电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE

论文题目 基于 oVirt/Qemu/Kvm 云平台系统分析与安全加固设计

专业学位类别 工程硕士

学号 201222240914

作者姓名 林雪峰

指导教师 刘丹 副教授

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

基于 oVirt/Qemu/Kvm 云平台系统分析与安全加固设计

（题名和副题名）

林雪峰

（作者姓名）

指导教师 刘 丹 副教授

电子科技大学 成 都

（姓名、职称、单位名称）

申请学位级别 硕士 专业学位类别 工 程 硕 士

工程领域名称 电子与通信工程

提交论文日期 2015.4 论文答辩日期 2015.5

学位授予单位和日期 电子科技大学 2015 年 6 月 日

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：林雪峰　　日期：2015 年 6 月 3 日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：林雪峰　　导师签名：

日期：2015 年 6 月 5 日

SYSTEM ANALYSIS AND SECURITY REINFORCEMENT DESIGN BASED ON OVIRT/QEMU/KVM CLOUD PLATFORM

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Major: **Master of Engineering**

Author: **Lin xuefeng**

Advisor: **Associate Professor Liu Dan**

School : **Research Institute Electronic Science and Technology**

摘　要

随着云计算产业如火如荼的发展，云办公环境已经不再只是停留在概念阶段上，越来越多的企业和公司把自己的办公环境从原来的传统办公模式转移到崭新的云办公平台上。与传统办公环境相比，给企业带来了提高办公效率、公司资料统一管理和降低企业开销等优点。然而，同时也带来了新的安全隐患，例如：在 oVirt 云平台中，虚拟机磁盘数据是以明文的形式存放在服务器上，并没有考虑到虚拟机磁盘数据安全问题，以及公司内部员工可以利用 USB 设备随意拷贝公司资料，容易造成公司内部数据泄露的安全问题等。

本文的研究正是源于 oVirt 云平台安全方面的需求，从 oVirt 云平台安全方面问题入手，以保护 oVirt 云平台用户以及公司敏感数据安全。针对现有的云平台的攻击方式，结合 Qemu、KVM 和 VDSM 源码详细分析，重点分析了 oVirt 云平台框架、oVirt 云平台磁盘存储管理、oVirt 云平台文件系统和虚拟机磁盘镜像文件读写流程，并通过实验证明了虚拟机磁盘镜像文件、虚拟机磁盘删除和 USB 设备管理在这三方面存在着严重的安全隐患。

本文通过上述 oVirt 云平台系统分析所发现的安全问题, 制定了 oVirt 云平台安全加固的解决方案，同时设计并实现了 oVirt 云平台安全加固系统，本系统主要包括了虚拟机磁盘镜像加解密、虚拟机磁盘深度擦除和 USB 设备实时管控三个子系统。

虚拟机磁盘加解密子系统是在 Qemu 层中增加了密钥管理模块和磁盘数据加解密模块，设计了用户登录身份认证，确保了用户登录的安全。并通过磁盘数据加解密模块，对虚拟机磁盘镜像文件数据进行加解密，达到了防止黑客以及内部管理人员窃取虚拟机用户数据的作用。

虚拟机磁盘擦除子系统是在 VDSM 层中增加了深度擦除模块，采用了权限认证和多安全级数据擦除机制，给用户提供了多种虚拟机数据擦除方式，虚拟机用户数据通过数据覆写的方式进行深度擦除，使得虚拟机用户数据经擦除后难以恢复。

USB 设备实时管控子系统是在 Qemu 层中增设了两个白名单认证模块和访问控制模块，通过修改 Qemu 中 USB 设备重定向机制，添加 USB 设备授权访问，增设一个 USB 设备白名单来实现 USB 设备实时管控, 公司管理员能够实时对员工的 USB 设备进行管控。

经过测试，本 oVirt 云平台安全加固系统功能上达到了设计目标，在性能方面

虚拟机磁盘加密、虚拟机磁盘擦除以及 USB 设备实时管控对系统效率影响均在合理范围内，达到了预期目标。

本课题的分析研究顺应了云安全发展趋势，解决了基于 oVirt/Qemu/Kvm 云平台存在的安全问题，促进我国云办公的安全发展。

关键词：oVirt 云平台，KVM-QEMU，虚拟化，云平台安全

ABSTRACT

With the rapid developm ent of the cloud computing industry, the cloud of fice system is no longer a conception. More and m ore corporations shift from the traditional office system, to the newly of fice system based on cloud com puting platform. Compared to the traditional of fice system, cloud office system bears the m erits of improving work ef ficiency, centralizing cor porations' data m anagement and lowering the corporations' cost.However, the new data security risks exist as well,For examples, on the oVirt cloud platform, the virtual machine disk stored on the servers with plaintext format and the data security problem s of th e virtual machine disk are not taken in to considerations. Internal employees may use USB devices to duplicate corporation files and causes the internal data leakage problems.

The research of this paper is from the need of the oV irt cloud platform security requirements, which is from the security problems of the oVirt cloud platform to protect the oVirt cloud platform users and senditive company data security. Focused on the attack existing methods of the cloud platform, with studying the source code of QEMU、KVM and VDSM, we analy ses the fram ework of the oV irt cloud platform , the management of the disk storage and the file system of the oV irt cloud platform, at the same time, we proves that there are serious security hidden danger in the virtual machine disk im age files, delete virtu al machine disk and USB device m anagement through the experiment.

By analyzing the safety problem s we found on oVirt Cloud platform, we designed security reinforcement solutions for oVirt Cloud platform and im plements security reinforcement system for oVirt Cloud platform at the same time. Which mainly includes the secutity on thre e subsystems of the en cryption of the virtual m achine disk, data deletion of the virtual machine and real-time management of the USB devices.

The encryption of the virtual machine disk has the key management module and the disk data encryption module in QEMU, we design a m echanism for user authentication, which can ensure the safety of the user login. Through encrypting the disk data in QEMU to prevent h ackers or adm inistrators from stealing client data on the v irtual machine.

Data deletion of the virtual machine disk has the erasing module, which is a method

to adopt the authority certification and multiple security level data deletion mechanism in VDSM, by overrid ing the data to deeply delete c lient data on the virtu al machine, which makes the restoring of the client data on the virtual machine impossible.

Real time management of the USB devices has the whitelist authentication m odule and access control module in Qem u, which is a m ethod to change the redirect mechanism of the USB devices in QEMU a nd add authorization for U SB devices t o access. It helps corporation 's administrators by adding a whitelis t for certain USB devices to access to realize the real-time management of the USB devices. From the three aspects m entioned above, the purpose to enhance security of the oV irt cloud platform can be reached.

It has been tested that security en hancement functions applied on oV irt cloud platform fulfils the design purpose, the system efficiency affected by the encryption of the virtual machine disk, data deletion of the virtual machine and real time management of the USB devices remains in the reason able range, the results m eets the expected goals.

The analysis and research of th is subject keep with the trend of the security development of cloud technology . It prom otes the data security of the of fice system based on the oVirt/Qemu/KVM cloud platform, and actively pushes forward the security research of the cloud technology in China.

Keywords: oVirt cloud platform, KVM-QEMU, virtualization, cloud platform security

目 录

第一章 绪 论

1.1 研究背景及意义

最早在 20 世纪 90 年代，云的概念被美国人 John McCarthy 所提出，当时他提出了把计算机能力可以作为一种服务来销售，像水、电一样的公用事业提供给用户[1,2]，这便是云思想的起源。从 2006 年，谷歌首次提出了“云计算[3]”的概念以来，随着云计算、云存储、虚拟化等关键技术的迅猛发展，云应用的时代已经悄然来到我们身边，成为了人们生活的一部分。如今，越来越多的公司把自己的办公环境从原来的传统办公模式转移到崭新的云办公平台上。云办公环境利用云计算、云存储和桌面应用等虚拟化技术和远程桌面技术，根据企业实际办公需求，构建办公环境。与传统办公环境相比，云办公环境具有提高服务器资源利用率、降低整体成本、集中式统一管理、可扩展性、智能备份容灾、移动办公等优点。

基于上述理念，红帽公司也推出了基于 KVM[5]项目的 oVirt 开源软件，KVM 开源虚拟化技术具有高安全、高可扩展性、高性能等优势，已经逐渐成为企业级 hypervisor。然而，如果需要部署一个成熟的虚拟化环境，仅仅有 hypervisor 还是不够的。虚拟化管理越来越重要，它能同时管理主机与客户机，还要安全动态迁移、数据访问控制和高可用性等高级功能。oVirt[6]（open Virtualization）就是这样一个开源的桌面管理平台，是 redhat 虚拟化管理平台 RHEV 的开源版本，提供基于 Web 的虚拟机控制管理平台，是一个基于 KVM（Kernel-based Virtual Machine，基于内核的虚拟机）的开源 IaaS 项目，支持主流的 x86 硬件，并允许用户在其上运行 Windows 及 Linux 操作系统。无论一台主机上有几个虚拟机，还是管理数百台主机上的成千个虚拟机，都能很好的对服务器上虚拟机进行控制管理。因此，oVirt 专门面向那些基于 KVM 虚拟化人群，为主机和客户机提供功能强大和特性丰富的服务器虚拟化管理系统，具体包括虚拟机动态迁移、高可用性、系统调度管理、存储管理等。

虽然云平台办公环境给企业带来了不少好处，可提高办公灵活性、办公效率、节省企业硬件资源开销、方便资源集中管理。但是，世界上任何事物都具有两面性，云办公环境同时也增加了更多的安全风险。从 2011 年以来，苹果、谷歌、微软、亚马逊等公司的云平台都受到了攻击，用户数据被泄漏，给用户造成了巨大的影响，用户对自己保存在平台上的数据安全性产生质疑。如何去保证企业私有云平台的数据安全，则是企业私有云构建者需要去思考的问题。

在基于 Qemu/Kvm 的 oVirt 开源虚拟化管理平台中，该系统的设计并没有考虑

到磁盘数据安全问题，所有客户机磁盘数据都是以明文的方式存储在服务器上，这样云平台攻击者很轻易的就能获取到服务器上的用户隐私数据，造成用户数据泄漏，用户磁盘数据丝毫没有安全性可言。所以，有必要在 oVirt/Qemu/Kvm 基础之上，设计一套保护用户磁盘数据安全的方案，对用户磁盘数据进行加密，用户在不想要虚拟机时，对虚拟机磁盘进行复写擦除，在这种情况下，即使云平台攻击者得到了服务器磁盘或者磁盘数据时，也很难恢复破解磁盘数据得到用户真实数据。另外，通过研究发现 oVirt 云平台缺乏 USB 设备安全管控[7]，本文也在 oVirt 云平台上增加了 USB 实时管控模块，保证企业服务器中的数据不能随意拷贝带走，需要企业允许的情况才能使用 USB 设备，从而在很大程度上提高了云平台用户数据安全性。

1.2 云平台国内外发展现状

云计算是在网络基础上，通过虚拟化的方式实现对 IT 资源共享的一种新型计算模式，实现了配置优化和资源整合，不同的用户可以随时获取不同的服务方式，支持可扩展性，用户根据自己需要使用来付费，最大程度地降低了用户成本等各种需求。

云计算按照归属可以分为三种，分别是：私有云、公有云、混合云[8]。私有云是面向于公司企业，把私有云搭建在自己的数据中心内，属于企业自己所有，以提高企业办公效率，统一资源管理和减少开销为目的。而公有云则是大众提供各种服务，不属于某一家公司所拥有，以获取盈利而存在。混合云则介于私有云和公有云之间。云计算被认为是继计算机、互联网诞生与广泛应用之后的又一次重大变革，将给商业模式和工作方式带来根本性的变化。

目前，云计算产业在国外，特别是美国的云计算市场为代表的已经相当成熟了，但在中国还处于发展的初级阶段：整体规模较小，仅占全球云计算产业市场份额 3%。

在国外，云计算产品应用市场发展势头迅猛，世界主要信息强国都十分关注云计算及其应用，各国都纷纷制定了本国的云计算发展计划，制定了发展策略和政策，把云计算作为了国家的发展战略重心之一。美国是云计算的发源地，也是云计算市场发展最快、规模最大的国家，美国政府广泛推进云计算应用，在联邦、洲和地方各级政府机构中广发应用。2011 年，美国政府制定了“ 联邦云计算战略 ”，优先使用云计算服务，在实施的这几年里的效果来看，云计算服务对美国来说取得了显著的成效，不仅方便了整个社会的信息管理，而且降低了美国财政开支。欧洲也紧随美国的步伐，加强云计算的投入，同时制定了 “ 第七框架计划 ”，越来

越多的政府、医院、企业等机构采用了云计算服务。日本也非常重视云计算应用的发展，投入大量资金发展本国云计算基础设施建设，从而社会和政府服务提供便利，并谋求利用云计算来创造出新的服务和产业。IBM、Google、Microsoft、Amazon 等著名的电子信息公司都推出云计算产品和服务[9]，英特尔、思科等传统硬件厂商也向云计算服务转型，云计算已经受到了国际市场的高度重视和关注。

在国内，中国的云计算产业也如火如荼的发展，各地区都纷纷大力发展云计算产品，并投资建设云计算基础设施。在政府的监管之下，云计算服务提供商以基础设施、平台或者软件的方式来为个人、企业和政府提供云计算服务。但由于目前处于起步阶段，云计算的快速产业化还存在诸多障碍：用户认知不足、缺乏标准、稳定性和可用性担忧，难以规范服务质量。其中标准和安全以及相关法律制定完善是最核心的，也是最迫切需要解决的问题。

然而，在政府的支持下，目前我国各大城市也在纷纷建设与云计算相关的项目，例如：北京、宁波、上海、广州、廊坊等城市都建有云计算服务基地，并通过联合大型 IT 企业，积极推动云计算的研究和发展起到了良好作用。未来的发展将是一个巨大的突破，预计在未来几年，中国云计算市场规模将以超过 80%的年均复合增长率快速发展，到“十二五”末，产值将超过 1 万亿元。

云计算推动 IT 行业转向以业务为中心模式的重大变革，其重点是着眼于运营效率、快速响应和竞争力等实际成果。云计算的表现形式多种多样，简单的云计算在日常网络应用中无处不在，如 Google 的搜索服务，腾讯 QQ 空间提供的在线 Flash 图片和谷歌企业应用套件等。目前，云计算服务[10]的主要形式有：SaaS（Software as a Service，软件即服务）、PaaS（Platform as a Service，平台即服务）和 IaaS（Infrastructure as a Service，基础设施既服务）。

