附件 3

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

研究生学位论文开题报告

报告题目 虚拟机隔离与监控技术的研究

学生姓名 刘文清 学号 **201628018629130**

指导教师 涂碧波 职称 博导

学位类别 工学硕士

学科专业 网络空间安全

研究方向 虚拟化安全

研究所（院系） 信息工程研究所

填表日期 **2018.6.8**

中国科学院大学制

1、 选题的背景及意义

1.1 背景

虚拟化（Virtualization）是一种资源管理技术，是将计算机的各种实体资源，如服务器、网络、内存及存储等，予以抽象、转换后呈现出来，打破实体结构间的不可切割的障碍，使用户可以比原本的组态更好的方式来应用这些资源。这些资源的新虚拟部份是不受现有资源的架设方式，地域或物理组态所限制。

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 33600 行[2]，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-2018-1087[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。

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。然而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。

2017 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。2017 年，Uber 主动公开了去年曾向黑客支付 10 万美元封口费以隐瞒 5700 万账户数据泄露事件。两名黑客通过外部代码托管网站 GitHub 获得了 Uber 工程师在 AWS 上的账号和密码，从而盗取了 5000 万乘客的姓名、电子邮件和电话号码，以及约 60 万名美国司机的姓名和驾照号码。Uber 同样还记录了 “有关用户运动和旅行历史的详细数据”，这也就意味着，“黑客可以根据这些数据追找到用户的位置，甚至是家庭住址。2016 年，雅虎公司揭示了自己公司数据泄露的情况，泄露的数据量达到了 10 亿。同时谷歌公司 iCloud 平台泄露了用户数据。

虚拟化技术已经产生了巨大的影响，并在云计算中扮演着至关重要的角色。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。所以从 Hypervisor 的角度对虚拟机进行保护是至关重要的。最终的目标是保护虚拟机不受恶意的 Hypervisor 攻击，从保护 Hypervisor，对 Hypervisor 与 VM 的交互进行防护，保障 VM 的资源安全隔离这 3 个部分来对 VM 进行保护。

1.2 意义

为了保护运行在云平台上的用户的数据，从对 VM 进行保护的角度进行考虑，我们通过对 Hypervisor 进行监控并且对 VM 的内存资源进行隔离，当然这些都是运行在一个相对安全的隔离区域，否则监控和资源隔离会受到很大的影响，并且没有意义。可以更好地保护 VM，隔离其与恶意 hypervisor，更好地保护运行在云平台上用户的数据。

2、 国内外本学科领域的发展现状与趋势

现在有许多虚拟化方面的保护虚拟机安全的方法[4][5][6][7][8][9][10][11][12][13][14][15][16]。

2.1 安全的运行时隔离空间

硬件辅助就是利用现有硬件的安全特性提供一个隔离的受保护的执行环境，然后此隔离的执行环境中提供安全监控、验证、检查等机制。能够保证代码的完整性和数据的机密性。各平台上当前的主要硬件设备是 x86 平台上的 SGX[17]（软件保护扩展），ARM 平台上的 TrsutZone。

2.1.1 硬件

SGX

SGX 全称 Intel Software Guard Extensions，顾名思义，其是对因特尔体系（IA）的一个扩展，用于增强软件的安全性。这种方式并不是识别和隔离平台上的所有恶意软件，而是将合法软件的安全操作封装在一个 enclave 中，保护其不受恶意软件的攻击，特权或者非特权的软件都无法访问 enclave，也就是说，一旦软件和数据位于 enclave 中，即便操作系统或者和 VMM（Hypervisor）也无法影响 enclave 里面的代码和数据。Enclave 的安全边界只包含 CPU 和它自身。SGX 创建的 enclave 也可以理解为一个可信执行环境 TEE（Trusted Execution Environment）。不过其与 ARM TrustZone（TZ）还是有一点小区别的，TZ 中通过 CPU 划分为两个隔离环境（安全世界和正常世界），两者之间通过 SMC 指令通信；而 SGX 中一个 CPU 可以运行多个安全 enclaves，并发执行亦可。当然，在 TZ 的安全世界内部实现多个相互隔离的安全服务亦可达到同样的效果。

