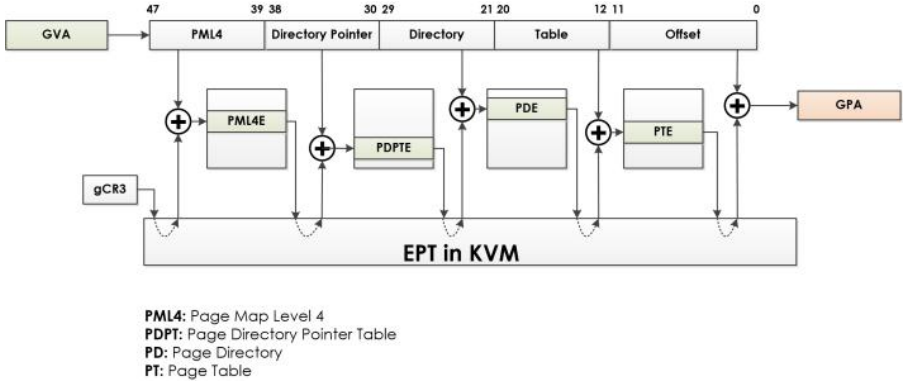

图 5.3 EPT 翻译原理图.

Figure 5.3 The EPT translation.

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），如图5.3分别是 EPT 地址翻译原理图。

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。

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

表 5.1 VM Mark 表.

Table 5.1 VM Mark Table.

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

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Payne 等 (2008) 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。

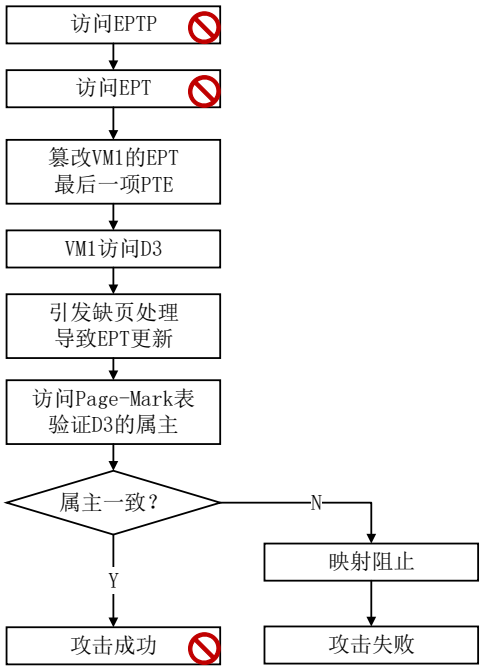

图 5.4 内存双映射攻击阻止说明图.

Figure 5.4 Blocking memory double mapping attack.

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（double mapping）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

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5.4描述了攻击路径，其中这些攻击路径会被多次阻止。

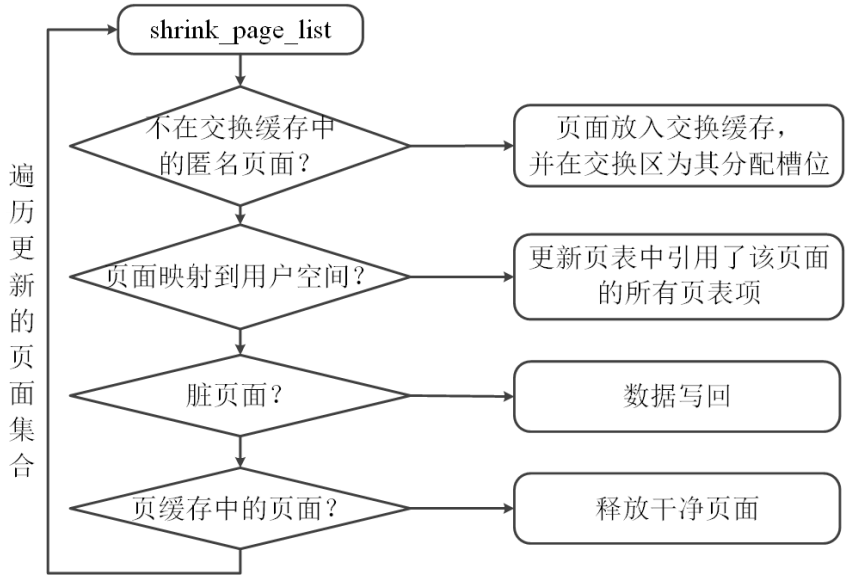

图 5.5 页面释放流程图.

