**\_\_\_\_虚拟机安全套件\_\_\_\_\_用户使用手册**

作者：\_\_\_刘文清 \_\_\_\_\_

参与者：\_\_\_\_张坤\_\_\_\_\_\_

版本信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本/状态** | **作者** | **参与者** | **编制/修订历史时间** | **备注(修改单号)** |
| V1.0 | 刘文清 |  | 2019.1.2 |  |
| V2.0 | 刘文清 | 张坤 | 2019.4.2 | 概述、系统使用 |
| V3.0 | 刘文清 | 张坤 | 2019.6.1 | 概述 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**目录**

[1 概述 4](#_Toc13598)

[2 使用须知 4](#_Toc28942)

[3 系统使用 5](#_Toc17576)

[3.1 系统安装 5](#_Toc30154)

[3.2 系统加载/卸载 7](#_Toc22531)

[4 功能说明 7](#_Toc9253)

## 概述

产品的功能和性能简单说明、运行环境说明、使用权限说明等，如必要，可给出主界面图示。

* 虚拟机安全套件概述

当前，云计算平台在互联网中越来越重要，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，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，可能会危及整个云计算基础设施，并危及云中的任何数据。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。

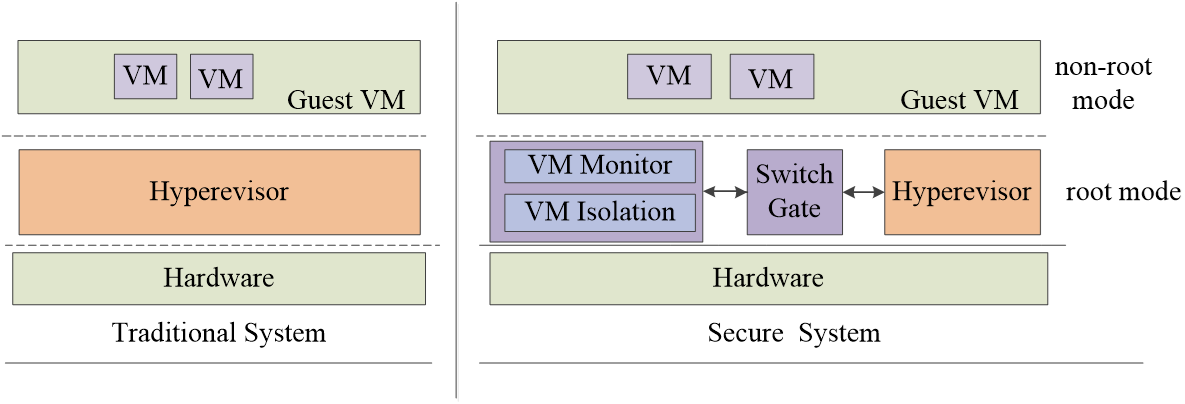


图1. 系统总体架构图

虚拟机安全套件，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，系统总体架构图如图1。首先，通过创建另一份内核页表从而创建一个安全隔离的可信执行环境，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，该地址空间是通过Switch Gate与原先的Hypervisor进行交互和地址空间切换。其次，创建了事件驱动的虚拟机上下文安全切换监控（VM Monitor），用于监控虚拟机与Hypervisor的交互过程，实现虚拟机退出重定向。最后，实现了高效的虚拟机内存高强度隔离VM Isolation（虚拟机地址映射监控），通过隔离虚拟机和Hypervisor之间的内存，控制内存访问，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（如图2），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。



图2. 内存高强度隔离效果图

虚拟机安全套件是在X86平台使用KVM虚拟机监控器和多台Linux系统的客户机完成的，该系统适用于商业云平台提供商，有一定的可移植性、一定的适用性。该系统的性能开销是可接受的，在安全性上能够使得虚拟机之间的内存高强度隔离，阻止虚拟机跨域攻击带来的内存覆写等问题；能够保证虚拟机运行时CPU状态的完整性。