相对而言, 云计算的架构是 IaaS 为底层基础, PaaS 为中间层, SaaS 为最上层。云计算体系架构如图 1-1 所示：

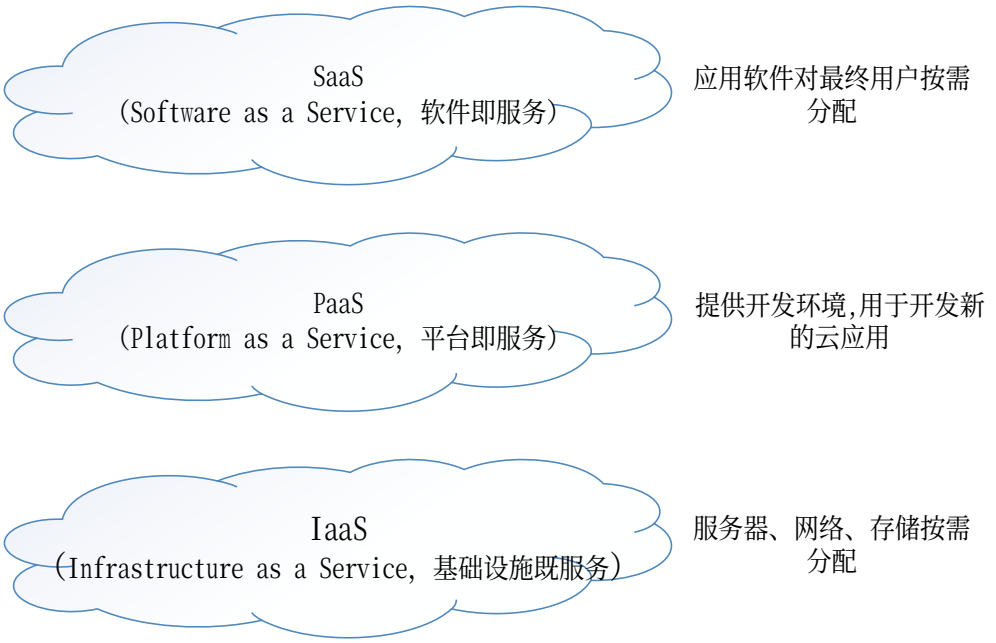


图1-1　云计算服务模型体系

随着业界对云计算的深入研究以及各大 IT 公司的大力推广，这三层有逐渐融合的趋势，云计算提供商也开始在各个层面分别推出自己的云计算产品来抢占先机。

1.3 云平台安全状况

随着云计算产业如火如荼的发展，越来越多的公司企业将自己的办公环境转移到云平台办公环境上，其起到了提高资源利用率，提高办公效率，降低企业开销的目的，与此同时，云平台也可能会破坏传统的安全架构、安全措施和安全配置方案，带来新的安全问题。

企业对云技术方面尚还处于探索阶段，通过它来提高运营效率仍处于起步阶段，给企业带来营业成本的大大降低。这样的情况下，使得企业对云技术带来的安全问题不够关心。并且云技术本身也还处于发展阶段，意味着企业和机构的安全防御手段显得乏力和薄弱。但是，为了紧随时代的步伐，云技术的引入和应用是必须的，我们也必须要面对云技术带来的安全问题，除了技术本身的不断发展和完善，我们也必须寻找一种能对数据源安全防护的技术，在云技术的过渡期中，确保用户数据安全可靠，这种技术就是加密技术，对用户虚拟机镜像文件进行加密。

在云平台中，用户使用虚拟机的磁盘镜像文件直接以文件的形式存储在服务器上，用户也可以随意创建、拷贝、修改、删除、读取虚拟机镜像文件，也可以

通过快照直接创建用户虚拟机，或者恢复到虚拟机的某一虚拟机状态，创建一个用户虚拟机就如同拷贝一个文件简单容易，这给用户虚拟机带了新的系统安全问题。例如：用一个存在漏洞或者病毒的虚拟机镜像快照大量创建用户虚拟机，致使新创建的用户虚拟机也有同虚拟机镜像相同的系统安全问题，在用户不知的情况下，这将会带来不可预计的严重的后果；用户也可能通过虚拟机快照不经意使虚拟机回滚到了有漏洞的某一状态，使得不法分子有了可乘之机，对用户虚拟机进行攻击，窃取用户的磁盘数据等等。

1.4 课题研究内容及目标

1.4.1 研究内容

1）研究 oVirt 云平台框架，以及其磁盘存储管理。

目前有关于 oVirt 的资料还很少，特别是技术性文档，由于涉及到数通底层到 oVirt-Engine 管理界面数据交互，分析 oVirt 云平台框架和 VDSM 源码，分析出从底层到界面往返的数据通路。

2）研究 Qemu-Kvm 磁盘虚拟化技术

在 oVirt 云平台中，磁盘镜像文件都是以明文的形式存放的，所以需要研究 Qemu-Kvm 磁盘虚拟化技术, 研究其 I/O 虚拟化过程中的 I/O 操作模拟、页面映射、内存操作等机制。分析源码，找到磁盘数据加解密的切入点。

3）研究磁盘加解密技术，各个模块设计

实现磁盘加解密算法开放性，设计各模块之间的接口和数据结构。

4）研究磁盘擦除技术

在 oVirt 云平台中, 虚拟机磁盘镜像文件删除后, 删除的数据其实还在硬盘上，只是修改了标记允许被新数据覆盖，大部分数据恢复软件可以将这些删除的数据恢复。所以研究磁盘擦除技术，在虚拟机删除时候，对磁盘数据进行复习擦除。

5）研究 USB 设备实时管控

在 oVirt 云平台中，缺乏对 USB 设备的安全管控，这样可以通过移动设备任意拷贝服务器上的数据，可能对企业带来安全隐患。所以研究在 oVirt 云平台上对 USB 实时管控，利用 KVM 和 Qemu 虚拟化技术实现管控的实时性，从而增强云平台的数据安全性。

1.4.2 研究目标

1）保障加密后系统安全

通过对 oVirt/Qemu/Kvm 源码分析，对 oVirt 云平台磁盘文件镜像加密后，磁盘镜像文件能像原系统中一样正常使用，读取时候能够正常解密。在错误密钥的情况下，无法对磁盘镜像文件进行解析。

2）加密算法的开放性

系统设计提供了开放的加解密接口，用户可以增加其它的加解密算法。

3）保障磁盘擦除后系统安全

在 oVirt 云平台中增加了磁盘深度复写擦除功能后，并不影响其它虚拟机的正常使用，保障系统安全。

4）降低性能影响

加密后的虚拟机对磁盘镜像文件读取速率的影响不超过 10%。

5）保证 USB 管控的可靠性和实时性

在 oVirt 云平台中增加了 USB 实时管控模块后，管理员能够实时的管理控制用户的 USB 使用状况, 保障经过授权的用户 USB 设备能够正常在远程虚拟机中使用，同时未经授权用户 USB 设备不能在远程虚拟机中使用。

1.5 论文组织结构

本文共分六章，对 oVirt/Qemu/Kvm 云平台文件系统、透明加解密系统和 USB 设备实时管控做了详细描述。

第一章是绪论部分，介绍了本论文的研究背景和意义，概述了目前云平台国内外发展现状以及安全状况，并对其研究总结，提出了本论文研究内容及目标。

第二章主要对 oVirt /Qemu/Kvm 磁盘加密关键技术研究分析, 介绍和分析 oVirt 框架、磁盘存储管理、Qemu-Kvm 磁盘虚拟化技术、加解密技术和磁盘擦除技术、USB 设备实时管控技术。其中大部分技术都是通过分析 oVirt/Qemu/Kvm 源码，通过调式验证所得出。

第三章结合 oVirt/Qemu/Kvm 源码分析，提出了方案设计需求和目标，并阐述了 oVirt 云平台的安全设计整体架构。

第四章根据上一章提出的总体方案，分别对 oVirt/Qemu/Kvm 磁盘加解密、磁盘深度擦除和 USB 设备实时管控各个模块详细设计，并通过流程图和关键代码来详细阐述。

第五章对本文设计出的云平台安全加固技术原型系统进行了功能和性能测试, 并对结果进行分析。

第六章是总结与展望，对本文的 oVirt/Qemu/Kvm 安全加固系统设计进行了总结，并展望下一步的研究方向。

第二章 oVirt 云平台安全加固基础研究

2.1 虚拟化技术

虚拟化技术是构建云平台基础的不可或缺的关键技术之一，虚拟化可以在一个物理平台上虚拟机出多个不同平台的虚拟机，相比直接使用物理平台而言，虚拟化技术能更有效的利用资源，动态调配更灵活，高可靠性等优势。企业可以利用虚拟化技术，在不抛弃现有的基础设施情况下，搭建全新的信息基础架构，更加充分地利用现有的 IT 资源。

2.1.1 软件虚拟化和硬件虚拟化

对于虚拟化，虚拟化层能够拦截计算元件对物理资源的直接访问，并将其重定向到虚拟资源池中是关键技术，如图 2-1 所示。根据虚拟化层是通过纯软件的方法来模拟硬件的访问，还少通过物理资源提供的机制来实现这种“ 截获并重定向 ”，我们可以把虚拟化技术分成两类：软件虚拟化和硬件虚拟化。

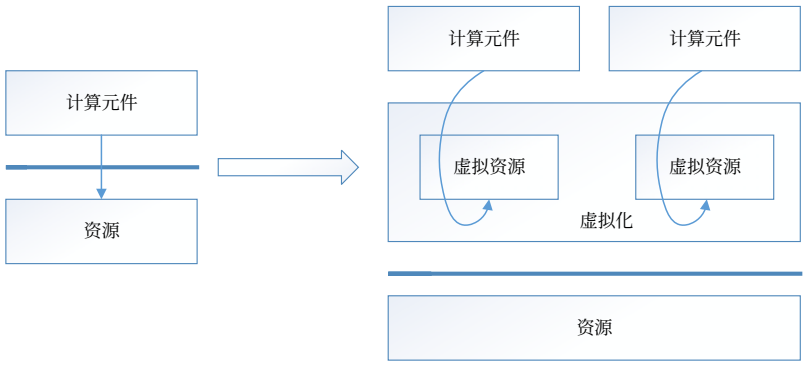


图2-1　虚拟化技术

1）软件虚拟化

软件虚拟化，简而言之就是用纯软件的方式，通过现有的物理平台对其访问的截获和模拟。常见的软件虚拟化如 Qemu，它是通过纯软件来模拟 X86 平台处理器的取指、解码和执行的，客户机的指令并不直接在物理平台上执行。由于所有的指令是通过软件模拟的，所以性能通常不是很好，但我们可以在同一个平台上运行多个架构平台的虚拟机。

在纯软件虚拟化解决方案中，客户机操作系统是通过虚拟机监视器来与硬件通信的，虚拟机监视器所处位置是传统意义的操作系统的位置，而客户机操作系统所处位置是在传统意义的应用程序位置，这就使得系统的复杂性增加。软件堆

栈复杂度增加，使得环境难于管理，从而会增加确保系统安全性的难度。

2）硬件虚拟化

硬件虚拟化技术，顾名思义就是在物理平台上提供了截获和重定向特殊指令的功能，在一些新的硬件还提供了额外的资源，可以帮助实现关键硬件资源的软件虚拟化，从而提高了性能。

在 X86 平台中，通过支持虚拟化技术的 CPU 所扩展的指令集来控制虚拟过程，通过它的指令集，虚拟机监视器能够很容易的监视客户机的运行状况，一旦客户机需要访问物理资源的时候，硬件会暂停客户运行，将控制权交给虚拟机监控器来处理。虚拟机监控器还可以利用硬件虚拟化增强机制，限制客户机对一些特定资源的访问，完全通过硬件重定向到虚拟监控器所指定的虚拟资源，整个过程并不需要暂停客户机。但是，硬件虚拟化需要 CPU、主板芯片组、BIOS 和软件的支持。

2.1.2 VMM 经典模型

VMM（Virtual Machine Manager）是在虚拟化环境中所有虚拟机的管理和调度中心， VMM 不仅充当了操作系统的部分或者全部角色，对上模拟虚拟机的硬件平台，对下需要管理真实的硬件设备。经过虚拟化技术的不断发展，VMM 技术可以分为三种模型：Hosted 模型、Hypervisor 模型和混合模型。这三种模型如图 2-2 所示：

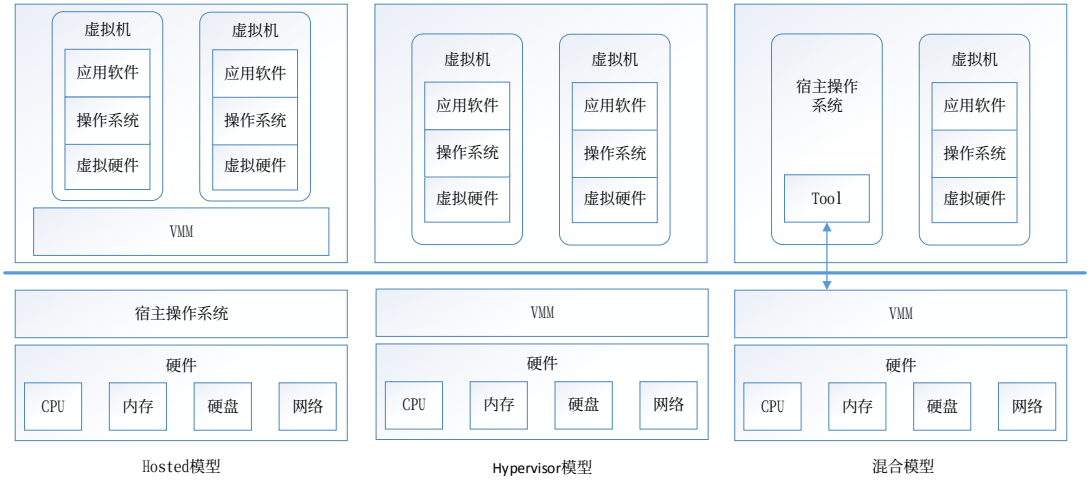


图2-2　　VMM 的三种模型

1）Hosted 模型

在 Hosted 模型中，我们可以看出宿主操作系统管理了物理硬件资源，VMM 需要向操作系统申请调用，来获取 CPU、内存和 I/O 设备的信息。它的优点就是

可以很好的利用宿主操作系统的设备驱动程序, 不需要再实现 I/O 设备的驱动程序, 这样 VMM 可以专注于提供虚拟化功能。它的缺点就是 VMM 需要通过调用宿主操作系统来获取硬件资源，使得虚拟化效率较低。

2）Hypervisor 模型

在 Hypervisor 模型中，VMM 不仅充当了操作系统的角色管理硬件资源，管理 CPU、内存和 I/O 设备等硬件资源，同时还需要向上提供虚拟化功能，负责 CPU 虚拟化、内存虚拟化和 I/O 设备虚拟化。VMM 同时具有管理物理硬件和提供虚拟化的功能，使得虚拟化的效率比较高，但是 VMM 需要重新实现操作系统的许多功能，增加了 VMM 实现难度。在虚拟机安全方面，虚拟机将依赖于 VMM 的安全。

3）混合模型

混合模型则均衡了上面两种模型的优缺点，VMM 也位于最底层，管理和控制大部分的物力资源, 将大部分的 I/O 设备的管理交给一个运行的特权虚拟机。VMM 和特权虚拟机系统共同合作，使得虚拟化效果比较好。

2.2 KVM 和 Qemu 虚拟化技术

2.2.1 KVM 架构

KVM 模块是 KVM 虚拟机的核心，也是 linux kernel 的一个模块，其主要功能是创建虚拟机，为虚拟机分配内存，读写 VCPU 寄存器和 VCPU 的运行，对虚拟机客户机的运行提供一定的支持。在 KVM 中, KVM 并不模拟任何设备, 需要 Qemu 通过下图 2-3 中的/dev/kvm 接口分配一个客户机的地址空间, 向 KVM 提供模拟的虚拟设备，以达到访问外设的目的。