（1）允许应用开发者保护敏感信息不被运行在更高特权等级下的欺诈软件非法访问和修改。

（2）能够使应用可以保护敏感代码和数据的机密性和完整性并不会被正常的系统软件对平台资源进行管理和控制的功能所扰乱。

（3）使消费者的计算设备保持对其平台的控制并自由选择下载或不下载他们选择的应用程序和服务。

SGX 同样存在它的弊端：1)SGX 在应用程序层而给管理程序层提供细粒度的保护。2) SGX 还没有用于服务器芯片和商品产品，相反，SGX 应该用于云中的数据中心和应用。SGX 只能在客户操作系统上保护软件的相对较小部分的内存，开发人员必须花费大量时间重新构建受保护的软件或重新开始构建。它的主要应用还是在应用层。3)这需要开发人员花时间重构代码，并将代码分为受信任部分和不受信任部分。这很复杂，也很耗时。因此，SGX 保护整个虚拟机或操作系统等大型软件是不常见的。

TrustZone

TrustZone(TM) 技术出现在 ARMv6KZ 以及较晚期的应用核心架构中。它提供了一种低成本的方案，针对系统单芯片（SoC）内加入专属的安全核心，由硬件建构的存取控制方式支援两颗虚拟的处理器。这个方式可使得应用程式核心能够在两个状态之间切换（通常改称为领域（worlds）以避免和其他功能领域的名称混淆），在此架构下可以避免资讯从较可信的核心领域泄漏至较不安全的领域。这种内核领域之间的切换通常是与处理器其他功能完全无关联性（orthogonal），因此各个领域可以各自独立运作但却仍能使用同一内核。提供用户层安全接口，在敏 感数据处理时能够切换到安 全环境，用于敏感数据的防 篡改、防泄漏使用 TrustZone 技术隔离为 上层软件提供隔离、可认证的执行环境支持虚拟化和多虚拟机技术，管理程序和系统 MMU。但是 TrustZone 是基于 ARM 系统的。

2.1.2 软件

在软件方法层面，有各种方法被提出提供安全的隔离空间，用来运行和管理正在运行的程序或者用户的数据。从提供更高管理层和微小型的管理层（减小可信 TCB 大小）两个方面进行介绍。第一种，因为 Hypervisor 在系统中拥有最高的权限，当其被攻击后会对 VM 的安全性产生更大的影响，从而有学者提出提供更高特权级别的管理层，即使用内嵌虚拟化技术创建一个更能高层的虚拟机监控器。第二种，通过创建微小型的虚拟机监控器，减小原来虚拟机监控器的代码量的大小，保证 TCB 减小，从而减小其攻击面。

- 内嵌虚拟化是在一个 hypervisor 上运行另一个 hypervisor，将原来的监控器的控制权的下降，限制了其对虚拟机资源的访问。Cloudvisor[18] 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。Tinycheeker 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 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。嵌套虚拟化有一个很大的不足：系统运行时需要在两个系统之间进行切换，切换过程的性能开销相对比较大，主要是特权寄存器的访问和特权级别的切换。嵌套虚拟化本身会使得代码更复杂。
- 微内核，NOVA[19]借鉴微内核思想，提出一种针对 x86 架构的轻量级虚拟化框架。该框架主要基于一个能被独立设计、开发和验证的可信模块。在 NOVA 中，传统的 Hypervisor

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

综上，嵌套虚拟化会带来更多的性能开销。微型 Hypervisor 会使得原先虚拟机监控器的功能减小，兼容性差。

2.2 保护 Hypervisor 的完整性

由于 Hypervisor 位于虚拟化的最底层，负责虚拟化的整个软件栈安全，同时也是攻击者面临的最后一道软件层防线。然而减少攻击面并不能完全保证虚拟化层的安全。因此，需要提供安全机制来保护 Hypervisor 的安全。可信启动能够保证 Hypervisor 启动的安全性，但是 Hypervisor 运行时的威胁却如梦魇般相随。保证 Hypervisor 运行时的安全最有效、最常用的方法就是保证 Hypervisor 运行时的完整。

HyperCheck[20]和 HyperSentry 采用完整性验证的方式，实时地保证 Hypervisor 不被篡改，防范 Rootkits 攻击等。HyperCheck 提供了一种基于硬件辅助的 Hypervisor 完整性探测框架，利用 SMM 对系统内存和寄存器制作快照，然后通过 NIC 将内存数据发送给远程的分析服务器进行分析（物理、内存获取模块）。为了便于获取物理内存的数据，HyperCheck 提出借助符号表和寄存器值的方案，防止因拷贝－变化攻击而获取错误数据（寄存器验证模块），例如攻击者不改变原有 Hypervisor 的页表，而是通过修改 CR3 等寄存器伪造一个新的页表。远端分析服务器通过对内存快照分析而验证 Hypervisor 的完整性，同时利用带外机制监测对 NIC 设备的 DOS 攻击（分析模块）。在上述方案中，完整性验证操作需要由 Hypervisor 自身触发，这期间可能存在擦洗攻击，即在完整性验证之前攻击者已擦除其攻击痕迹.