* 虚拟机安全套件功能说明

**地址空间隔离**

隔离空间创建——使用两套不同的内核页表创建另一安全执行环境，两个环境（普通地址空间和安全隔离地址空间）位于不同的地址空间。

安全切换门——实现两个地址空间的切换，保证切换的安全性，切换不可绕过。

隔离地址空间安全防护——在隔离空间中监控关键的特权寄存器的更新，监控MMU中虚实地址映射（MMU更新），监控DMA映射。

**交互关键数据监控**

隐藏VMCS结构体在隔离地址空间和剥夺宿主机的VMCS结构体访问功能，阻止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，阻止篡改宿主机或者虚拟机的特权寄存器（CR0、CR3、CR4）等关键信息。

**虚拟机地址映射监控**

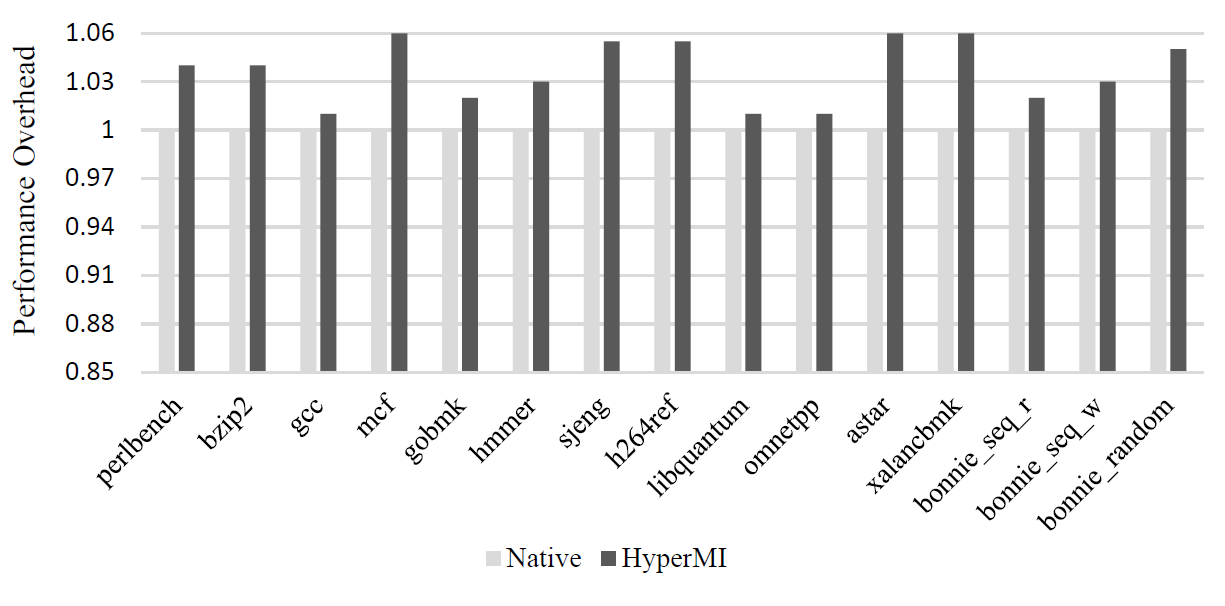
EPT监控——隐藏EPT的地址在隔离地址空间中，监控EPT的更新，保证EPT不会被恶意篡改。

虚拟机内存高强度隔离——物理页动态标记与跟踪，虚拟机物理内存隔离，内存页共享接口设定，实现虚拟机内存高强度隔离，减轻虚拟机跨域攻击带来的内存覆写问题。

* 虚拟机安全套件性能说明

**性能测试**

虚拟机安全套件所实现的三大功能的性能开销涨幅平均小于原系统的10%，使用宏观测试工具集，SPEC CPU和Bonnie++进行测试，测试结果表明spec cpu所测试结果中最高性能开销是6%（内存分配操作相关），bonnie++的测试结果都小于10%，主要是IO操作相关的更改较少。