KVM 模块使 Linux 宿主机成为了一个虚拟机监控器，并且在原有的两种执行模式基础上，新增了客户模式，客户模式有其自身的用户模式和内核模式。如图 2-3 所示：

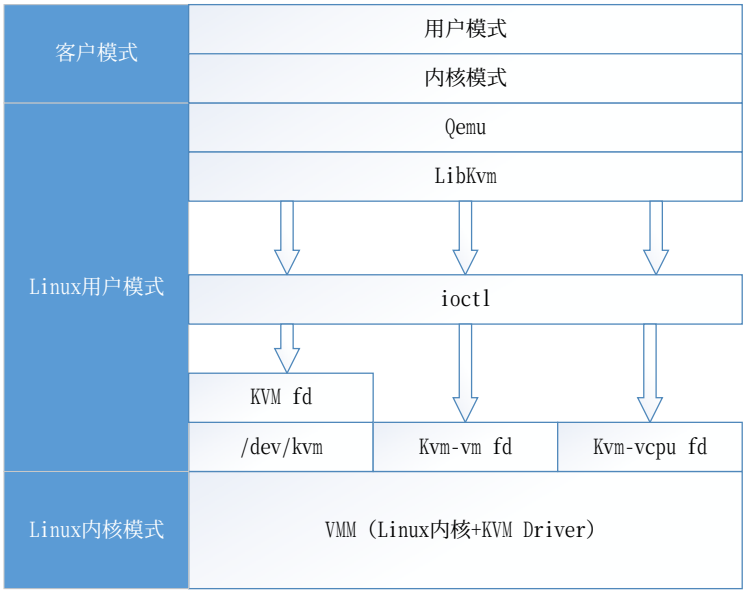


图2-3　KVM 架构

在虚拟机运行时，上图 2-3 中的三种模式的工作分别如下：

客户模式 :执行客户机中非 I/O 的客户代码, 虚拟机操作系统运行在该模式下。

用户模式：代表用户执行 I/O 的指令，Qemu 在该模式下运行。

内核模式：对客户模式切换的实现，因 I/O 操作或其他指令引起的客户模式的退出，需要在内核模式下的 KVM 来处理。

在上面 KVM 架构图中，也可以看出 KVM 是内核模块，它被看作成为一个标准的 linux 字符集设备（/dev/kvm）。Qemu 通过调用 KVM 提供的 LibKvm 应用程序接口，再用 KVM_fd 调用 ioctl 系统调用来访问/dev/kvm 字符设备来创建和初始化虚拟机。创建完虚拟机后，会返回一个 KVM-vm fd 虚拟机文件描述符，可以通过该文件描述用 ioctl 来向虚拟机发送命令，访问控制该虚拟机。而 Kvm-vcpu fd 文件描述符就对应了虚拟机的 VCPU，通过 ioctl 来调度和设置虚拟机的 VCPU。

2.2.2 KVM 客户机调度管理

客户机操作系统是由 VMM 来调度执行的，在 VMM 调度过程中，Qemu 调用 ioctl 来进入内核模式，接着通过在上图 2-3 中的 KVM Driver 获取当前物理 CPU，然后从 VMCS 结构读取客户机的状态信息，载入到该物理 CPU 中，通过 VMLAUCH 命令使物理 CPU 进入非根操作模式，执行客户机代码。

当客户机操作系统执行特权指令或其它重要事件时，比如 I/O 操作、对控制寄存器访问操作，产生缺页错误等，客户机会产生 VM Exit，客户机停止执行并退出到 KVM 中, 保存客户机操作系统状态保存到 VMCS 结构中, KVM 通过读取 VMCS

结构中的 VM_EXIT_REASON 获取 VM Exit 的原因，决定是由自己处理还是交给 Qemu 去处理。如果是 I/O 操作，就交给 Qemu 来处理，处理完成后，重新进入客户模式执行客户端代码。如果是外部中断，则通过 KVM 来处理，处理完后，重新进入客户模式执行客户端代码。

2.2.3 KVM 内存管理

KVM 内存虚拟化就是实现客户机虚拟地址到客户机物理地址再到主机物理地址的过程，转换的实现方式目前有两种：软件影子页表虚拟化和硬件内存虚拟化。

对于影子页表内存虚拟化，影子页表简化了内存地址转化的过程，实现了客户机虚拟内存地址到宿主机物理内存地址的直接映射。影子页表最初是空的，随着客户机操作系统对内存页表的访问修改逐步建立起来的，客户机中的每一个页表都一一对应一个影子页表。如图 2-4 所示：

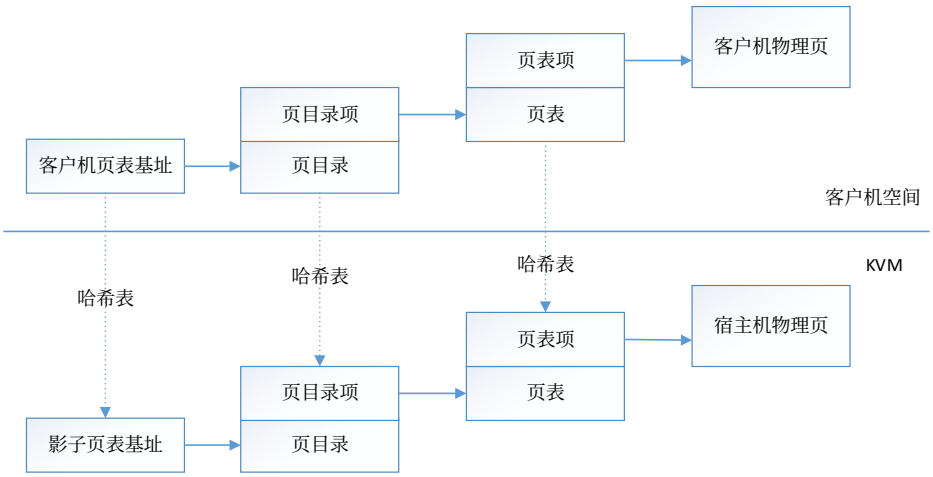


图2-4　客户机页表和影子页表

从上图可以看出，每个虚拟机都拥有一个哈希表，KVM 就是通过这个哈希表来实现客户机页表到影子页表之间的对应关系的，从而能够利用哈希表快速地从客户机页表找到对应的影子页表。在哈希表检索时，如果不为空，则表示生成了相应的影子页表。如果为空，说明 KVM 还没有为其建立影子页表，还需要生成新的影子页表，KVM 将分配新的物理页，同时建立客户机页表到对应影子页表的映射关系。在客户机操作系统中出现进程切换时，客户机操作系统将把等待切换的进程的页表基址载入 CR3 寄存器中，这一特权指令将会被 KVM 所截获，进行处理。同时通过哈希表把客户机页表基址对应的影子页表基址也载入到客户机 CR3 寄存器中，使得客户机在运行时候，访问的是 CR3 寄存器所指向的新影子页表。

在用影子页表寻址时，影子页表的缺页异常通常会有两种原因：一种是客户机操作系统自己所造成的缺页异常；另外一种是客户机页表和影子页表不一致造成的。上述的两种情况，都会被 KVM 所捕获，然后对产生异常的客户机进行异常分析，并对其做出相应的处理。如果异常是第一种原因，KVM 会直接把异常交给客户机的缺页异常处理机制来进行处理。如果是第二种原因，KVM 会通过客户机页表来同步对应的影子页表。

对于硬件内存虚拟化，是在对客户机内存虚拟地址到客户机内存物理地址映射基础上，引入 EPT 技术，EPT 页表来实现了客户机内存物理地址到宿主机内存物理地址的转换。这两次地址转换都是通过硬件的协助来完成的。同前面的影子页表相比，提高了客户机操作系统的运行性能，影子页表内存虚拟化需要 KVM 为每一个客户机操作系统的进程维护一组影子页表，这在内存上带来了较大的开销，影子页表和客户机页表上的同步也比较繁琐。而硬件内存虚拟化避免了这些缺点，使得客户机有更好的运行性能。

2.2.4 KVM 和 Qemu 设备管理

虚拟机的设备模拟和管理实际上是由用户空间的 Qemu 来实现的，而内核下 KVM 的作用是实现了 I/O 操作的截获。在一个操作系统中，我们可以通过 PIO（端口 IO）和 MMIO（内存映射 IO）两种方式来与硬件设备交互。同时硬件可以发出中断请求，交给操作系统来处理。在 KVM 虚拟机的环境下，KVM 和 Qemu 就必须截获和模拟出 PIO 和 MMIO 请求，模拟虚拟硬件中断。对于 PIO 的截获是由硬件直接提供的，在虚拟机发起 PIO 指令的时候，客户虚拟机会产生 VM Exit，接着硬件会把 VM Exit 的原因写入到 VMCS 结构的 VM_EXIT_REASON 中，这样 KVM 就会模拟 PIO 指令。而 MMIO 的截获是在 MMIO 页的访问致使缺页异常，被 KVM 所截获，然后 MMIO 指令由 x86 模拟器模拟执行的。由于虚拟机的设备模拟是由 Qemu 来实现的，所以设备的 PIO 和 MMIO 访问都会经过 Qemu 这软件层，KVM 通过异步通知和 I/O 指令的模拟来完成对设备的访问。这也是本文选取在 Qemu 这层实现磁盘加解密模块的原因，而且 Qemu 在用户空间内，源代码的修改编译比内核中 KVM 修改编译会更加容易。

2.3 磁盘加解密算法研究

在本文中涉及到对虚拟机磁盘加解密，磁盘加密是保护虚拟机用户数据存储的有效方法，所以不得不谈及磁盘的加解密算法。目前比较常用的加解密算法主要分为下面三类：

1）对称算法

对称加密算法是最早应用的传统加密算法，技术也非常成熟，主要应用于数据的加密。在对数据加解密过程中，采用了同一个密钥，因此加解密方需要协商一个密钥。从而对称加密算法的安全性对密钥的安全性依赖程度很高，所以本文将对磁盘加密的密钥进行加密存储，以保证用户密钥的安全。对称加密算法优点是具有良好的加解密速度、公开加解密算法、加解密效率高，其缺点是密钥管理分配比较复杂。目前常用的对称算法有 DES 算法、AES 算法、Blowfish 算法、RC5 算法、IDEA 算法等等。

本文对虚拟机磁盘加解密过程中采用了 AES 算法，下面将着重介绍一下 AES 算法。

AES 算法是美国联邦政府采用的数据加密标准，具有加密强度高、效率高、灵活性好等特点。它采用的一个迭代分组密码，其分组长度和密钥长度均是可变的，可以采用 128 位、192 位和 256 位的密钥，想对应的迭代次数分别为 10 轮、12 轮、14 轮。如图 2-5 所示：

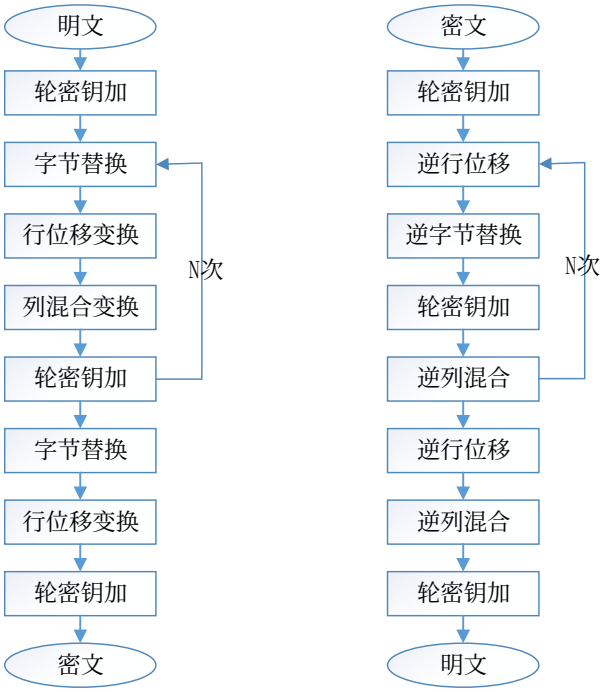


图2-5　AES 加解密框图

在字节替换中，是通过一个 S 盒对加密的分组按字节进行简单的替换。

在行位移变换中，加密的分组分别按不同的偏移量向左循环移动。第一行保持不变，第二行移动 1 个字节，第三行移动 2 个字节，第四行移动 3 个字节。

在列混合变换中，是利用在域（2^8）的特征上对列进行混合替换。

在轮密钥加中，是利用当前加密分组和扩展密钥其中的一部分进行按位异或运算。

2）非对称算法

非对称加密算法需要使用的是一对相互匹配的密钥进行加解密，其中一个向外界公开的密钥称为公开密钥，而另外一个相匹配的密钥称为私有密钥，公开密钥用于对明文的加密，私有密钥用于对密文的解密。在非对称算法加解密过程中，首先解密方生成一对密钥，把其中的一个公开密钥发送给加密方，加密方得到公开密钥后，对需要加密的数据进行加密，然后把加密过的密文发送给解密方，解密方通过与公开密钥相匹配的私有密钥对密文进行解密。显然，如果使用非对称加密算法对数据加解密，加解密双方在通信之前，需要解密方提前生成公开密钥发送给加密方，而自己保存私有密钥由于密文解密。由于非对称加密算法拥有相匹配的两个密钥，所以适合在分布式系统中数据加密使用，密钥管理也比较方便。但是非对称加密算法比较复杂，与前面的对称加密算法相比，其效率比对称加密算法低很多，因此对磁盘这样的大量数据的加解密并不适合用非对称加密算法。目前，常用的非对称加密算法有：RSA 算法、DSA 算法、ECC 算法和 Diffie-Hellman 算法等等。

3）不可逆算法

不可逆加密算法与前面两种对称加密算法和非对称加密算法有所不同，它并不需要密钥进行加密，对需要加密的数据直接由不可逆加密算法转换成密文，并且这些加密后的密文是不能被破解的，只有相同的明文通过相同的不可逆算法才能得到相同的密文。由于不可逆加密算法没有密钥，所以不需要密钥的管理和分配，但加密计算比较复杂，适合用于对少量数据进行加密，在对计算机的口令加密广泛应用。在本文中也采用了 MD5 不可逆加密算法对用户虚拟机密码加密，防止窃取者以及云平台管理人员获得虚拟机用户密码，来获取到用户虚拟机数据信息，增加了用户虚拟机密码破解难道。MD5 算法的特点是：不管是字符串还是文件以及他们的大小如何，都能通过 MD5 算法计算出相同位数的 MD5 值，并且 MD5 值是不可逆的，我们不能通过现有的 MD5 反向计算出原来是什么字符串。唯一的方法是通过枚举法来测试输入的明文，检验明文经过 MD5 算法加密后的 MD5 值是否与获得的 MD5 值相同，如果相同，则说明解密成功。可以看出，想要破解 MD5 值是非常困难的。