Hypersafe[21]提出了两个关键技术：不可被绕过的内存锁定机制和受限的指针索引。首先是通过不可绕过的内存锁定机制保护 Hypervisor 的代码和静态数据不能被篡改。由于内存锁定机制并不能灵活地适用于动态数据的保护，Hypersafe 提出数据重定向机制，其目的

是将动态数据转换成指针索引.而这些索引值是通过线下执行 Hypervisor 而得到的执行流图。这样就把执行流的可能性限制在了可信集之内。不可绕过的内存锁和受限的指针索引保证了 Hypervisor 控制流的完整性。但文章中只是说明了在对页表进行更改时需要验证操作行为是否是合法的，并没有提出判断操作合法性的依据和标准。完整性验证只能保证攻击

Hypervisor 后能被及时地觉察，但并不能防范于未然。Hypersafe 克服了 HyperCheck 和 HyperSentry 的不足，能够防护 Return-oriented 类型的攻击。但 Hypersafe 的自保护机制需要进行代码剔除，其灵活性不高。

2.3 虚拟机隔离

租户模式和资源共享是云计算的主要威胁，而内存去重机制更加剧了这一威胁。客户虚拟机之间的隔离性是由 Hypervisor 维护的，而在不可信的虚拟化环境下同一物理机内虚拟机之间的隔离性难以保证，这可能导致客户虚拟机的数据泄漏、跨域访问、控制流截获等威胁。

底层资源分配隔离

将不同虚拟机以及虚拟机与 Hypervisor 之间隔离，在分配页面时通过查询页面属主表保证一个页面只能属于一个虚拟机。除此之外，Cloudvisor 限制 Hypervisor 对虚拟机 EPT 表的更新，防止恶意 Hypervisor 违反页属主唯一的原则。通过这些机制，Hypervisor 能够保护虚拟机的数据不被其它虚拟机窃取。然而在虚拟化中，虚拟机需要频繁地与 Hypervisor 进行交互，这使得 Cloudvisor 在切换过程中性能损耗较大。并且由于 Cloudvisor 引入了新的软件层，在一定程度上增加了虚拟化软件栈的复杂度和潜在安全漏洞。硬件的隔离与 Cloudvisor 类似，SMMU 和 H-SVM 通过对 CPU 进行逻辑扩展实现虚拟机间的隔离。该方案的隔离性并不基于软件层，可信基只包含内存、CPU 和 Cache 等硬件资源。H-SVM 方案借鉴了 Cloudvisor 的内存属主机制，并且基于 CPU 的 SMM 运行模式利用微码实现了虚拟机内存页的隔离。H-SVM 首先扩展了内存分配和释放指令，在 Hypervisor 进行内存管理时，需要调用 SMM 中的微码，只有通过对物理页属主的审查后才能进行相应内存分配。H-SVM 借鉴了 Cloudvisor 对内存页属主的跟踪机制，利用 OPT 表记录每个物理页的属主。在 Hypervisor 调用 map 分配内存时，H-SVM 首先根据安全内存中的 OPT 表判断 Hypervisor 分配的物理页是否已有属主。如果这个物理页是空闲的，则分配给这个虚拟机，并标记其属主为此虚拟机的 ID。这样保证了每个物理页只能分配给一个虚拟机。但是对于虚拟机之间和虚拟机与 Hypervisor 之间的共享内存，H-SVM 提供了一个接口可供客户虚拟机自己设置共享页。在 H-SVM 的后期实现中，作者利用 SMM 的隔离性和高特权级进一步对 H-SVM 进行了扩展，将安全模块与 Hypervisor 进行了完全隔离。但 SMM 的缺点也是显而易见的。H-SVM 需要 Hypervisor 配合来完成其操作，这对 Hypervisor 不透明，并且也没有像

HyperSentry 提供隐秘触发机制。此外，H-SVM 只能防范软件层攻击，对于物理攻击无能为力。NoHype[22]也利用了隔离的思想，将用户的操作、数据等限制在一个独立的环境中。但不同的是，NoHype 是对物理资源进行绝对隔离，做到了资源专用。NoHype 不仅可以防止跨域内存访问和 Cache 共享，还可以避免客户虚拟机间的相互影响。