虚拟机启动关闭时间的测试表明了开启和关闭虚拟机的时间延迟是1.1倍和1.08倍。

**微观测试Lmbench**

一套简易可移植的，符合ANSI/C 标准为UNIX/POSIX 而制定的微型测评工具。一般来说，它衡量两个关键特征：反应时间和带宽。测试表明主要在内存读入反应时间增加，其余方面的影响不大。

|  |  |  |
| --- | --- | --- |
| 带宽测评工具 | 反应时间测评工具 | 其他 |
| 拷贝内存  读内存  写内存  管道  TCP  读取缓存文件 | 上下文切换  网络：连接的建立，管道，TCP，UDP 和RPC  文件系统的建立和删除  进程创建  信号处理  上层的系统调用  内存读入反应时间 | 处理器时钟比率计算 |

**稳定性测试**——测试软件的运行稳定性，连续加卸载1000次；一定规模的虚拟机的最大运行个数是10个；虚拟机安全套件和虚拟机最大运行时长超过7\*24小时。

* 虚拟机安全套件代码组成说明

整个虚拟机安全套件系统包含两个模块，内核系统linux-4.4.1的patch\_SecureVM补丁和可加载内核模块SecureVM。首先对linux-4.4.1系统进行打补丁，在内核系统linux-4.4.1上运行模块SecureVM，提供安全的虚拟机运行环境，保障虚拟机的安全性，性能开销会较未更改的系统性能开销稍微大。

* 虚拟机安全套件运行环境说明

运行环境是Linux系统，使用已提供的内核代码，qemu版本在2.6以上，Libvirt在3.0.0版本以上，并安装qemu-KVM。

* 虚拟机安全套件使用权限说明

使用权限说明，内核系统编译过程需要root权限，SecureVM模块运行的过程需要root权限。

## 使用须知

正常的使用流程、对权限等方面的具体说明、一些关于使用的规范等常见问题。

虚拟机安全套件系统软件包括patch\_secureVM和SecureVM模块。

首先，在linux系统上用root权限编译内核系统linux-4.4.1，（建议从低内核版本向高内核版本编译，即原装的系统内核版本号低于4.4.1）。在编译的过程中，配置内核的虚地址翻译为4级页表，该过程在make oldconfig确定。

其次，使用KVM虚拟化系统，安装qemu、qemu-kvm、libvirt，配置虚拟网络，安装步骤可以参考网站<https://blog.csdn.net/u013593323/article/details/76709678?utm_source=blogxgwz0>。

当linux-4.4.1内核版本确定后，需要将patch添加到内核中，共4个补丁，使用命令patch -p1 < patch\_include\_linux（其余类似）即可。SecureVM系统模块运行时，需要按照LKM模块加载的方式去运行。在此过程中，需要以root权限执行。程序执行顺序是先加载SecureVM.ko，再开启虚拟机，运行虚拟机，关闭虚拟机，卸载SecureVM.ko。该顺序不可更改，否则虚拟机系统会崩溃。

## 系统使用

启动/停止/配置（包括检查系统状态）

### 系统安装

该部分主要完成指定内核安装、环境配置、SecureVM模块环境配置。指定内核安装是指安装Linux-4.4.1内核，基本步骤包括设置系统配置文件、编译、安装内核；环境配置是指安装qemu、libvirt、qemu-kvm等，并配置虚拟网络；SecureVM模块环境配置是指配置模块执行的文件。

该安装实例过程是在ubuntu14.05-server系统，内核linux-4.4.1，2台虚拟机，环境下完成。

* 指定内核安装：设置系统配置文件、编译、安装内核三个步骤，安装过程可以查看内核编译安装网站<https://www.cnblogs.com/wang_yb/p/3899439.html>。

|  |
| --- |
| 将内核源代码放在/usr/src/下，进入linux-4.4.1目录下。 |

|  |
| --- |
| 配置编译选项（5选1）：  make menuconfig :: 源码根目录下生成 .config (没有会自动生成)， .config中就是各个内核编译选项的选择状况.  make defconfig :: 根据当前系统的架构默认 .config 生成内核源码目录下的 .config (每个架构的配置文件: ex. arch/x86/configs/x86\_64\_defconfig)  make oldconfig :: 将已有的 .config 放到源码根目录下后执行， 目的是为了复用之前的内核编译选项的配置.  make xconfig :: 图形化配置， 需要qt3， 个人觉得没有必要， 有 make menuconfig 就足够了.  make localmodconfig :: 生成以正在使用的内核模块为对象的 .config  注：服务器上常用make oldconfig，默认系统开启的功能选项 |