2.4 磁盘数据擦除技术

磁盘数据擦除技术就是把计算机上用户信息和使用过的数据信息永久的删除，

使用任何恢复技术都无法恢复得到磁盘上用户数据信息的技术。在 oVirt 云平台中在删除用户虚拟机时，只简单的删除了虚拟机镜像文件，只在文件分配表中标记为了空簇，并没有对数据区中的虚拟机用户信息做任何数据销毁操作，用户虚拟机信息实际还在磁盘数据区中。这样使得用户虚拟机信息可能被他人窃取使用，对用户虚拟机数据造成不安全隐患，为了解决这个问题，所以采用了磁盘数据擦除技术。

在 oVirt 云平台中，虚拟机镜像文件都存放在服务器的硬盘上，所以在了解磁盘数据擦除技术之前，不得不谈及到硬盘数据存储的原理。硬盘的数据结构主要分为 MBR 区（主引导记录区）、DBR 区（系统引导记录区）、FAT 区（文件分配表）、DIR 区（文件目录区）和 DATA 区（数据区）。MBR 区中拥有硬盘的参数信息和引导程序，主要是引导操作系统和检查分区表。DBR 区用于检查文件系统和磁盘参数。FAT 区用于文件的寻址，和在其后的 DIR 区配置确定文件的位置。DIR 区记录了每个文件的起始位置，文件信息等。DATA 区是数据真正存储的地方，在 DATA 区中数据是随机存放的，我们删除文件数据时，实际上并没有改动或销毁数据区里面的内容，增加了我们恢复文件数据的可能性，所以我们需要将不再使用文件数据进行擦除，防止被他人恢复使用。

目前数据擦除主要分为两种不同的方式：硬销毁和软销毁。硬销毁是通过物理或者化学的手段来对存储介质进行破坏，使得数据很难恢复，从根本来对数据进行销毁，但是这种方式不能重复利用存储介质。软销毁是通过软件方式对存储介质上的数据擦除，常用的方法就是对数据区的数据进行多次数据覆写，以到达防止用户数据被窃取的可能性。采用软件擦除数据的方式可以重复利用存储介质，节省开销，并且不用担心虚拟机用户数据被他人窃取，考虑上述原因，在本文的虚拟机磁盘深度擦除模块中也采用的是软件擦除数据的方法。下面将介绍几种存储介质擦除标准：

1）国家保密局 BMB21-2007 标准要求是在对存储介质进行擦除时，需要对存储介质上的数据进行至少 6 次以上的覆盖，使得到达无法恢复数据的地步。

2）美国国防部 5220.22M 安全删除标准要求是在对存储介质进行擦除时，向存储介质的数据进行三遍数据覆写，第一遍用 0xff 覆写，第二遍用 0x77 覆写，第三遍用随机数覆写。从而达到用户数据擦除的目的。

3）美国国家安全局标准要求是在对存储介质进行擦除时，需要对存储介质的数据进行 7 次随机数覆写。

4）Gutman[21]覆写标准要求在对存储介质进行擦除时，需要进行 35 次数据擦除，每次覆写模式也不相同，是最安全的数据擦除标准，但是擦除速度慢，非常

消耗时间。

在理论上，如果要彻底擦除存储介质上的数据，需要重复对存储介质覆写 7 次以上，覆写的次数越多，数据恢复的可能性就越趋近于零。在实际应用中，以上标准都经常被使用到，事实也验证了上面三种标准的有效性。在一般情况下，存储介质经过一遍覆写后，就能够很好地起到对存储介质数据擦除的作用。其它情况下，可以根据不同的数据密级，采用不同的存储介质数据的擦除标准。

2.5 本章小结

本章的内容主要是为了后续章节做基础铺垫，从虚拟化技术说起，概括介绍了软件虚拟化技术和硬件虚拟化技术以及 VMM 三种经典模型。然后，详细介绍了 KVM 和 Qemu 对客户机操作系统调度管理、虚拟机内存管理和虚拟机设备管理虚拟机技术。最后，介绍了磁盘的加解密算法和磁盘数据擦除技术。从下面一章开始，我们在此基础上，将对 VDSM、Qemu、KVM 进行大量源码分析，分析研究 oVirt 云平台安全加固关键技术。

第三章 oVirt 云平台系统研究分析

3.1 oVirt 云平台框架分析

oVirt 云平台是一个基于 KVM 虚拟化技术的开源桌面云管理平台，从整体架构设计上来说, 它采用了 Engine/Node 分离的结构, 从而有利于功能的管理和划分, 方便 oVirt-Node 节点快速部署，如图 3-1 所示：

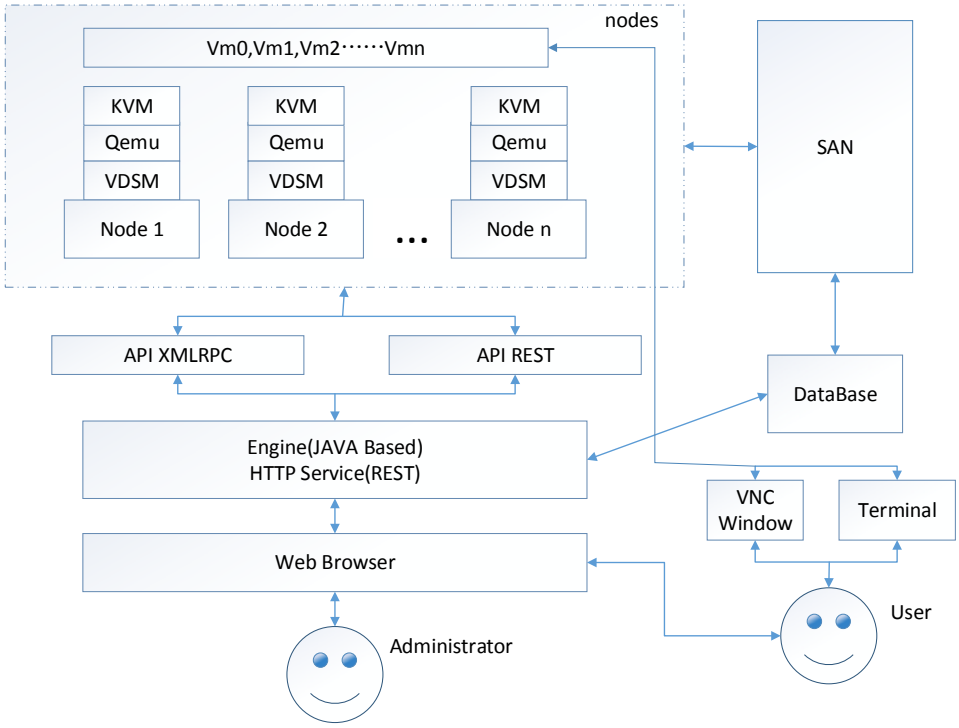


图3-1　　oVirt 云平台结构

3.1.1 oVirt-Engine 分析

oVirt-Engine 在整个系统中充当了管理者的角色，管理系统中所有的 oVirt 宿主机，它也提供了面向管理员、普通用户两个 Web 管理界面，允许系统管理员通过网页创建和部署虚拟机，以及为虚拟机添加或修改相关设备和用户权限，而普通用户只能在获得权限的情况下，并通过 Spice 或者 Vnc 操作属于自己的虚拟机。oVirt-Engine 对外提供管理服务，它拥有自己的数据库，数据库记录了所有虚拟机配置信息与状态信息，oVirt-Node 节点的状态信息，系统网络信息等等。

oVirt-Engine 也提供了另外许多功能：虚拟机生命周期管理；通过 LDAP 身份认证；存储管理-管理存储域（NFS/ISCSI/Local）和虚拟 VM 磁盘；高可用-任意一个 Node 节点崩溃，虚拟机会在其它 Node 重启，不会影响动到系统；动态迁移-

虚拟机在不停机情况下在宿主机之间进行迁移；节能优化-在非高峰时期将虚拟机集中在少数宿主机上运行；镜像管理-基于模板管理，自动管理虚拟机镜像；监控系统中所有虚拟机、主机、网络、存储等等。

3.1.2 oVirt-Node 分析

oVirt-Node 则是开源虚拟化桌面云管理平台 oVirt 的客户端部分，它可以由一个装有 Linux 系统上安装 VDSM（Virtual Desktop Server Manager）组成，也可以是经过定制的 Linux 系统，同时 VDSM 主机处理器必须开启 AMD-V 或 Intel VT 硬件虚拟化技术。oVirt-Node 主要负责功能上的实现，如虚拟机与设备的读写、开启、关闭等功能，也能对磁盘存储进行管理，磁盘存储管理会在下面的小节进行介绍分析，并把这些功能通过 XMLRPC 接口或 REST 接口与 oVirt-Engine 交互，oVirt-Engine 则通过这些接口控制各个 oVirt-Node 上的功能，实现了 Engine/Node 的分离。其主要结构如图 3-2 所示：

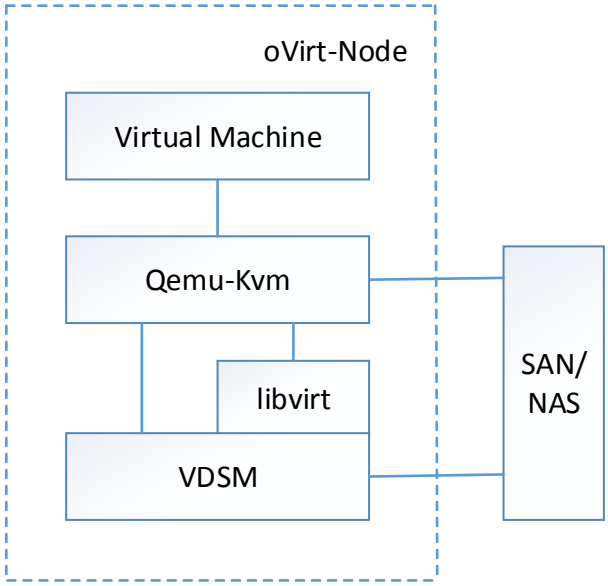


图3-2　oVirt-Node 结构

3.2 oVirt 磁盘存储管理分析

在 oVirt 桌面云管理平台中，通常会有一个 oVirt-Engine 和多个 oVirt-Node 节点。每个 oVirt-Node 节点服务器上都运行了一个 VDSM，通过网络进行互联起来，为了方便管理，oVirt 云平台抽象出数据中心的概念，一个数据中心有一组集群用来运行虚拟机，而虚拟机磁盘镜像文件和虚拟机 ISO 文件存放在存储域中，多个

存储域又组成了一个存储池。由于数据中心的所有 oVirt-Node 节点都拥有对数据中心的存储池的访问权限，因此，VDSM 实现了一个存储池管理角色功能，在一个数据中心中，所有 oVirt-Node 节点会选举出一个 oVirt-Node 节点为管理节点，它有管理虚拟机磁盘镜像、虚拟机模板的功能。存储管理结构如图 3-3 所示：

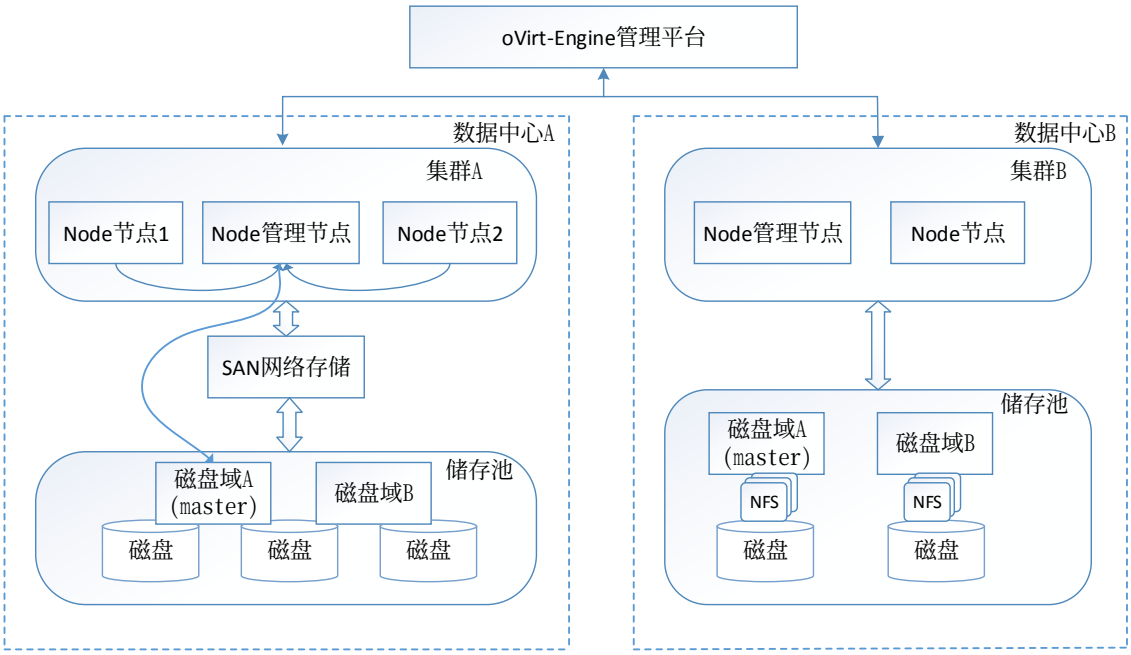


图3-3　　oVirt 存储管理结构

在上图 3-3 中，数据中心是完成 oVirt 功能的实体，它负责管理集群和存储池。在数据中心里，管理员可以创建虚拟机、配置虚拟机、虚拟机模版管理和虚拟机动态迁移，允许一个虚拟机可以在一个集群里面，从一个 Node 节点动态迁移到另外一个 Node 节点上。

存储池是由一组存储域所构成，是为了管理存储域之间的备份、恢复、合并等，在数据中心里，一个存储池抽象了一组存储域给 oVir-Node 访问或 oVirt-Engine 管理，并且在一个存储池里面存储域的类型是相同的，如 NFS 或者 ISCSI。

存储域是 oVirt 云平台中最基本的存储实体，它保存了所有虚拟机镜像文件和虚拟机数据配置信息，虚拟机镜像文件就是虚拟机运行时所操作的文件、驱动和一些设备文件对象，虚拟机配置信息则是保存虚拟机状态、磁盘大小数据信息、内存信息等等。存储域分为了两种类型：文件域和设备域。文件域使用了文件系统来存储数据，主要针对 LOCALFS（Local File System）文件系统和 NFS（Network File System）文件系统。在文件系统的帮助下，文件域具有良好操作虚拟机镜像文件的能力，每一个存储域实际对应于宿主机文件系统中的一个目录。而设备域直接操作原始的虚拟机数据，使用了 Linux 中的逻辑卷管理功能来组织数据，主要针

对 FCoE 、iSCSI 等块设备。

3.3 虚拟机镜像文件加密关键技术分析

虚拟机镜像文件是在 oVirt 云平台环境中十分重要的文件，它可以理解为虚拟机的磁盘，包含了虚拟机的操作系统、驱动、用户数据和虚拟机配置等信息。在 oVirt 云平台环境中, 虚拟机镜像文件是以明文的方式存放在 oVirt-Node 服务器上，这使黑客很容易获取到用户虚拟机里的个人数据信息，存在很大的安全隐患。以下用实验展示用户虚拟机磁盘数据不安全性，并将通过 Qemu-Kvm 源码分析磁盘虚拟化的实现原理，来找到虚拟机镜像文件安全加密的合理位置。