Figure 5.5 Shrinking pages process.

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5.5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。

表 5.2 内存页面安全释放.

Table 5.2 Releasing memory pages safely.

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

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，这样就成功阻止了攻击。

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。

表 5.3 Page Mark 表.

Table 5.3 Page Mark Table.

标签	OwnerID	Used	SharedID
描述	属主标识	未使用或者使用	共享与否

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，SharedID 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 SharedID 标记的物理页忽略属主检查，非 SharedID 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 6 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

表 6.1 测试环境.

Table 6.1 Evaluation environment.

资源名称/类型	描述
宿主机 KVM	服务器，Linux4.4.1 系统，32G 内存，硬盘 1T，8 核，64 个逻辑 CPU，Intel(R) Xeon(R) CPU E7-4820 v2 @ 2.00GHz 处理器。
客户机	内存 8G，磁盘大小 30G，系统版本 ubuntu-12.04.5，5 台
SPEC CPU2006	面向处理器性能的基准程序集
Bonnie++	针对硬盘和文件系统的基准性能测试工具，主要测试数据读、写速度，每秒磁盘寻道次数和每秒文件元数据操作次数
Memtester	内存压力测试工具
Lmbench	基准测试工具，用于测试内存带宽、管道传输带宽、Cache 文件读带宽、进程创建反应时间、上下文切换反应时间、系统调用反应时间

测试环境包括软硬件配置环境和测试工具，概括如表6.1。

测试步骤：

1）使用测试工具测 50 次，将结果进行平均化。

2）使用 SPEC CPU2006 工具的编译运行测试代码，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。

4）使用 Bonnie++ 测试工具测 50 次，将结果进行平均化。在 test 目录下执行，针对内存大小为 1000M 的系统。执行该命令./Bonnie++ -d /test（目录路径）-s 2000（测试内存大小）-u root，在 terminal 得到数据信息并整理。

5）使用 Memtester 工具测试，测试命令 memtester <MEMORY> [测试次数]。通过使用 memtester 测试工具测试内存带宽，申请 1024M 内存进行内存压力测试 2 次。

6）使用 Lmbench 进行测试，在./src 目录下执行 make results 命令测试，make see 命令查看测试结果，并记录系统的关键指标数据。

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：参见表6.1。

测试工具：使用 RDSTC 指令编写代码测试启动/关闭耗时。

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。

6.1.3 测试结果分析

6.1.3.1 基准工具测试

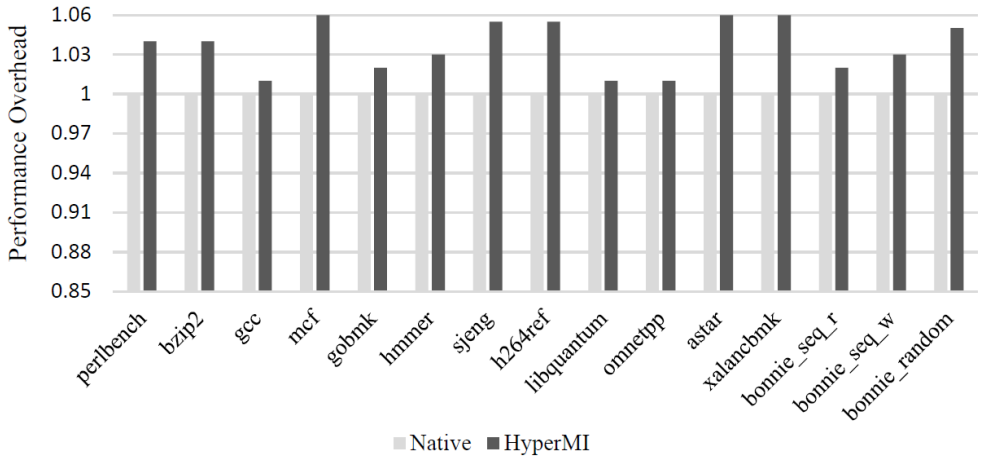

图 6.1 性能测试结果分析图.