基于硬件的虚拟机隔离

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

Iso-X 和 Intel SGX 通过对 CPU 进行安全扩展将内存的保护粒度缩小到了隔间，其保护对象不再是客户虚拟机，而是将保护对象缩小到进程内部模块。这两种方案都是 CPU 访存时对操作进行判断，而指令级的访存非常频繁，带来的性能损耗较大。再者，这两种方案需要从编程语言、编译器以及 CPU 硬件整个架构进行全套更新，适用性比较差。

3、 课题主要研究内容、预期目标

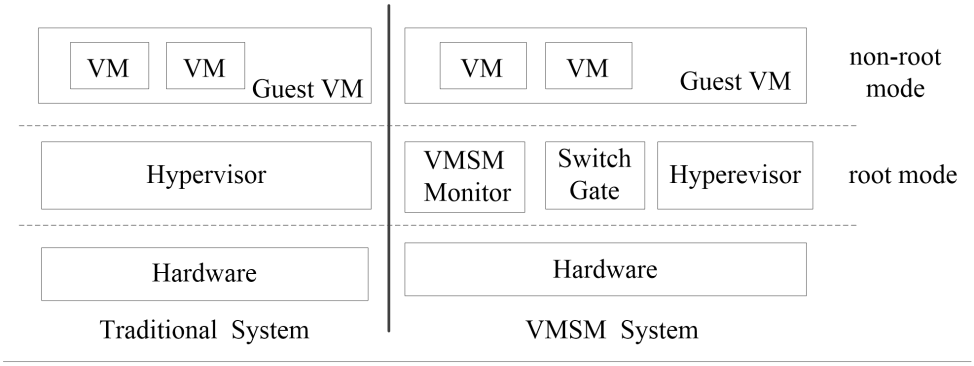


图 1 虚拟机隔离监控架构

主要研究内容：在安全执行环境运行的虚拟机隔离监控技术，VMSM 系统，如图 1。

在传统的 HyperVisor 系统上添加系统组件，这些系统组件具备其余的功能特性，能够使得整个系统能够更好地运作起来，例如，虚拟机隔离以及对应的监控防护功能等。

当然，当这些系统组件运行的环境并没有足够的安全保障时，其对应的功能也很难发挥它的作用，系统安全很难得到相应的保障，那么一个安全的可执行环境是系统必不可少的。当系统组件运行在安全的环境中，并且该安全环境能够达到一定的要求，能够防御一定的外

部攻击，同时系统组件能够发挥其功能，更全面地服务于整个系统。

该隔离空间有一定的灵活性和安全性，不会给系统运行带来很大的性能开销，减少大量的特权级别的切换。目的是实现一个安全的可执行环境，在该安全环境中可以运行一些系统组件，进行监控防护、虚拟机隔离等操作，与外部的内核攻击进行隔离。

同层隔离的安全执行空间，创建一个安全的执行环境，执行环境所在的安全空间是和 hypervisor 同层的。目的是减小性能开销。为了使得系统得到安全的防护，提出了内存安全隔离和虚拟机安全监控的方案，该方案是基于 x86。内存空间隔离是轻量级的，与 hypervisor 同层，能够减小性能开销，基于软件的解决方案。同时对安全隔离空间进行安全保护，主要通过监控对安全隔离空间的操作函数实现，监控操作是基于事件驱动的，非硬件，非轮询的，将监控的相关函数运行在隔离出来的安全空间中，可以实现不可绕过的监控，还能持续向该安全空间中存放其它系统相关组件。

虚拟机监控技术是主要监控虚拟机与 hypervisor 之间的交互过程。VM 与 hypervisor 的交互过程很频繁，其中 80%是 I/O 和缺页访问，当然，一个物理核只能在一个时间点上处理一个系统，hypervisor 和 VM 之间分时地在使用物理核。所以在物理核运行的系统需要进行切换，在这个切换的过程中，相关的状态信息会被记录到一个特殊的硬件结构体上，其中包含了有关 HOST OS 和 GUEST OS 的特权寄存器信息和其它重要的控制域信息。在上下文切换的过程中，可能受到信息篡改，控制流攻击等。这些信息不可以被篡改，篡改之后会导致加载恶意的页表、EPT 表、改变 CR0 等寄存器中的内容，可能导致系统中的保护机制被关闭，导致控制流攻击。