3.3.1 虚拟机镜像文件安全脆弱性实验

在 oVirt 云平台下，虚拟机磁盘对应到宿主机上只是一个文件夹，然而这些镜像文件都是以明文的方式存放在 oVirt-Node 节点上。

```
[root@node1 ~]# cd /home/ovirt/images/
[root@node1 images]# ll
total 112
drwxr-xr-x. 2 vdsm kvm 4096 Jan 23 23:15 733ce1ff-5e38-4f5c-b373-d0537cc8361e
drwxr-xr-x. 2 vdsm kvm 4096 Dec  9 03:47 5556e703-ecfe-4990-9254-2b9b33f02e84
drwxr-xr-x. 2 vdsm kvm 4096 Jan  5 04:10 5a9f54e9-0346-4468-bade-6bc6b1b9f337
drwxr-xr-x. 2 vdsm kvm 4096 Dec  9 03:15 5dda67b4-b208-4f19-9f7e-b17a5bb8a6ca
drwxr-xr-x. 2 vdsm kvm 4096 Jan  5 02:00 b6ff899d-f309-4461-9f32-14f1289e6ef0
drwxr-xr-x. 2 vdsm kvm 4096 Jan  5 01:48 e2bcb16d-f9a9-4e16-a4e6-a9b7d289260a
drwxr-xr-x. 2 vdsm kvm 4096 Dec 11 22:54 f250ca4f-e685-4b75-84b7-19b313c57b4a
drwxr-xr-x. 2 vdsm kvm 4096 Dec  9 02:13 f4ee4baf-b9c6-43ca-981f-a86500886f94
```

从上面可以看出，虚拟机磁盘分别与 oVirt-node 节点上的文件夹一一对应，下面我们就用 733ce1ff-5e38-4f5c-b373-d0537cc8361e 虚拟机为例子来做一个简单的安全检测实验，验证其安全性。

首先，我们在 oVirt-Engine 管理界面用 Spice 连接到 Test_vm1 虚拟机，它对应的 oVirt-Node 节点上 733ce1ff-5e38-4f5c-b373-d0537cc8361e 文件夹, 连接成功后，在虚拟机中的桌面上创建一个名为 test 的纯文本文件，文本本件的内容是“This is a test！”。接着，关闭虚拟机。如图 3-4 所示:

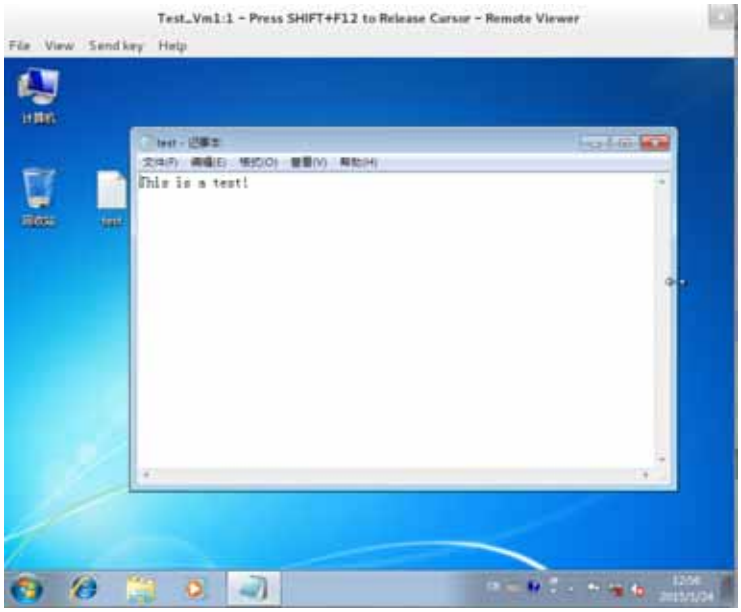


图3-4 Test_vm1 创建文本文件

然后，我们用命令行的方式，进入 733ce1ff-5e38-4f5c-b373-d0537cc8361e 文件夹，会发现里面有三个文件：其中 e7f7c093-b5b9-4754-9a5e-72bda10e48a8 是虚拟机磁盘数据文件，其它俩个文件是虚拟机镜像配置信息。

```
[root@node1 images]# cd 733ce1ff-5e38-4f5c-b373-d0537cc8361e/
[root@node1 733ce1ff-5e38-4f5c-b373-d0537cc8361e]# ls
e7f7c093-b5b9-4754-9a5e-72bda10e48a8  e 7f7c093-b5b9-4754-9a5e-72bda10e48a8.lease
e7f7c093-b5b9-4754-9a5e-72bda10e48a8.meta
```

接着，我们用 qemu 命令启动该虚拟机：

```
[root@localhost super]# qemu-system-x86_64 -m 2048 -smp 2 -boot order=cd -hda
/home/733ce1ff-5e38-4f5c-b373-d0537cc8361e/e7f7c093-b5b9-4754-9a5e-72bda10e48a8
VNC server running on `127.0.0.1:5900'
```

通过用 vncviewer 命令：vncviewer :5900，成功把虚拟机启动起来，并查看到了用户保存的个人数据信息。接着把 733ce1ff-5e38-4f5c-b373-d0537cc8361e 虚拟机文件夹拷贝到另外一台与 oVirt 云平台之外装有 Qemu-Kvm 的 linux 系统主机上，通过以上的相同方式也成功启动了虚拟机，并查看到了用户数据，如图 3-5 所示：

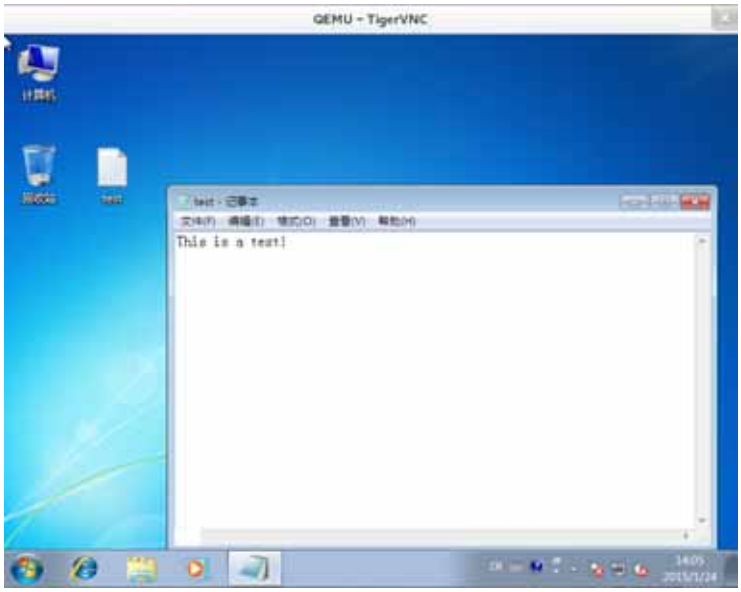


图3-5　启动虚拟机镜像查看文件

通过以上实验可得，在 oVirt 云平台环境下，用户虚拟机磁盘不是安全的，只要能够获取到虚拟机文件，就能通过 qemu 轻易的获取到用户数据。在 oVirt 平台实际应用情况下，黑客如果获取到了虚拟机镜像文件，不经过破解就能窃取到用户数据，我们也无法保证 oVirt 云平台管理员不会通过以上方法来窃取用户数据。所以，我们需要对虚拟机镜像文件加密来保护用户虚拟机数据信息，来保证用户虚拟机使用安全性。

3.3.2 Qemu-Kvm 虚拟机运行机制分析

由于需要找出虚拟机镜像加解密合理位置，很有必要分析虚拟机的启动运行情况，有利于帮助我们选取加解密切入点。由于这方面的技术资料很少，大部分内容都需要我们对 Qemu-Kvm 源码分析研究得出。

3.3.2.1 Qemu 核心流程

本文分析的 Qemu 版本为 qemu-1.6.1，在 Qemu-Kvm 中，Qemu 主要完成了虚拟机设备的模拟和通过 KVM 提供的 LibKVM 应用接口，使用 ioctl 系统调用与内核的 KVM 通信，创建和运行虚拟机。 Qemu 的核心流程主要分为了 5 个阶段，如图 3-6 所示：

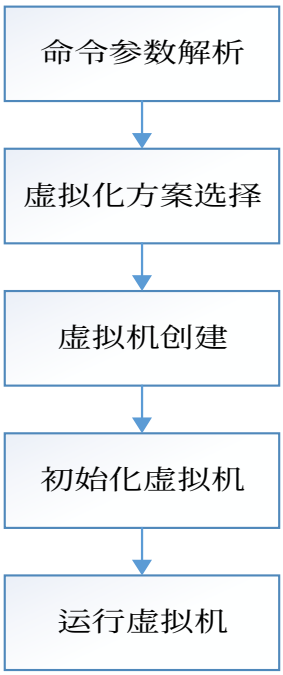


图3-6 Qemu 核心流程

1）命令参数解析

当启动 qemu-system-x86_64 应用程序，该程序入口为 vl.c 中的 main 函数，在 main 函数中的第一阶段主要对命令参数进行解析，比如是否开启 NUMA，创建 VCPU 数量，虚拟机内存资源分配数量等等。

2）虚拟化方案选择

通过 configure_accelerator()函数，选择使用哪一种虚拟化解决方案。

```
accel_list[] = {
    { "tcg", "tcg", tcg_available, tcg_init, &tcg_allowed },
    { "xen", "Xen", xen_available, xen_init, &xen_allowed },
    { "kvm", "KVM", kvm_available, kvm_init, &kvm_allowed },
    { "qtest", "QTest", qtest_available, qtest_init, &qtest_allowed },
};
```

accel_list[]数组声明了 Qemu 使用的系统模拟方案。“tcg”模式是不使用任何硬件虚拟化辅助方式，采用基于二进制指令动态翻译的方式，将目标平台的指令代码通过 TCG 的模块翻译为本机可以执行的指令。“xen”、“kvm”分别为两种主流的开源虚拟化解决方案。本文主要针对 KVM 这种硬件辅助的虚拟化解决方案。

3）虚拟机创建

通过上一阶段，选择了 KVM 虚拟化方案，接着进入到 kvm-all.c 文件下 kvm_init()函数，首先打开/dev/kvm，获取到 KVM 设备文件描述符 fd，便可以利

用 fd 调用 ioctl 向 KVM 设备发送命令。接着通过 KVM_GETAPI_VERSION 进行版本验证，最后通过 KVM_CREATE_VM 创建一个虚拟机对象，返回了虚拟机描述符 vmfd，应用程序可以利用 vmfd 描述符通过 ioctl 发送命令给虚拟机。

```
s->fd=qemu_open("/dev/kvm",O_RDWR);
s->vmfd=kvm_ioctl(s,KVM_CREATE_VM,0);
```

4）初始化虚拟机

通过前面阶段加载命令的参数解析和相关系统的初始化，找到对应的 machine 类型进行第四阶段的初始化：

```
QEMUMachineInitArgs args = { .ram_size = ram_size,
                             .boot_device = boot_order,
                             .kernel_filename = kernel_filename,
                             .kernel_cmdline = kernel_cmdline,
                             .initrd_filename = initrd_filename,
                             .cpu_model = cpu_model };
 machine->init(&args);
```

其中的参数包括从命令传入解析后得到的，ram 的大小，内核镜像文件名，内核启动参数, initramdisk 文件名, cpu 模式等等。我们使用系统默认类型的 machine，则 init 函数为 pc_init_pci()。经过一系列调用，完成了虚拟机初始化操作。

5）虚拟机运行

虚拟机运行真正的执行体是 Qemu 进程创建的一系列 POSIX 线程，而线程执行函数为 qemu_kvm_cpu_thread_fn()。在 qemu_kvm_cpu_thread_fn()函数中通过 kvm_init_vcpu()创建了 vcpu 描述符/vcpufd。并进入了 while(1)的循环，反复调用 kvm_cpu_exec()函数。在 kvm_cpu_exec()函数中又是 do()while(ret==0)的循环体，该循环体中主要通过 KVM_RUN 启动虚拟机运行，从此处进入了 KVM 的内核处理阶段，并等待返回结果，同时根据返回的原因进行相关处理，最后将处理结果返回。因为整个执行体在上述函数中也是循环中，所以后续又会进入到该函数的处理中，而整个虚拟机 cpu 的处理就是在这个循环中不断的进行。

3.3.2.2 KVM 核心流程

本节主要分析了 KVM 关于虚拟机运行的核心流程，使用的 linux 内核版本是 linux3.14.8。KVM 内核模块主要流程分为 4 阶段：初始化、虚拟机创建、VCPU 创建、VCPU 运行。

1）初始化

在这个阶段，KVM 内核主要做了一些初始化准备工作，通过 module_init 宏对

平台相关模块初始化，接着对 kvm 核心初始化：初始化全局变量 kvm_x86_ops、初始化 MMU 等数据结构、初始化定时器架构等等。

2）虚拟机创建

在 Qemu 核心流程中，分析了通过 ioctl(KVM_CREATE_VM)系统调用进入到 KVM 内核来创建虚拟机的。最终在 KVM 内核调用了 kvm_create_vm 函数来创建虚拟机, 它主要完成了虚拟机结构体的创建、KVM 的 MMU 操作结构的安装、KVM 的 IO 总线初始化、事件通道初始化等操作。如下图 3-7 所示：

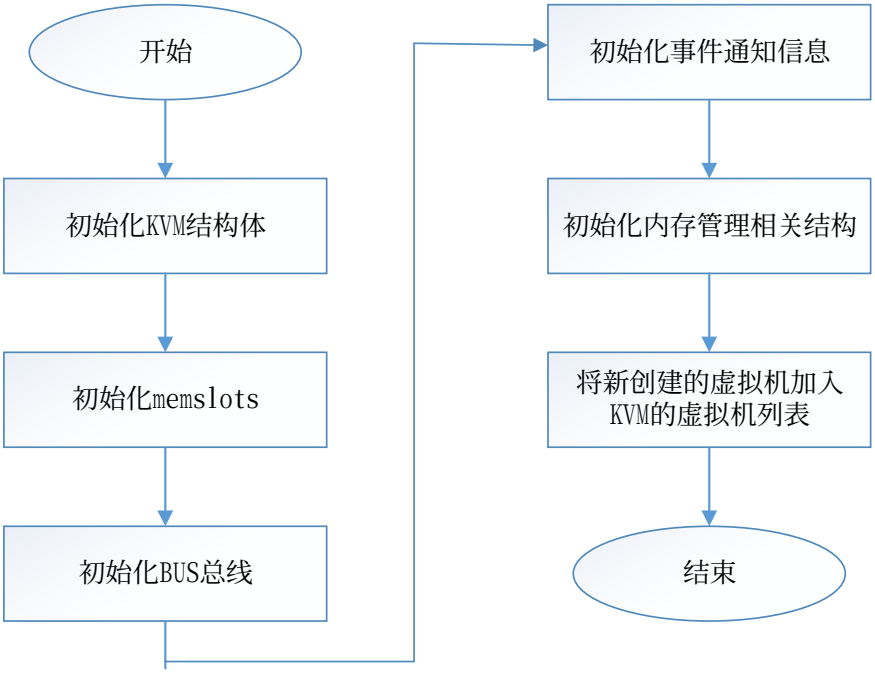


图3-7　虚拟机创建流程

3）VCPU 创建

在上一阶段完成了虚拟机创建后,便可以为虚拟机创建 VCPU。创建 VCPU 具体实现的内核函数为 kvm_dev_ioctl_create_vcpu()，它主要完成了 VCPU 描述符的创建，即 kvm_vcpu 结构体。主要过程如下图所示：