Figure 6.1 Performance evaluation.

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。

表 6.2 Lmbench 系统指标测试.

Table 6.2 System indicator evaluation of Lmbench.

微观测试	*Native*	*HyperMI*	开销对比
protection fault	0.781	0.839	1.07
mmap	18.9	19.1	1.01
signal installation	0.23	0.25	1.09
signal delivery	1.61	1.71	1.16
fork+exit	204	224	1.1
fork+exec	754	763	1.01
fork+/bin/sh	1973	2108	1.07
ctxsw 2p/0k	7.24	7.91	1.09
ctxsw 8p/64k	12.6	13.1	1.04
ctxsw 16p/16k	9.4	9.94	1.06

通过使用 Memtester 测试工具在 5 台虚拟机中测试内存带宽，在每台申请 1024M 内存进行内存压力测试 2 次，测试结果表明在开启系统系统后，宿主机可以通过内存压力测试。主要原因是有关内存的更改主要是在虚拟机地址映射——EPT 更新中验证物理页的合法性，在物理页释放时清空其上的内存，这些操作对内存的申请和释放有部分延迟影响外，不会造成操作失败的影响。有关 Lmbench 工具做过两次测试，第一次是虚拟机没有负载运行时宿主机的性能开销 (Native)，第二次是虚拟机有负载运行时宿主机的性能开销 (HyperMI)，测试的结果如表6.2。HyperMI 所引发的性能开销在内存访问、系统调用、进程创建等小于原先系统的 1.2 倍，大部分小于 1.1 倍，可以接受。

6.1.3.2 虚拟机启动关闭时间测试

表 6.3 虚拟机启动关闭时间测试.

Table 6.3 The evaluation of VM startuping and shutdowning time.

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

测试结果如表6.3，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

将系统主要分为 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6.4。

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6.5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 6.6 测试环境.

Table 6.6 Evaluation environment.

资源名称/类型	配置
宿主机 KVM	服务器，Linux4.4.1 系统，32G 内存，硬盘 1T，8 核，64 个逻辑 CPU，Intel(R) Xeon(R) CPU E7-4820 v2 @ 2.00GHz 处理器
客户机	内存 8G，磁盘大小 30G，系统版本 ubuntu-12.04.5，2 台

表 6.4 篡改 VMCS 攻击步骤.

Table 6.4 Modifying VMCS attack.

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

表 6.5 篡改 EPT 攻击步骤.

Table 6.5 Modifying EPT attack.

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

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6.6。

测试方法

分别实现表6.7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6.8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6.9。

C. 安全执行环境的地址空间隔离测试：在 HW 中创建变量 A，跳转到 NW 中，在 NW 中访问变量 A，查看是否访问成功或者页表的页表项 PTE 是否为 NULL。

表 6.7 测试方法.

Table 6.7 Evaluation method.

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

表 6.8 测试步骤.

Table 6.8 All attack test process.

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’

D. 虚拟机内存隔离测试——内存标记与跟踪：在内存页被分配之后，打印其 Page Mark 表中的标签，成功打印表明实现了标记与跟踪功能。

E. 虚拟机内存隔离测试——内存页安全释放：在内存页释放时，打印一个 unsigned long 型的变量 B（虚拟地址 = 物理页对应的虚拟地址）的值，内容为零，则在内存页释放时清空了内存页的内容，成功阻止重映射攻击。

F. 虚拟机内存隔离测试——共享页接口设定：开启虚拟机后，拦截 KSM 机制合并页操作，拦截 Balloon 机制扩充页操作，打印共享的页的 Page Mark 标签，成功打印表明共享页接口完全实现了。

```
[804987.918046] -------Create secure environment and prevent malicious data access !----------
[804987.918046]
[804987.918047] The control flow has switched to secure address space!
[804987.918054] The control flow has switched to normal address space (another address space)!
[804987.918055] We are find the pte of this variable ......
[804987.918055] ......
[804987.918055]
[804987.918057] One of the pte of address mapping in page table for temp_data is NULL!
[804987.918057]    We protected critical data structures in secure address space successfully!
```