对上下文数据结构的监控需要找到一个确切的时间点，上下文切换的原因是虚拟机退出，所以需要监控到虚拟机退出事件，才能更容易捕捉到上下文切换的过程。虚拟机退出的一般流程是首先对宿主机和客户机的状态进行保存，然后分析虚拟机退出的原因，对不同的事件进行不同的退出处理。

那么研究内容主要是捕捉虚拟机退出事件的发生时间点，在安全区域中监控上下文切换的过程，然后把虚拟机退出事件处理转交给 hypervisor 进行处理，即虚拟机重定向。

虚拟机隔离技术把虚拟机物理内存与 hypervisor 的物理内存隔离开，以及把虚拟机之间的物理内存隔离开，抵御同驻攻击。

同驻攻击一般是指共享同一物理资源的多个虚拟机，称为多租户，其上的虚拟机可以对共享资源进行任意的更改，访问不属于它的物理资源甚至污染物理资源。当其中一台虚拟机 VM1 被攻陷后，其它共享资源的虚拟机 VM2 会因为 VM1 对物理资源（主要是内存）进行了污染而被攻击，发生数据泄露攻击，所以会在很大程度上泄漏用户的数据。那么就有必要对虚拟机的资源进行高强度的隔离，物理资源中最重要的部分是物理内存资源。为了隔离物理资源，需要找一个确切的时间点，或者能够拦截所有物理内存分配的特殊事件，仔细分析一下，hypervisor 管理着所有的物理内存分配，而且这个是通过 EPT 更新的过程中分配所有的物理内存，所以可以在 EPT 更新的过程中标记物理内存，同时进行一些访问控制策略的实施，将物理内存标记分为 hypervisor 的或者其他 VM 的，将物理内存彻底地进行隔离。

综上所述，研究内容主要是监控 EPT 事件的更新，在此更新的过程中确保 EPT 的正确性，一个 VM 对应一个 EPT,防止加载恶意的 EPT，从而对虚拟机进行数据泄露攻击、控制

流攻击等。其次，要在 EPT 更新的过程中对各个 VM 和还有 hypervisor 的物理内存进行完整地隔离，阻止同驻攻击，数据泄露等攻击，加强运行在 VM 上的程序数据的保护。
综上所述，主要提出如下方案：

- 同层隔离的安全执行空间：同层隔离空间的创建，对该空间的安全保护机制。
- 虚拟机安全监控技术：虚拟机上下文安全切换，虚拟机退出重定向技术。
- 虚拟机安全隔离技术：EPT 更新监控，物理页标记和跟踪技术。

3.2 预期目标

实现针对 VM 的完全隔离，抵御如下攻击。

功能	描述
创建一个与 hypervisor 同层的安全隔离空间	创建一个相对普通环境的一个安全隔离空间，该空间是隔离与普通环境的，恶意的 hypervisor 是不可以直接访问的
不可绕过安全执行环境	提供安全的保护策略，对于这个安全的隔离环境，可以抵御一定的攻击。防止安全隔离环境被绕过及安全隔离空间被破坏。
普通环境和安全环境切换	安全隔离环境和普通环境可以按照规则进行切换，性能开销相对较低，切换的入口只有一个，负责安全切换环境。
虚拟机上下文安全切换	虚拟机上下文切换过程安全监控，防止控制流攻击，数据泄露，恶意篡改上下文。
虚拟机退出重定向	能够快速的检测到虚拟机退出事件，对上下文切换进行时间点记录，并转发虚拟机退出事件。
EPT 更新监控	监控 EPT 更新，拦截物理页分配过程，防止加载恶意的 EPT。采用特殊的方法保证 EPT 不被 Hypervisor 恶意访问。
物理页标记和跟踪技术	采用相对创新的方法对 VM 的物理页进行标记，该创新的实现方法能够减小性能开销。防御同驻攻击，防御恶意 hypervisor 对 VM 物理内存直接访问篡改等。