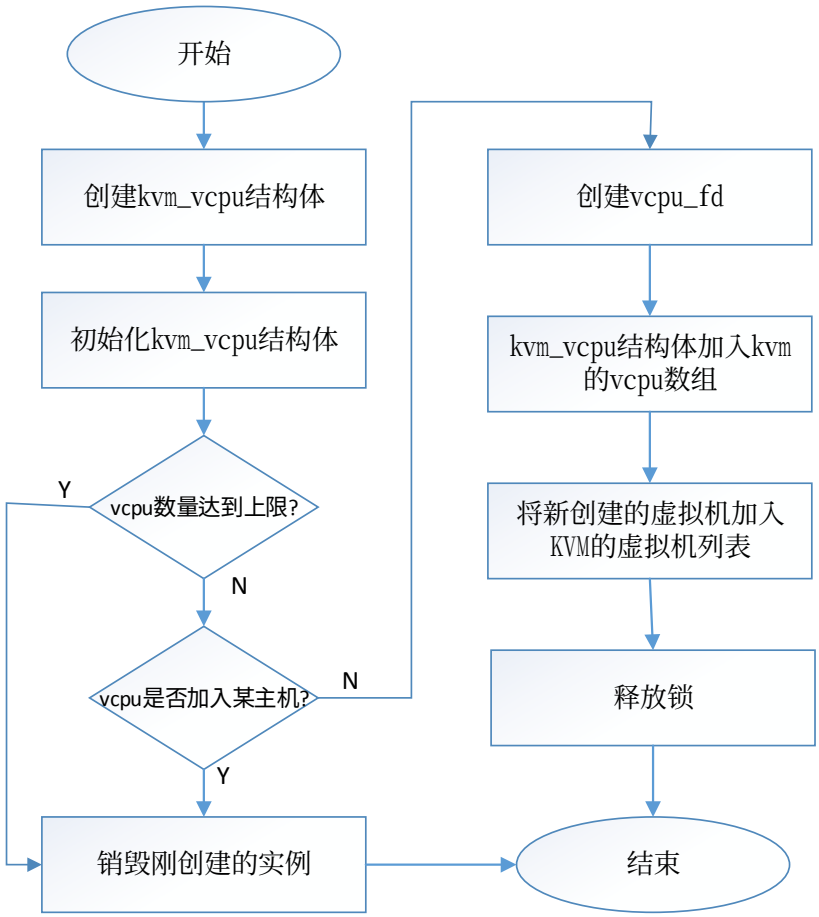


图3-8　VCPU 创建过程

如上图 3-8 所示, kvm_vcpu 结构体是由 kvm_arch_vcpu_create()函数创建所得，直接调用了 kvm_x86_ops 中的 create_cpu 方法，主要对相关寄存器和 CUPID 进行初始化，调用 kvm_arch_vcpu_setup()初始化 kvm_vcpu 结构。接着判断 VCPU 数量是否达到了 VCPU 数量的上限值 KVM_MAX_VCPU，如果是，则销毁刚创建的实例；否则，则继续判断当前的 VCPU 是否已经加入了某一个已有的 KVM 主机，如果是，

则销毁刚创建的实例。接着会创建当前 VCPU 所对应的文件描述符 vpcu_fd，并且将 kvm_vcpu 添加到 KVM 的 vcpu 数组中，添加操作使用了两个原子操作，不会因为中途中断、进程切换的方式导致添加出错。最后，释放所有的内核锁，完成了 VCPU 的创建操作。

4）VCPU 运行

在上面阶段, 虚拟机和 VCPU 都创建并初始化后, 就可以调度 VCPU 运行了。Qemu 可以通过 kvm_vcpu_ioctl(cpu,KVM_RUN,0)陷入内核，是虚拟机运行。接着调用_vcpu_run()进入循环后，调用 vcpu_enter_guest()进入客户模式，并执行客户机

系统的相关指令，如果客户机系统向内核申请资源请求，如果是 IO 请求，则提交给用户模式下的 Qemu 处理，非 IO 请求则将处理结果反馈给客户模式。具体见图 3-9 所示：

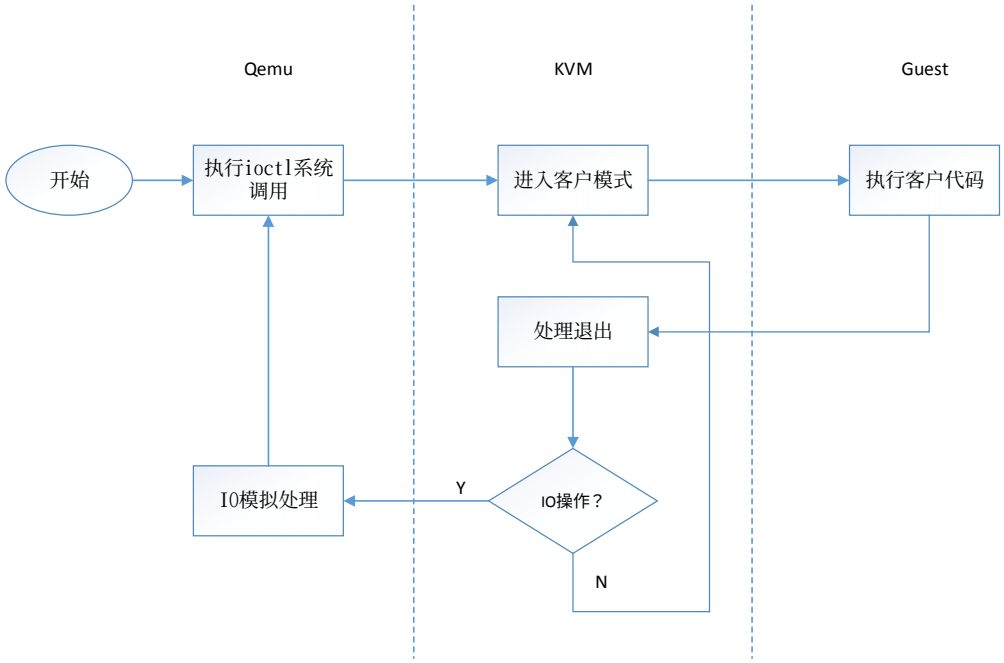


图3-9　VCPU 运行过程

3.3.3 虚拟机磁盘读/写数据研究与分析

在本节中，我们将通过对 Qemu1.6.1 源码调试分析，总结出虚拟机读写磁盘数据的流程，从而在这过程中，找到虚拟机磁盘加解密合理的切入点。在上一小节中分析了虚拟机运行机制，运行一个客户机操作系统包括了：执行客户机代码、处理 timer（定时器）、处理 I/O 操作以及响应管理器的命令。

在宿主机中，为了一个磁盘 I/O 操作执行花费很长时间才能完成时不至于终止对客户机的执行，Qemu 采用的是一种混合架构，把事件和线程组合在一起。Qemu 也是作为一个普通的进程运行在宿主机中，Qemu 进程又包括了多个线程，因此，下面我们通过对 Qemu 源码分析，以线程模型为线索来分析虚拟机磁盘读写数据流程，如图 3-10 所示：

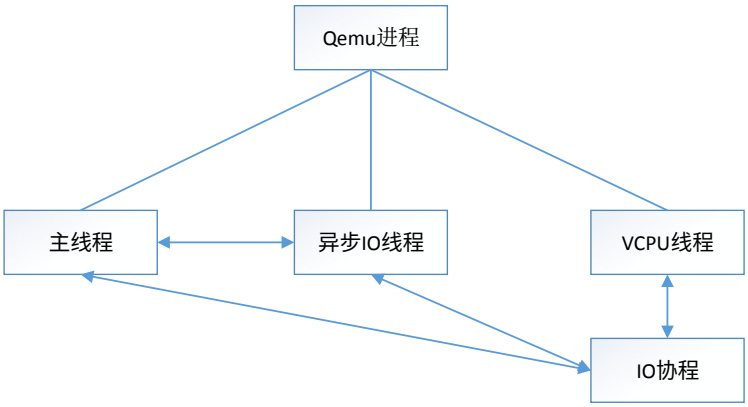


图3-10　qemu 线程模型

从图 3-10 中，我们可以看出 Qemu 进程包含了四个线程：主线程、VCPU 线程、IO 协程和异步 IO 线程，下面将分别对它们进行分析研究。

1）主线程

主线程函数入口是 main_loop()函数，接着进入主要事件循环 main_loop_wait()函数，主要完成了三件事情：其一是执行 select 操作，查询所有文件描述符有无可读写操作，文件描述符充当了一个非常关键的角色，主要的 IO 描述符包括 block io、signalfd、eventfd 和 socket，其中 block io 是虚拟机相关的 io，为了保证高性能，主要采用的是异步 io，signalfd 是 Qemu 的时钟模拟利用 linux kernel 的 signalfd，定期产生 SIGALRM 信号，eventfd 主要用于 Qemu 和 KVM 之间的相互通信，比如 Qemu 的模拟设备向 KVM 发送一个模拟中断，KVM 向 Qemu 报告客户机各种状态；其二是执行定时器回调函数；其三执行下半部（BH）回调函数。当一个文件描述符准好了、一个定时器过期或者是一个 BH 被调度时，事件循环就会调用一个回调函数来处理这些事件。

2）VCPU 线程

VCPU 线程主要是执行客户机代码，和主线程不能同时运行，通过一个全局互斥锁来实现，其它详细介绍见上一小节。

3）I/O 协程

Qemu 中的 I/O 协程主要是基于 setcontext 函数族以及函数之间的跳转函数 siglongjmp 和 sigsetjmp 来实现的，它的作用是向内核提交 I/O 操作。

4）异步 I/O 线程

在异步 I/O 线程中，它并不进行真正的读写操作，只是注册提交读写任务，Qemu 会创建 aio_thread 专门的线程对虚拟机镜像文件进行读写操作，aio_thread 线程会从任务队列中获取需要读写的操作，接着根据注册的相对应的磁盘处理函数来对数据进行处理，把数据写入虚拟机镜像文件或从虚拟机镜像文件把数据读

取到 buffer 中并返回，如图 3-11 所示：

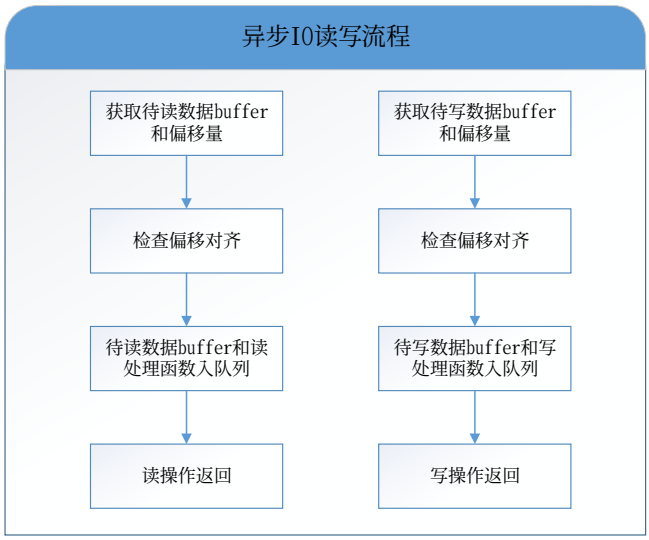


图3-11　异步 IO 读写流程

根据上面的源码分析，我们总结了从客户机到 Qemu 的 I/O 完成的总流程，以磁盘写过程为例，读过程相似。客户机到 Qemu 的 I/O 执行流程如图 3-12 所示：

图3-12　客户机到 Qemu 的 I/O 执行总流程

综上，我们可以提出一种对 oVirt 云平台虚拟机镜像文件加解密的解决方案，

Qemu 的磁盘数据 I/O 操作采用的是异步 IO 处理，使用协程机制实现。要完成对磁盘数据进行加解密，加密函数被调用时必须在完成数据写操作之前，解密函数被调用时需要在完成数据读操作之后。经过不断地实验研究，找到了 raw 格式磁盘加解密切入点为 raw_co_ready 和 raw_co_writev 函数，经过磁盘加解密起到了保护虚拟机用户磁盘数据的目的，至此，我们已经明确了虚拟机磁盘加解密的关键技术，具体详细实现过程将在后续章节介绍。

3.4 虚拟机镜像深度擦除关键技术分析

在 oVirt 云平台中，当用户不再使用自己虚拟机的时候，我们需要通过 oVirt 管理界面把虚拟机删除掉，同时虚拟机镜像文件也被删除了。实际上，通过这种方法删除虚拟机，被删除的虚拟机镜像用户数据其实还在硬盘上，只是修改了标记允许被新数据覆盖，这些删除的数据是可以通过数据恢复软件恢复回来的。下面将用实验来展示 oVirt 平台删除虚拟机后，用户虚拟机数据的不安全性，并通过分析源码找到 oVirt 平台深度擦除虚拟机合理方案。

3.4.1 oVirt 平台删除虚拟机安全脆弱性实验

前面在虚拟机镜像文件安全实验中，对虚拟机的镜像文件怎么存放在 oVirt-Node 节点, 本节不再阐述。下面我们以 733ce1ff-5e38-4f5c-b373-d0537cc8361e 虚拟机为例子来做一个简单的安全检测实验，验证通过 oVirt 管理界面删除虚拟机以后，虚拟机数据是否能在其磁盘上查看到。

首先，我们用 df 命令查看主机文件系统磁盘情况，找到虚拟机镜像文件存放在系统磁盘，接着用 debugfs /dev/mapper/vg_node1-lv_home 命令，进入 debugfs 模式下。用 debugfs 指令找到虚拟机镜像文件。

```
debugfs:  cd 733ce1ff-5e38-4f5c-b373-d0537cc8361e
debugfs:  ls
13107290  (12) .    12582922  (12) ..
13107295  (44) e7f7c093-b5b9-4754-9a5e-72bda10e48a8
13107294  (56) e7f7c093-b5b9-4754-9a5e-72bda10e48a8.lease
13107293  (3972) e7f7c093-b5b9-4754-9a5e-72bda10e48a8.meta
```

虚拟机镜像数据文件为 e7f7c093-b5b9-4754-9a5e-72bda10e48a8，用 blocks 命令查看该文件所占有的数据块号，列出虚拟机镜像数据文件占有的所有数据块号，接着任选一个数据块，通过 bd 命令显示数据块里的内容，本实验选取的数据块 1996930。如下图 3-13 所示：