图 6.2 地址空间隔离测试结果图.

Figure 6.2 The evaluation of address isolation function.

6.2.3 测试结果分析

表 6.9 攻击案例.

Table 6.9 Attack examples.

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

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。

安全执行环境创建了另一个隔离的安全的地址空间 HW，在该地址空间中的数据和代码是无法被 NW 中的代码访问，测试结果如图6.2。结果表明所有的在 HW 地址空间中创建的数据和代码的虚拟地址映射在 HyperMI 页表上，并不在 Hypervisor 的页表上，地址映射不成功，所有在 NW 中访问 HW 中的数据和

代码的操作是不可行的，成功地实现了地址空间隔离。

```
[804567.827437] ----------Allocate PageFlag for pages and track pages !----------
[804567.827445] ----------There is 6 examples (PageFlag) to be shown !----------
[804567.827445]
[804567.827448] 1    page pfn                        : 2586b6 !
[804567.827449] 1    page owner(address of VM)   : ffff880029c6c000 !
[804567.827450] 1    page used                       : 1 !
[804567.827451] 1    page shared                     : 0 !
```

图 6.3 内存页分配与跟踪测试结果图.

Figure 6.3 The pages allocated and tracked.

```
[804602.890518] ----------Releasing page and release PageFlag for pages !----------
[804602.890519] ----------There is 6 examples to be shown !----------
[804602.890519]
[804602.890520] The page which pfn is 43696 is released! The corrupting PageFlag is deleted!
[804602.890523] The page which pfn is 3a768 is released! The corrupting PageFlag is deleted!
[804602.890525] The page which pfn is 550f0 is released! The corrupting PageFlag is deleted!
[804602.890528] The page which pfn is 72f59 is released! The corrupting PageFlag is deleted!
[804602.890530] The page which pfn is 29c59 is released! The corrupting PageFlag is deleted!
[804602.890532] The page which pfn is 6141 is released! The corrupting PageFlag is deleted!
```

图 6.4 内存页安全释放测试结果图.

Figure 6.4 The pages released safely.

```
[804674.751883] Track memory dynamically and change PageFlag for merged memory !----------
[804674.751892] ----------There is 6 examples (PageFlag) to be shown !----------
[804674.751892]
[804674.751894] 1    page pfn 271d65 is changed to be 813402 !
[804674.751896] 1    page owner ffff880851d04000 changed to be ffff880851d04000 !
[804674.751897] 1    page used 1 changed to be 1 !
[804674.751898] 1    page shared 0 changed to be 1 !
[804675.752601] 2    page pfn 3c219a is changed to be 813402 !
[804675.752609] 2    page owner ffff880851d04000 changed to be ffff880851d04000 !
[804675.752611] 2    page used 1 changed to be 1 !
```

图 6.5 共享页接口设定测试结果图.

Figure 6.5 The pages shared.