|  |
| --- |
| 编译：  内核编译选项设置好之后， 只需简单的命令make -j N (N是并发数)， 就可以编译。 |

|  |
| --- |
| 安装内核：  make modules\_install (安装内核模块到 /lib/modules 下)  make install (安装内核二进制映像， 生成并安装boot初始化文件系统映像文件) |

* 打补丁

打开文件夹patch\_SecureVM，进入到/usr/src/linux-4.4.1，执行如下命令对内核打补丁。

patch -p1 < patch\_include\_linux

patch -p1 < patch\_asm\_x86

patch -p1 < patch\_mm

patch -p1 < patch\_drivers\_virtio

解除补丁：

patch -p1 -R < patch\_include\_linux

patch -p1 -R < patch\_asm\_x86

patch -p1 -R < patch\_mm

patch -p1 -R < patch\_drivers\_virtio

* 环境配置

使用KVM虚拟化系统，在**服务器**宿主机上安装qemu、qemu-kvm、libvirt，配置虚拟网络，安装步骤可以参考网站<https://blog.csdn.net/u013593323/article/details/76709678?utm_source=blogxgwz0>。

其中，使用virt-install（命令行）安装虚拟机，并非使用virt-manager（图形化界面）。在**远端的windows系统**上自行安装VNC软件（方便连接服务器），可不安装，根据所使用的系统自行处理。

Virsh创建虚拟机：

|  |
| --- |
| virt-install --name vm2 --disk path=/home/juku/image/vm2.img，size=20 --ram 2048 --vcpus 4 --cdrom /usr/local/src/CentOS-6.7-x86\_64-bin-DVD1.iso --network bridge=br0 --vnc --vnclisten=0.0.0.0 --vncport=5901 |

创建的虚拟机硬盘大小20G，内存2G，4个vcpu，名字为vm2，网络使用桥接br0，vnc访问端口5901。

Virsh常用命令：

|  |
| --- |
| virsh start x 启动名字为x的非活动虚拟机  virsh list 列出虚拟机  virsh create x.xml 创建虚拟机（创建后，虚拟机立即执行）  virsh suspend x 暂停虚拟机  virsh resume x 启动暂停的虚拟机  virsh shutdown x 正常关闭虚拟机  virsh destroy x 强制关闭虚拟机  virsh undefine x 删除虚拟机 |

Virsh使用方法可查看网站：[virsh使用方法](https://www.cnblogs.com/reblue520/p/6239796.html)

* SecureVM模块环境配置

在任意路径下创建文件夹code，并将SecureVM代码放在该文件夹下。

|  |
| --- |
| 创建文件夹：mkdir code |

### 系统加载/卸载

* SecureVM模块加载

进入code/SecureVM/下，编译、加载模块。

1. make
2. modprobe virtio\_balloon.ko

2）insmod secureVM.ko

* 开启、运行、关闭虚拟机

1. virsh start vm1

2) virsh shutdown vm1

* SecureVM模块卸载

rmmod secureVM.ko

## 功能说明

给出系统的API使用配置等，有界面的用给出图示。

整个虚拟机安全套件系统能够保证虚拟机的安全性，对于虚拟机和用户来说，保护是透明的。

* **系统运行功能调用**

1)  加载系统代码 insmod secureVM.ko

2) 卸载系统代码 rmmod secureVM.ko

3) 虚拟机访问基本命令

virsh list #列出在运行的虚拟机

virsh start vm1 #启动vm1虚拟机

virsh shutdown vm1 #关闭vm1虚拟机

virsh destroy vm1 #强制关闭vm1虚拟机

virsh undefine vm1 #删除vm1虚拟机

virsh suspend vm1 #暂停/挂启vm1虚拟机

virsh resume vm1 #从暂停状态还原vm1虚拟机

4）为单个文件生成补丁

diff -up linux-2.6.28.8/net/sunrpc/svc.orig.c linux-2.6.28.8/net/sunrpc/svc.c > patch