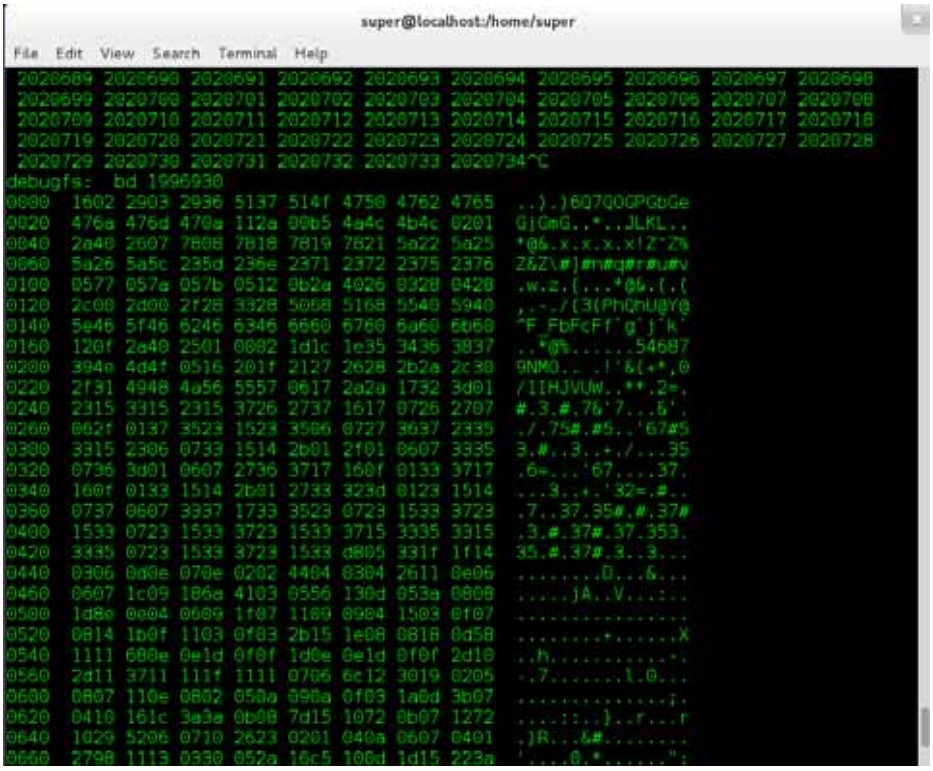


图3-13 虚拟机删除前对应磁盘块内容

然后, 连接到 oVirt 管理员界面, 删除掉 733ce1ff-5e38-4f5c-b373-d0537cc8361e 所对应的虚拟机，再查看磁盘文件，发现 733ce1ff-5e38-4f5c-b373-d0537cc8361e 文件夹也被删除了。这时，再重复上面查看磁盘块内容的方法，通过 bd 1996930 命令查看磁盘块内容。

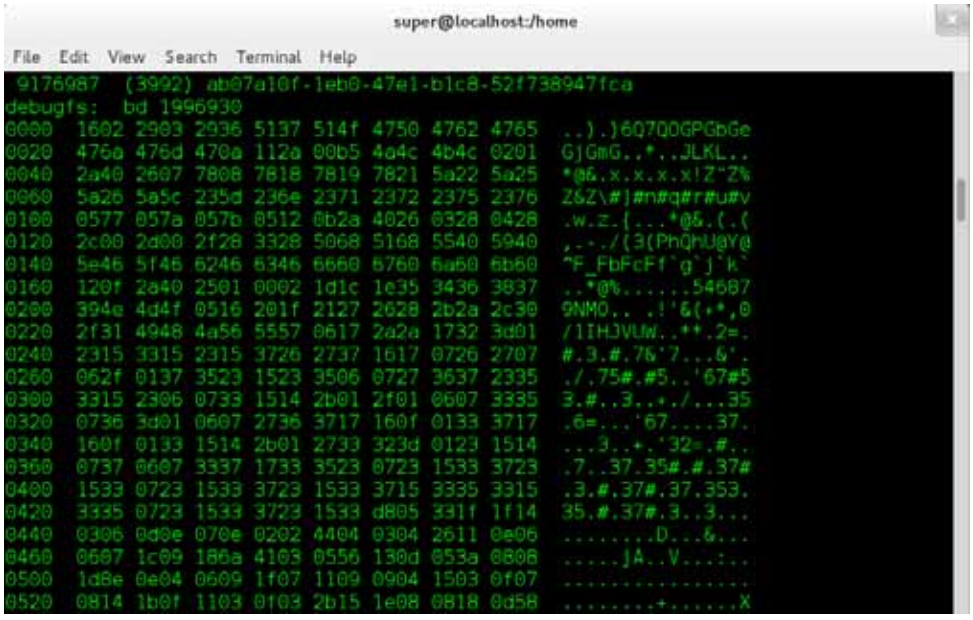


图3-14 虚拟机删除后对应磁盘块内容

通过上面两张图，发现在虚拟机删除前后，虚拟机镜像文件对应的磁盘块上数据内容完全相同，实验证明了，虚拟机镜像文件虽然被删除掉了，实际上虚拟机用户数据还在磁盘上，并可用大部分恢复软件恢复磁盘上的用户虚拟机数据。说明 oVirt 管理平台仅仅只删除虚拟机镜像文件的方法，是存在一定安全隐患的，所以下面将研究分析虚拟机深度擦除的方案，保障虚拟机在删除后，使得用户虚拟机数据难以恢复。

3.4.2 虚拟机磁盘深度擦除研究与分析

在 oVirt 云平台中，oVirt-Engine 云管理平台删除不再使用的虚拟机时，只是简单的把用户虚拟机镜像文件删掉，并没有对镜像文件进行数据擦除处理。在上面的实验也验证了用户虚拟机存在被他人恢复使用的隐患，下面将讨论虚拟机磁盘深度擦除技术。

oVirt 云平台中，虚拟机磁盘删除实现与虚拟机磁盘文件读写相比，并不需要 Qemu 和 KVM 的介入，直接在 VDSM 层进行虚拟机删除操作，相对比较简单。oVirt-Engine 管理平台删除虚拟机时,通过 XMLRPC 接口把虚拟机删除命令以及虚拟机参数传递给在前面小节所述的 Node 管理节点，其中虚拟机参数包括了虚拟机所属存储池 spUUID、虚拟机所属磁盘域 sdUUID、虚拟机唯一标识符 UUID 和删除控制信息。Node 管理节点在接受到虚拟机删除命令后，根据上面的参数查找到虚拟机镜像文件的位置，经过一系列调用，最终调用 overwrite_selector 函数接口删除了虚拟机镜像文件及虚拟机配置信息。

根据上面分析，我们可以提出一种对 oVirt 云平台虚拟机磁盘深度擦除的解决方案，在 oVirt 原本删除虚拟机线路的基础上，增加虚拟机磁盘深度擦除模块，并添加一条与其并行的虚拟机深度擦除的线路来调用该模块。这样可以增加一种删除虚拟机方式供用户选择，对于一些没用重要数据的虚拟机就用 oVirt 云平台原本的删除方式，如果虚拟机有保密的重要数据，在删除虚拟机时候，便可以勾选增加的深度擦除并选择想要的存储介质擦除标准，虚拟机便采用该擦除标准对虚拟机进行深度擦除，起到防止他人通过恢复技术窃取到用户虚拟机中重要数据的目的。至此，我们已经明确了虚拟机磁盘深度擦除的关键技术，具体详细实现过程将在后续章节介绍。

3.5. USB 设备实时管控关键技术分析

3.5.1 USB 设备重定向研究与分析

在 oVirt 云平台中，虚拟机桌面对 USB 设备的访问是一个较为复杂的过程，通过对 VDSM、Qemu、Spice 等开源代码的研究与分析，得到如下流程：

oVirt 云平台用户通过一台 PC 远程登录到 Web 界面，启动一个虚拟机，虚拟机启动过程前面小结有细致介绍。虚拟机启动完后，使用 Spice 工具远程登录虚拟机桌面，用户 PC 机上的 USB 设备，通过 Spice 协议重定向到远程的虚拟桌面。这时 oVirt-Node 节点上的 Spice 服务器将通知 Qemu 设备模拟器。这样一个 USB 设备就被成功的挂载到了虚拟机上。USB 设备重定向如图 3-13 所示：

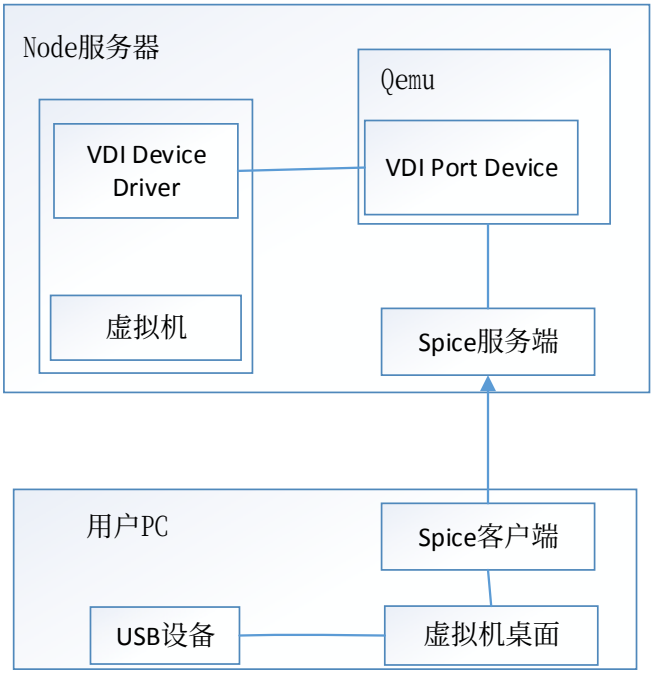


图3-15　USB 设备重定向

从上图 3-13 看出，USB 设备最终会通过 Qemu 虚拟机模拟器，在 Qemu 中，每个 USB 设备对应了一个 USBDevice 数据结构，它定义了对该物理设备访问操作的方法接口。当 Spice 协议把客户远程 PC 的 USB 设备重定向到 Node 服务器后，Qemu 检查该设备的合法性，检查通过后，将为其分配 USBDevice 数据结果，完成模拟工作。

3.5.2 USB 设备实时管控研究与分析

根据上面分析可以看出，用户虚拟机的 USB 设备并没有任何控制，考虑到企业私有云对企业数据的保护要求，USB 设备不加管控会给企业带来安全威胁，一

方面企业员工可以随意的通过 USB 设备拷贝数据，可能会造成公司数据泄露；另一方面，一旦用户账户泄露，攻击者就能随意进入虚拟桌面拷贝数据。所以，USB 设备实时管控是很有必要的。

通过上面的分析，我们可以提出关于 oVirt 云平台 USB 设备实时管控的解决方案。由于 USB 设备都会拥有唯一的一对 PID 和 VID，用户 PC 机上插入的 USB 设备重定向到服务器虚拟机上最终是通过 Qemu 来完成的, 我们可以在 Qemu 中通过 USB 设备的 PID 和 VID 标识符对 USB 设备进行管理限制。增加 USB 设备监控模块，对 USB 设备实时监控，只让云平台管理员允许插入的 USB 设备重定向对应的虚拟机中，USB 设备在没获得权限的情况下，用户 PC 机上插入了 USB 设备，该 USB 设备也无法重定向到虚拟机中。并且在云平台管理员临时取消其中一个 USB 设备的插入某一虚拟机权限时, 对应的虚拟机会实时的把该 USB 设备卸载掉，从而达到对 USB 设备实时监控的目的。至此，我们已经明确了 USB 设备实时管控的关键技术，具体详细实施过程将在后续章节介绍。

3.6 本章小结

本章主要对 oVirt 云平台系统进行了研究分析，分析了 oVirt 云平台框架和磁盘存储管理技术，然后以 oVirt 云平台安全加固为主线，分别对虚拟机磁盘加密、虚拟机深度擦除、USB 设备实时管控三面关键技术分析，并通过实验证明了现有 oVirt 云平台在这三方面的安全隐患。最后，通过分析研究，分别得出了安全加固的解决方案。这些关键技术分析研究设计到大量的 VDSM、Qemu、KVM 内核级的源码分析，对这些关键技术的分析和研究是实现本系统安全加固的重大难点。

第四章 oVirt 云平台安全加固方案总体设计

上一章我们对云平台系统研究进行了详细的分析，并进行了安全性的相关实验。通过实验证明了 oVirt 平台中用户虚拟机数据存在一定的安全隐患，同时通过分析得出了虚拟机磁盘加密、深度擦除和 USB 实时管控三方面的安全加固关键技术。本章将根据第三章的研究分析成果，设计一套在 oVirt 云平台环境下的安全加固方案。

4.1 方案设计需求概述

4.1.1 功能需求

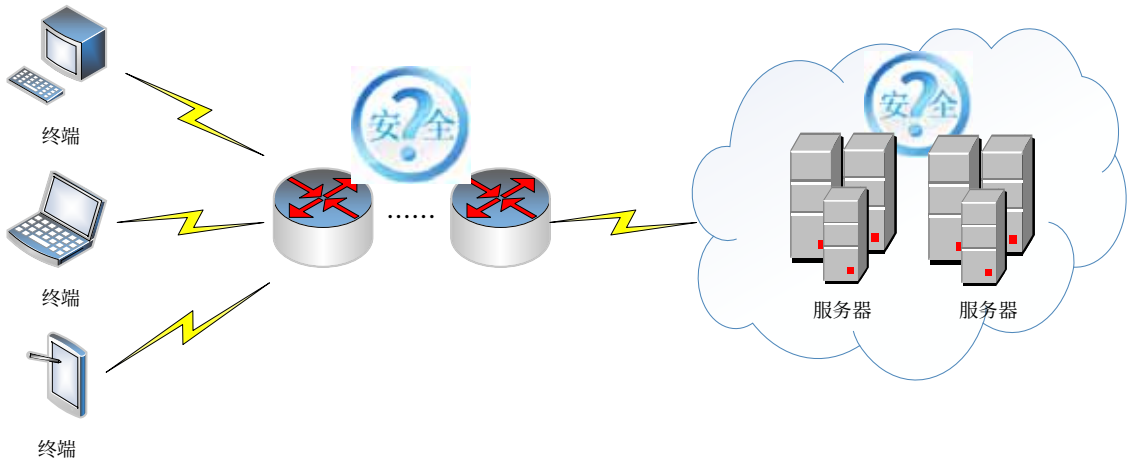


图4-1 oVirt 云平台网络模型

如图 4-1 所示，在 oVirt 云平台环境下，用户使用 Spice 或者 VNC 工具，通过网络连接到云平台服务器虚拟机上，来操作属于自己的虚拟机。目前，在 oVirt 环境，用户的虚拟机镜像文件是以明文的方式存放在服务器上，在第三章中，已通过实验证明了用户虚拟机存在很大的安全隐患，一旦服务器因某些漏洞被黑客攻入，用户虚拟机数据就暴露无遗，这些数据的泄露，可能将会给个人以及企业带来严重的后果。与此同时，在实际运用环境中，我们也不能排除企业内部人员利用移动设备盗取公司内部数据，因此企业需要对企业内部 USB 设备使用情况进行实时管控。为了让用户和企业能够放心使用云平台上的虚拟机，我们需要保证 oVirt 云平台的安全性。有必要设计一套在 oVirt 云平台环境下的安全加固方案，以确保在一定程度上增强企业云办公环境的安全性。