4、 拟采用的研究方法、技术路线、实验方案及其可行性分析

4.1 安全隔离空间

研究方法

通过 3 个部分进行安全隔离空间的创建。首先创建安全隔离区域，创建一个安全的切换

门（唯一性、原子性），当然要设置一些安全策略对安全隔离空间进行保护，避免受到攻击，从而绕过安全监控甚至破坏安全隔离空间的完整性。

技术路线

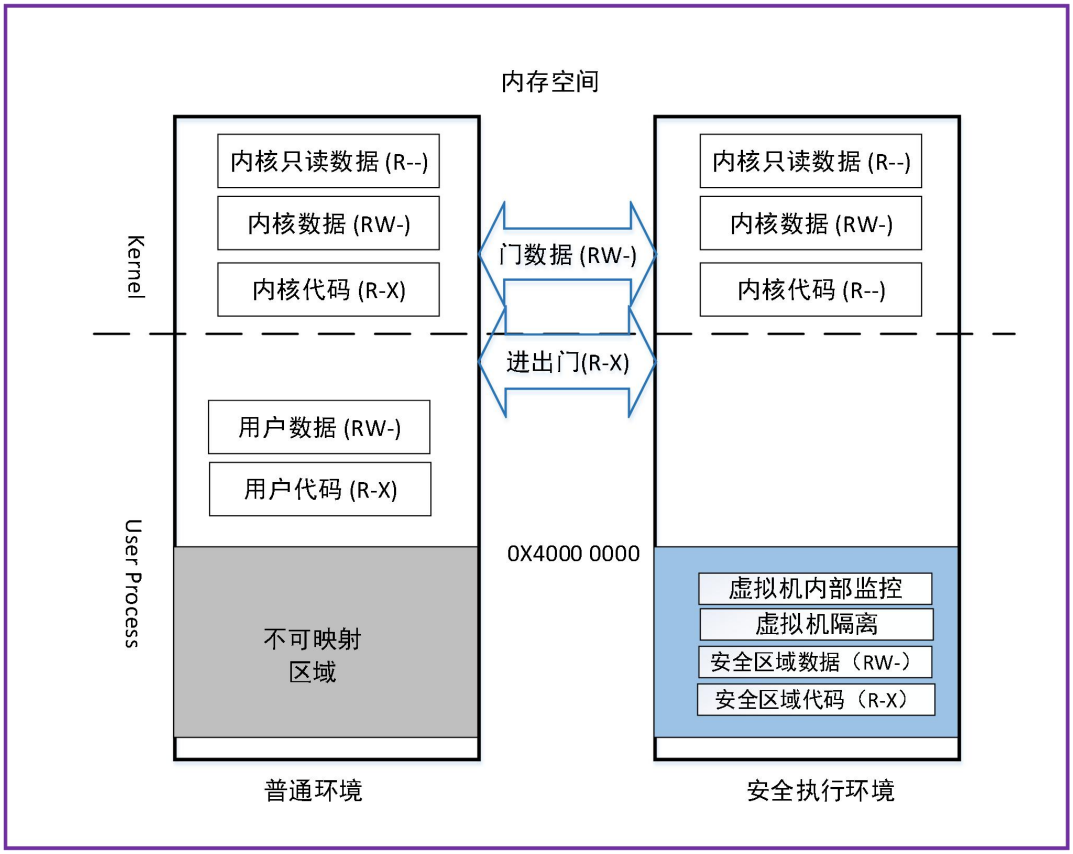
1 两份页表--创建隔离空间

一 份页表用于原系统正常的运行，并不包含安全空间的页表的地址以及运行在该空间中进程的相关地址。安全空间的页表基地址另一份页表存放安全空间的相关地址。

实现的技术：申请空间，创建页表，内容复制于原页表的内核空间部分，将该页表相关的地址在内核页表中删除，以防泄露安全空间的地址。

2 安全的切换门

该安全的切换门需要保证原子性，确切性，唯一性。唯一性的目的是要确保两个环境切换的出入口只有一个，否则攻击者可以从其他地方进入安全隔离空间，那么就失去了隔离的意义。技术关键点是在切换门的共同区域中包含出入口的地址，进入安全隔离区域的地址是需要放在一个相对安全的区域，不可以让攻击者通过访问 MMU 的方式轻易访问的到这个数据。原子性是指在切换的过程需要确定该切换过程是不可被随意中断的，否则会泄漏安全区域的地址。如下图：



3 安全保护机制

禁止对 MMU 的相关操作，为了防止被恶意攻击的内核使用其它的地址空间或者恶意页表，绕过安全执行空间（加载其余恶意页表），更改页表项的属性（可以随意更改代码段数据段），或者对页表进行恶意更改（写地址映射，导致随意访问其余空间），需要把有关 MMU 的操作的函数放到安全区域执行。涉及到 pte pmd pud pgd cr3 cr0 cr4 cr8 lgdt lldt 的写操作。监控对页表的写操作，一旦对页表进行写操作，则跳到安全区域中执行。当 hook 后，对 hook 的函数进行写禁止操作，防止原函数被 hook 到别的函数，使得原本的 hook 操作失效。