这条命令会产生类似如下的输出， 你将它重定向到一个文件中， 这个文件就是patch. 命令中是先未更改文件（ linux-2.6.28.8/net/sunrpc/svc.orig.c），后更改过文件。

参数详解:

-u 显示有差异行的前后几行(上下文)， 默认是前后各3行， 这样， patch中带有更多的信息.

-p 显示代码所在的c函数的信息.

5）为多个文件生成补丁

diff -uprN linux-2.6.28.8.orig/net/sunrpc/ linux-2.6.28.8/net/sunrpc/ > patch

这条命令对比了linux-2.6.28.8.orig/net/sunrpc/和linux-2.6.28.8/net/sunrpc/两个目录下的所有源码差异.

参数详解:

-r 递归地对比一个目录和它的所有子目录(即整个目录树).

-N 如果某个文件缺少了， 就当作是空文件来对比. 如果不使用本选项， 当diff发现旧代码或者新代码缺少文件时， 只简单的提示缺少文件. 如果使用本选项， 会将新添加的文件全新打印出来作为新增的部分.

6）打补丁

生成的补丁中，路径信息包含了你的Linux源码根目录的名称，但其他人的源码根目录可能是其它名字，所以，打补丁时，要**进入Linux源码根目录**，并且告诉patch工具，请忽略补丁中的路径的第一级目录(参数-p1).

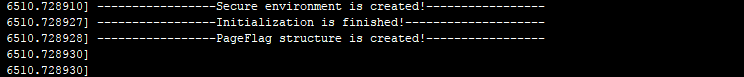
patch -p1 < patch1.diff

1. 补丁卸载

patch -p1 -R < [补丁包路径]patch\_name

* **系统初始化**

unsigned long init\_struct(void);//初始化系统，保证虚拟机安全套件正常启动，新页表被创建，安全隔离的地址空间被创建，使用到的pageflag结构体（描述page-mark表）被创建。



* **安全隔离地址空间**

**1）切换门关键步骤**

|  |
| --- |
| pgd\_addr\_cp\_gate = \_\_va(read\_cr3());  oldcr3 = read\_cr3();  DISABLE\_INT\_FIRST();  writecr3(\_\_pa(pgd\_addr\_cp\_gate));  \_\_flush\_tlb\_all();  DISABLE\_INT\_SECOND(); ENABLE\_INT();  writecr3(oldcr3); |

**2）创建安全隔离地址空间的关键函数**

void set\_pgd\_cp(void)

用于创建新页表，即安全隔离的地址空间。

unsigned long get\_pgd\_cp(void)

用于获取隔离地址空间的入口地址，即新页表入口地址。

**3）保护隔离空间的关键函数**

|  |  |
| --- | --- |
| 函数 | 描述 |
| set\_hook\_pte( (void\*)native\_set\_pte，(void\*)new\_native\_set\_pte );  set\_hook\_pmd( (void\*)native\_set\_pmd，(void\*)new\_native\_set\_pmd );  set\_hook\_pagefault( (void\*)do\_page\_fault，(void\*)new\_do\_page\_fault); | 分别对pte、pmd的访问监控，阻止对pte和pmd随意篡改，以及对页表所在页进行访问控制，阻止恶意篡改页表所在页，通过监控pagefault来实现对页表所在页的篡改阻止。 |