VM 内存高强度隔离功能除地址映射监控外，还包括内存动态标记与跟踪功能，通过实验对内存页分配与跟踪、内存安全释放、共享页接口设定功能进行测试。内存页分配与跟踪测试结果参见图6.3，通过拦截内存页更新，打印每一内存页的 Page Mark 标签信息，pfn 标记了所有页的页框号，可以唯一代表内存页，Used 标记页的使用情况，OwnerID 标记了属主信息（虚拟机或者 Hypervisor），SharedID 标记是否是共享页，内存页被成功标记与跟踪，为后续 VM 内存高强度隔离提供基础。内存安全释放测试结果如图6.4，通过拦截页释放函数，在页释放时将整个页的内容设置为 0，结果表明其内容被彻底清空，不会造成信息泄露的威胁。共享页接口设定参见图6.5，在内存复用机制开启后，通过拦截页“迁移”操作，及时变更页的 owner 信息，变更 used 信息，变更 shared 信息，实验过程中开启了 KSM 机制和 Balloon 机制，这两种机制“迁移”页操作都可以被检测到，系统可以在内存复用情况下正常运行。

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测试工具进行整体性能测试，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和系统安全功能充分测试并验证了 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Chhabra 等 (2011); Evtyushkin 等 (2015); Champagne 和 Lee (2010); Steinberg 和 Kauer (2010)。

第 7 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。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。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。

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

佚名. 系统虚拟化: 原理与实现[M]. 2009.

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. 2015.

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. 2016.

AZAB A M, PENG N, ZHI W, et al. Hypersentry:enabling stealthy in-context measurement of hypervisor integrity[C]//2010.

BAHRAM S, JIANG X, ZHI W, et al. Dksm: Subverting virtual machine introspection for fun and profit[C]//IEEE Symposium on Reliable Distributed Systems. 2010.

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.

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

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

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. 2012.

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

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. 2016: 565-578.

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.

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.

CVE. Survey on cve[Z]. 2018.

CVE-2017-8106. Survey on cve-2017-8106[Z]. 2017.

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.

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.

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. 2015: 190-202.

GARFINKEL T. A virtual machine introspection based architecture for intrusion detection[J]. Proc.network & Distributed Systems Security Symp, 2003:191-206.

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. 2003: 193-206.

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. 2013.

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.

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. 2007: 128-138.

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.

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.

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.

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

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. 2013: 511-526.

MCCUNE J M, LI Y, QU N, et al. Trustvisor: Efficient tcb reduction and attestation[J]. Cylab, 2010, 41(3):143-158.

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. 2013: 1-1.

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

PAYNE B D, CARBONE M, SHARIF M, et al. Lares: An architecture for secure active monitoring using virtualization[J]. 2008:233-247.

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

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. 2009: 74-81.

SALTZER J H. Protection and the control of information sharing in multics[C]//Acm Symposium on Operating System Principles. 1973: 119.

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.

SHARIF M I, LEE W, CUI W, et al. Secure in-vm monitoring using hardware virtualization[C]// Acm Conference on Computer and Communications Security. 2009.

SHI L, WU Y, XIA Y, et al. Deconstructing xen[C]//Network and Distributed System Security Symposium. 2017.

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. 2010: 209-222.

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.

WANG J, STAVROU A, GHOSH A. Hypercheck: a hardware-assisted integrity monitor[C]// International Conference on Recent Advances in Intrusion Detection. 2010: 158-177.

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).

WANG X, CHEN Y, WANG Z, et al. Secpod: a framework for virtualization-based security systems [C]//Usenix Conference on Usenix Technical Conference. 2015b: 347-360.

WANG Z, JIANG X. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity[C]//Security and Privacy. 2010: 380-395.

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. 2012: 127-140.

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. 2011: 203-216.

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. 2017: 242-256.

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。

作者简历及攻读学位期间发表的学术论文与研究成果

作者简历

姓名：刘文清　　籍贯：山西大同培养层次：学硕

2016.9 – 2019.6 在中国科学院信息工程研究所获得硕士学位。

2012.9 – 2016.7　在同济大学点子与信息工程学院（系）获得学士学位。

已发表 (或正式接受) 的学术论文:

[1] HyperMI: A Privilege-level VM Protection Approach against Compromised Hypervisor(TrustCom2019)（已接收）

申请或已获得的专利:

实现虚拟机安全隔离的方法与装置（已受理）

参加的研究项目及获奖情况:

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

2018 年中国科学院大学“三好学生”

2017 年第十四届“华为杯”全国研究生数学建模竞赛三等奖