实验方案

隔离空间创建：内存中的一块连续地址作为安全执行环境区域，该地址可以在内核地址空间或者用户地址空间。需要通过修改内核使用的内存转换表来完成，以便所有的转换地址都不指向安全环境使用的物理内存区域。为了保护这个新的地址空间，所有的内存转换表都必须是这个新的地址空间的一部分，以便安全环境可以独占。因此，内核不能修改任何内存转换表来篡改虚拟内存访问权限。

切换门：唯一性的保证是通过预先创建一个空间，该空间是内核不可以通过 MMU 式进行访问的。预先将物理内存剥离出来供存放安全区域地址使用。

原子性的保证与实现是通过使用开关中断的方式。防止外部的攻击打断切换过程，多次关中断会更加安全。

安全保护机制：对于内核或安全环境进行访问权限设置。内核代码在普通和安全环境中的访问权限不同，防止在安全环境中恶意的内核代码执行非法程序。其余类似。主要保护 MMU 和特权寄存器 CR0 CR3 CR4。

可行性分析

功能	可行性分析
安全隔离空间创建	创建的安全的隔离空间，安全的工具就可以运行在该区域中，一般包括虚拟机监控和虚拟机隔离实现。
安全切换	切换的过程是叫将页表进行切换，达到了原子性、确定性。切换过程是唯一的。切换过程理论上是性能开销相对较低。
安全机制保护	从 2 个方面进行保护，一个是 MMU,另一个是特权寄存器，保证安全隔离空间不被篡改，保证不可绕过安全隔离空间。

4.2 虚拟机监控

研究方法

监控的主要内容是 hypervisor 和 VM 之间的交互过程。监控虚拟机上下文切换和虚拟机加退出重定向。

技术路线

1 上下文切换监控

在该过程中会使用到一个用来存放系统状态信息的数据结构体，VMCS。通过监控 VMCS 的相关操作，在该过程中验证操作的安全性，数据以及参数的合法性。防止恶意篡改该数据结构体。

2 虚拟机退出重定向

对 VMCS 操作的地方有很多，虚拟机退出是一个检查点，所以选择在虚拟退出的时候进行监控，拦截对 VMCS 的操作。

实验方案

1 上下文切换监控

为了不让恶意的 hypervisor 访问或者篡改 VMCS 结构体，最好的方法是将该数据结构体隐藏在安全区域中，对于相关的操作进行严格监控。

2 退出重定向

首先对退出事件进行拦截，随后对 VMCS 的读写以及其他访问操作进行监控，之后将退出的处理转交给 hypervisor 进行处理。

可行性分析

功能	可行性分析
上下文切换监控	保证 VMCS 数据结构的完整性，防止控制流攻击，加载页表、EPT 表
退出重定向	准确地拦截对 VMCS 数据结构的监控

4.3 虚拟机隔离

研究方法

1 内存隔离

将物理内存分为 hypervisor 和各个 VM 的。

2 EPT 更新监控

监控 EPT 更新，以及 EPT 的其他相关操作，来保证 EPT 的完整性。

技术路线

1 内存隔离

通过使用物理页标记的方法对物理内存进行标记。

2 EPT 更新监控

将 EPT 的相关信息隐藏，保证 EPT 的安全性，同时在 EPT 更新的过程中，分配物理内存页，并且对分配过程进行监控，验证合法性。

实验方案

1 内存隔离

通过使用物理页标记的方法对物理内存进行标记。

2 EPT 更新监控

将 EPT 的相关信息隐藏，使得 hypervisor 操作 EPT 时，需要请求安全模块进行处理。

可行性分析

功能	可行性分析
内存隔离	使得 VM 之间的隔离性达到很强的高度。抵制同驻攻击。
EPT 更新监控	保证 EPT 的完整性，同时完成内存隔离，验证内存分配过程的合法性。防御加载恶意的 EPT，防御数据泄露。

5、 已有研究基础与所需的研究条件

已有研究基础：

已经通过大量的调研工作，获取了关于内存虚拟化技术和同层级别监控的资料，虚拟机监控和虚拟机隔离的资料。

所需研究条件：

合适的硬件设备，软件安装包等。

6、 研究工作计划与进度安排

工作计划	进度安排
基础知识加深了解	2018.6-2018.7
同层安全隔离空间的创建实现及相关文档的书写	2018.7-2018.8
虚拟机监控技术的实现及相关文档的书写	2018.8-2018.10
虚拟机隔离技术的实现及相关文档的书写	2018.10-2018.12

论文撰写	2019.1-2019.4

参考文献

[1] L. Deng, P. Liu, J. Xu, P. Chen, and Q. Zeng, Dancing with wolves: Towards practical event-driven vmm monitoring, in Proceedings of the 13th ACM.
SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 83–96, ACM, 2017.