* **虚拟机监控**

主要实现虚拟机与宿主机之间的交互数据监控，通过控制对VMCS结构体的写访问实现，关键函数如下：

|  |  |
| --- | --- |
| 函数 | 描述 |
| set\_hook\_vmcs\_readl( (void\*)vmcs\_readl，(void\*)new\_vmcs\_readl);  set\_hook\_vmcs\_writel( (void\*)vmcs\_writel，(void\*)new\_vmcs\_writel);  set\_hook\_vmcs\_clear( (void\*)vmcs\_clear，(void\*)new\_vmcs\_clear); | 分别实现对vmcs\_readl vmcs\_writel vmcs\_clear的挂钩。 |
| unsigned long new\_vmcs\_readl(unsigned long field);  void new\_vmcs\_writel(unsigned long field， unsigned long value);  void new\_vmcs\_clear(struct vmcs \*vmcs); | 主要在安全隔离地址空间中实现了对VMCS的读、写访问和销毁操作。 |

* **内存标记与跟踪**

|  |  |
| --- | --- |
| 函数 | 描述 |
| set\_hook\_handle\_pfn\_modify((void\*)handle\_pfn\_modify，(void\*)new\_handle\_pfn\_modify);//挂钩函数  int new\_handle\_pfn\_modify(pfn\_t pfn，struct kvm\_vcpu \*vcpu) //验证页映射是否合法 | 该函数主要是验证当前即将被分配的物理页上的pageflag信息，判断是否是属于当前虚拟机（pageflag属主是否是当前虚拟机）或者该页从未被使用过(usedID=0)，否则的话，映射阻止。 |
| set\_hook\_handle\_pfn\_map((void\*)handle\_pfn\_map，(void\*)new\_handle\_pfn\_map);//挂钩函数  int new\_handle\_pfn\_map(pfn\_t pfn，struct kvm\_vcpu \*vcpu)// 分配映射 | 已经验证过物理页是合法的，即将分配映射，在分配前设置该页的pageflag结构体。 |

* **内存安全释放**

|  |  |
| --- | --- |
| 函数 | 描述 |
| set\_hook\_clean\_page\_modify((void\*)clean\_page\_modify，(void\*)new\_clean\_page\_modify);//挂钩函数 | 实现对clean\_page\_modify的挂钩。 |
| int new\_clean\_page\_modify(struct page \*page);//内存释放函数 | 该函数主要实现内存安全释放，拦截即将被释放的物理页，释放相应的pageflag结构体，并清空其上内容，设置为零。 |

* **虚拟机内存高强度隔离功能**

1）内存复用balloon机制下，虚拟机内存隔离

|  |  |
| --- | --- |
| 函数 | 描述 |
| set\_hook\_inflate\_page\_modify( (void\*)inflate\_page\_modify，(void\*)new\_inflate\_page\_modify );//balloon针对内存膨胀的挂钩函数  set\_hook\_deflate\_page\_modify( (void\*)deflate\_page\_modify，(void\*)new\_deflate\_pag  e\_modify );//balloon针对内存收缩的挂钩函数 | inflate\_page\_modify和deflate\_page\_modify 这两个内核函数主要用于balloon机制中“气球”膨胀或者收缩时释放内存给其余虚拟机或者从其余虚拟机中将内存收回，在该过程中物理内存的属主会发生变化，new\_inflate\_page\_modify和new\_deflate\_page\_modify这两个函数的主要作用是两种功能发生作用时对物理页内存的属主实时变更，更新对应的pageflag结构体，避免违背虚拟机内存高强度隔离功能中一个物理页对应一台虚拟机的基本原则。 |

2）内存复用KSM机制下，虚拟机内存隔离

set\_hook\_ksm\_page\_modify((void\*)ksm\_page\_modify，(void\*)new\_ksm\_page\_modify);//挂钩KSM机制中页合并ksm\_page\_modify函数，跳转到new\_ksm\_page\_modify函数处理；KSM机制主要是对包含有相同数据的物理内存页进行合并，可能合并的是同一虚拟机的或者不同虚拟机的物理页，实现这一功能的主要内核函数是ksm\_page\_modify。那么对应的挂钩函数new\_ksm\_page\_modify的功能是拦截即将被合并的页，更新其pageflag结构体，打印属主信息变化等。

打印在内存复用情况下部分物理页的使用情况及pageflag结构体相关信息，如下图。