[2] Common Vulnerabilities and Exposures, https://cve.mitre.org.

[3] Common Vulnerabilities and Exposures,
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1087

[4] Monirul Sharif, Wenke Lee, Weidong Cui, and Andrea Lanzi. Secure In-VM Monitoring Using Hardware Virtualization. In Proceedings of the 16th ACM Conference

[5] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. KVM: the Linux Virtual Machine Monitor. In Proceedings of the 2007 Ottawa Linux Symposium, June 2007.

[6] Mccune J M, Li Y, Qu N, et al. TrustVisor: Efficient TCB Reduction and Attestation. In Proceedings of S&P, pages 143-158, 2010.

[7] Jin S, Ahn J, Cha S, et al. Architectural Support for Secure Virtualization under a Vulnerable Hypervisor. In Proceedings of MICRO, pages 272-283, 2011.

[8] Szefer J, Lee R B. Architectural Support for Hypervisor-Secure Virtualization. In Proceedings of ASPLOS, pages437-450, 2012.

[9] A. M. Azab, P. Ning, E. C. Sezer, and X. Zhang, “HIMA: A hypervisorbased integrity

measurement agent,” in Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC ’09), 2009, pp.193–206.

[10] J. Criswell, A. Lenharth, D. Dhurjati, and V. Adve, “Secure virtual architecture: a safe execution environment for commodity operating systems,” in Proceedings of the 21st ACM SIGOPS symposium on Operating systems principles (SOSP ’07), 2007, pp. 351–366.

[11] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, “Terra: a virtual machine-based platform for trusted computing,” in Proceedings of the 19th ACM symposium on Operating systems principles (SOSP’03), 2003, pp. 193–206.

[12] A. Seshadri, M. Luk, N. Qu, and A. Perrig, “SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes,” in Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (SOSP ’07), 2007, pp. 335–350.

[13] J. Rhee, R. Riley, D. Xu, and X. Jiang, “Defeating dynamic data kernel rootkit attacks via VMM-based guest-transparent monitoring,” in Proceedings of the International Conference on Availability, Reliability and Security (ARES ’09), 2009, pp. 74–81.

[14] N. L. Petroni Jr. and M. Hicks, “Automated detection of persistent kernel control-flow attacks,” in Proceedings of the 14th ACM conference on Computer and communications security (CCS ’07), 2007, pp. 103–115.

[15] B. D. Payne, M. Carbone, M. Sharif, and W. Lee, “Lares: An architecture for secure active monitoring using virtualization,” in Proceedings of the 29th IEEE Symposium on Security and Privacy, 2008, pp. 233–247.

[16] L. Litty, H. A. Lagar-Cavilla, and D. Lie, “Hypervisor support for identifying covertly executing binaries,” in Proceedings of the 17th USENIX Security Symposium, 2008, pp. 243–258.

[17] Anati, S. Gueron, S. Johnson, and V. Scarlata, “Innovative technology for cpu based attestation and sealing,” in Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy, vol. 13, 2013.

[18] Zhang F, Chen J, Chen H, et al. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization. In Proceedings of SOSP, pages 203-216, 2011.

[19] Steinberg, U. and B. Kauer. NOVA: A Microhypervisor-Based Secure Virtualization Architecture. In Proceedings of EUROSYS, pages 209-222, 2010.

[20] Wang J, Stavrou A, Ghosh A. HyperCheck: A Hardware-Assisted Integrity Monitor. In Proceedings of RAID, pages 158-177, 2010.

[21] Wang Z, Jiang X. Hypersafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity. In Proceedings of S&P, pages 380-395, 2010.

[22] Keller E, Szefer J, Rexford J, et al. NoHype: Virtualized Cloud Infrastructure without the Virtualization. In Proceedings of ISCA, pages 350-361, 2010.