为了解决目前 oVirt 云平台的上述安全隐患，本文设计并实现了基于 oVirt 云平台的安全加固方案，包括了虚拟机磁盘加密、虚拟机磁盘深度擦除、USB 实时管控三大主要模块。oVirt 云平台经过安全加固，用户虚拟机数据是以密文的形式存放服务器上，即使窃取者得到了虚拟机镜像文件，在没有虚拟机密钥和加解密环境下，很难盗窃到用户数据；通过深度擦除虚拟机后，虚拟机数据也被复写，通过一般恢复软件难以恢复原来数据；企业也能通过 USB 实时管控来控制企业员工 USB 使用情况。这样在很大程度上，保障了用户和企业数据的安全性。

4.1.2 运行环境

oVirt 云平台一般需要部署一台 oVirt-Engine 和多台 oVirt-Node 节点（至少一台），本文其运行环境如表 4-1 和表 4-2 所示：

1）硬件环境

表 4-1 oVirt 云平台部署硬件环境

oVirt-Engine	CPU : Inter(R) Core(TM) i7-4770　3.40HZ　4 核 内存：16G DDR3 硬盘：SATA 1T
oVirt-Node1	CPU : Inter(R) Core(TM) i7-4770　3.40HZ　4 核 内存：16G DDR3 硬盘：SATA 1T

2）软件环境

表 4-2 oVirt 云平台部署环境

oVirt-Engine	Fedoral 19 操作系统、oVirt-Engine3.4、Linux3.14.8、Vdsm4.14.6、virt-viewer0.56、Qemu1.6.1、Spice-gtk0.23、libvirt1.1.3
oVirt-Node1	Fedoral 19 操作系统、Windows 7 操作系统、Linux3.14.8、Vdsm4.14.6、Qemu1.6.1、Spice-gtk0.23、libvirt1.1.3

4.2 方案设计原则和思路

为了保证 oVirt 云平台环境下用户和企业数据的安全，在第三章中，通过对系统的关键技术分析和实验验证，明确了虚拟机磁盘加密、虚拟机擦盘深度擦除、

USB 设备实时管控三方面安全加固的切入点。下面在阐述 oVirt 云平台安全加固总体框架之前, 本小节先介绍一下 oVirt 云平台安全加固的方案设计原则及设计思路。

4.2.1 设计原则

在 oVirt 云平台安全加固方案的设计和对 oVirt 云平台的安全性加以改进的过程中，遵循以下四点设计原则：

1）不影响系统正常使用

本方案设计的目的是为了对 oVirt 云平台的安全加固, 在对虚拟机磁盘加密后，不会影响虚拟机的正常运行、关闭、睡眠等；虚拟机深度擦除后，不会影响到系统的其它功能；USB 设备实时管控, 保障允许用户插入的 USB 设备能够正常使用，否则不能使用。

2）降低对系统性能的损耗

对 oVirt 云平台安全加固的同时，我们需要关注对 oVirt 云平台相关源码修改过后，是否会影响 oVirt 系统本身的性能。如果有影响，我们应该尽量降低虚拟机磁盘加解密及擦除和 USB 实时管控对系统运行效率的影响。如果只一味地去追求安全性，使其 oVirt 运行效率低下，将适得其反。所以，我们在设计过程中需要重视对系统性能的影响。

3）磁盘加解密开放性

对 oVirt 云平台中虚拟机磁盘加解密时，应设计开放性的磁盘加解密接口，用户可以使用不同的加解密方法对虚拟机进行加解密。

4）新增安全加固模块使用简单

在对 oVirt 云平台安全加固后, 应保证新增功能在 oVirt 管理界面上简单易用，不应使其步骤过于繁琐，应尽量简化操作，这样能够给用户带来方便。

4.2.2 设计思路

下面我们将通过结合上一章对 oVirt 云平台的对虚拟机磁盘加密、虚拟机深度擦除、USB 设备实时管控三方面关键技术分析，分别阐述在 oVirt 云平台安全加固中这三方面的设计思路。

1）虚拟机磁盘加密

在 oVirt 云平台中，虚拟机对应在宿主机上以一个文件的形式存在。客户虚拟机操作系统需要进行 I/O 操作时，会产生一个 VM exit 退出到 KVM 中，KVM 捕获到该 I/O 操作，然后交由 Qemu 来完成处理。Qemu 实现了虚拟机设备的模拟，Qemu 模拟读写数据并发出请求, 这时真实设备驱动才会访问硬件。如图 4-2 所示：

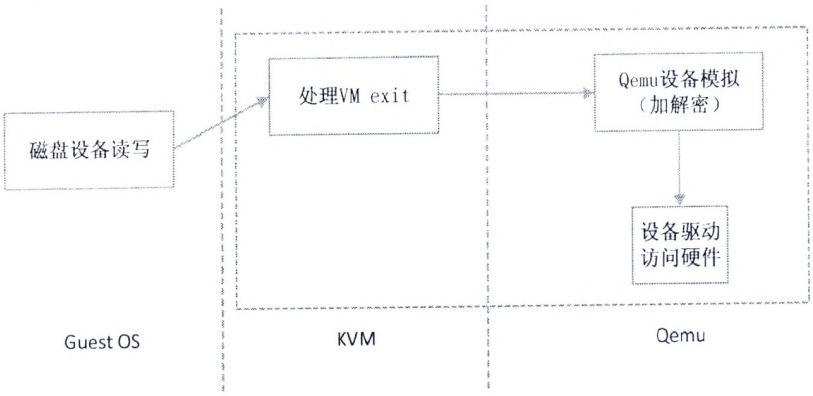


图4-2 虚拟机磁盘加密思路

从上图 4-2 可以看出，所有的 I/O 操作请求都会通过 Qemu 来模拟处理，所以我们可以选择在 Qemu 这一层中实现虚拟机磁盘的加解密。在数据写入物理存储介质之前进行加密，相反，在读取数据时对数据进行解密，这样便能实现对虚拟机磁盘数据的加解密。同时，磁盘块设备是固定长度的块来读写数据的，对于大多数磁盘而言，这个固定长度都是 512 字节，也称为磁盘扇区。正是由于块设备的特殊性，磁盘数据都是通过固定大小的块来读写，所以我们在加解密过程中，在上图 4-2 中的框内不需要考虑其块的内容，只需要对磁盘数据按照固定大小块进行加解密读写就行了，这也符合前面提到的设计原则。

虚拟机磁盘通过加解密后，用户虚拟机数据信息得到了有效保护。由于虚拟机镜像文件经过加密，窃取者或者云平台管理人员即使在获得了用户虚拟机镜像文件，在没有用户密钥的情况下，他们也无法正确解析出虚拟机中的用户数据信息，达到了虚拟机磁盘加密的目的。

2）虚拟机深度擦除

在 oVirt 云平台中，删除用户虚拟机只是简单地将虚拟机镜像文件删除，虚拟机用户数据实际还在磁盘的数据区中，并没有做任何处理。在上一章的实验也证明了这一点存在安全隐患，因此我们将采用数据覆写的方式对虚拟机用户数据进行深度擦除。

虚拟机深度擦除的设计思路主要是通过虚拟机所属存储池 spUUID、虚拟机所属磁盘域 sdUUID、虚拟机唯一标识符 UUID 定位到需要被删除的虚拟机镜像文件，本文也提出了多级数据擦除安全机制，通过存储介质的擦除标准设定了四种磁盘擦除方式分别是：一次擦除采用 0x00 填充；三次擦除采用依次以 0x00、0xFF、RAND 填充；6 次擦除采用依次以 0xFF、RAND、RAND、RAND、0xFF、0x00 填充；7 次擦除采用依次以 RAND、0x00、0xFF、RAND、RAND、0x00、0xFF 填充，其中第一遍和最后一遍采用的全 1 和全 0 填充主要是为了进一步净化噪声，

用户可以根据自己虚拟机数据密级来选择自己的擦除磁盘的方式。由于虚拟机磁盘文件一般都是数十 GB 甚至几百 GB 以上，所以覆写擦除是一个耗时的操作，选择不同的擦除方式耗时会有所不同，擦除是次数越多耗时越多。

对于虚拟机文件覆写擦除操作属于 IO 密集型操作，为了在虚拟机文件擦除操作时，让系统能够正常处理其它操作，我们使虚拟机擦除模块在后台运行，相对于用户是透明操作。多级的数据擦除安全机制也起到了平衡数据擦除安全性和性能需求的目的。

通过虚拟机磁盘深度擦除，虚拟机用户数据被彻底销毁，有效地防止了不坏好意者恢复并使用用户虚拟机数据，使得用户虚拟机数据安全得到保护，起到了虚拟机磁盘深度擦除的作用。

3）USB 设备实时管控

对于用户来说，USB 设备是不可或缺的重要设备，oVirt 用户通过 Spice 工具连接到虚拟机桌面，在远程 PC 机上插入 USB 设备，便可被重定向到远程虚拟机桌面上。从上一章分析可以看出，在 oVirt 云平台中，并没有对 USB 设备进行有效的管控。这为不怀好意的人有了可乘之机，使得企业的重要数据信息存在严重安全隐患。因此，我们需要对 USB 设备进行实时管控。

由于虚拟机设备最终都是由 Qemu 模拟来完成的，我们的 USB 设备实时管控的思路是：在 Qemu 中 USB 设备重定向机制中添加 USB 设备授权访问，设置一个 USB 设备白名单, 只有 USB 的 PID 和 VID 和白名单内 USB 的 PID 和 VID 一一对应，才允许虚拟机加载该 USB 设备。USB 模拟重定向过程中，USB 设备监控模块监控 USB 设备白名单, 如有更改, 通知 Qemu 中 USB 设备管控模块完成相关 USB 设备的加载或卸载，实现 USB 设备的实时管控。实现原理如图 4-3 所示：

通过 USB 设备的实时管控后，USB 设备必须经过 oVirt 云平台管理人员允许插入到某一个虚拟机情况下，USB 设备才能被重定向到对应的虚拟机中，否则，插入 USB 设备，远程虚拟机桌面并不会加载该 USB 设备。同时，oVirt 管理人员在一个虚拟机中删除某一 USB 设备白名单后，在该虚拟机中，USB 设备会被立刻卸载掉。这样对 USB 设备起到了实时管控目的。

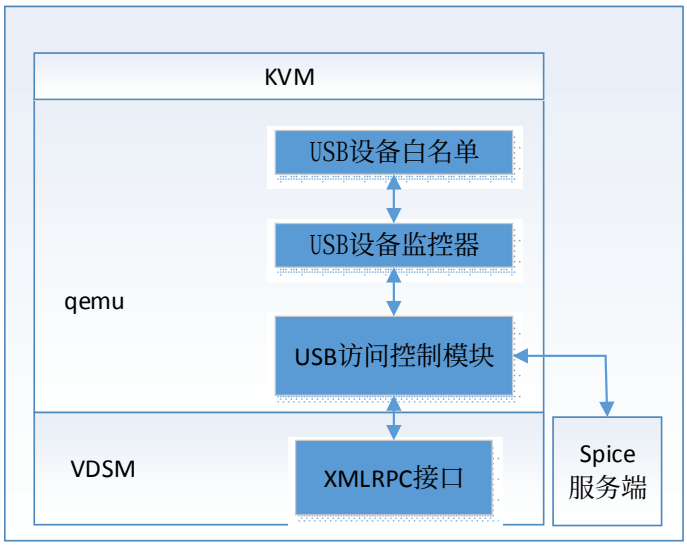


图4-3 USB 设备实时管控思路

4.3 oVirt 云平台安全加固设计整体框架

结合第三章的系统关键技术分析以及上面提出的设计思路，通过源码研究分析总结以及不断地实验验证，针对 oVirt 云平台环境，最终设计出了 oVirt 云平台安全加固解决方案的总体框架。如图 4-4 所示：

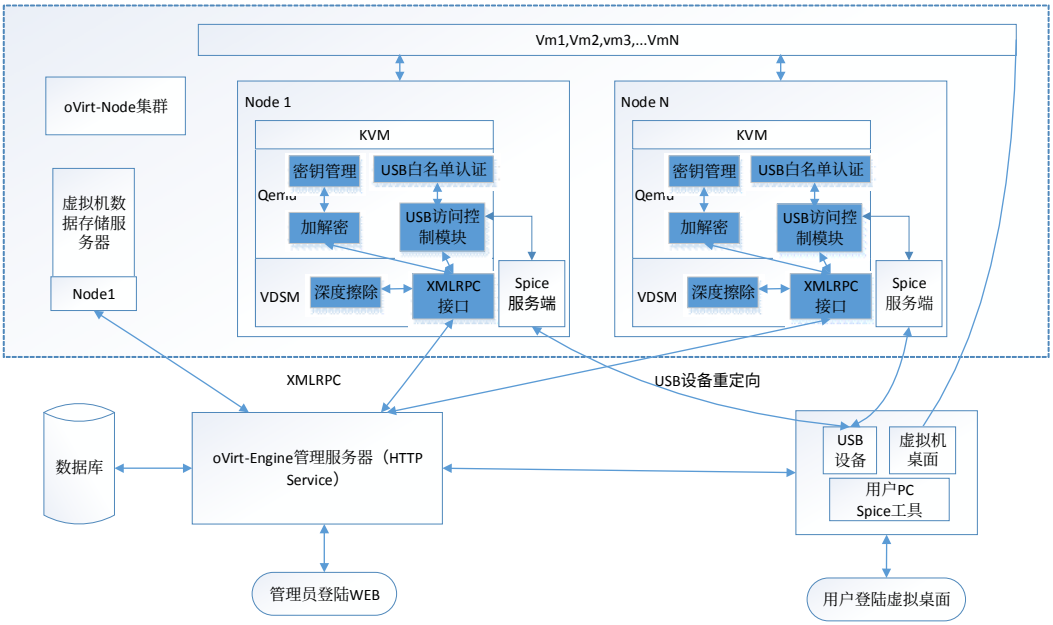


图4-4 oVirt 云平台安全加固整体框架

根据上图 4-4 可以看出，本 oVirt 云平台安全加固解决方案中包括了密钥管理模块、加解密模块、磁盘深度擦除模块、USB 访问控制模块、USB 白名单认证模块。密钥管理模块和加解密模块存于 Qemu 中，密钥模块用于管理和存储与虚拟

机的 UUID 对应的密钥，对用户身份进行验证，通过才能进行进行后续虚拟机操作。加解密模块是用于对虚拟机磁盘加解密，在对磁盘写入数据之前加密，对磁盘读取数据后进行解密，这操作对于用户是透明的。虚拟机深度擦除模块是一个存放于 VDSM 层，是一个独立的擦除模块，专门以多安全级的擦除标准对文件进行深度擦除。USB 访问控制模块和 USB 白名单认证模块同加解密模块一样也是位于 Qemu 中的，USB 访问控制模块是实时对虚拟机 USB 设备进行监控，实现对虚拟机 USB 设备实时管控，实时地管理虚拟机 USB 设备是否加载或卸载。而 USB 白名单认证模块是用于对 USB 设备的授权认证，通过 USB 设备的 VID 和 PID 标识符来管理 USB 设备，用户在远程桌面使用 USB 设备时，需要认证该 USB 设备是否授权。

为了能更好地理解总体框架中各个模块之间的关系，我们对上述总体框架图中的各个模块进行归纳整理，整理后最终得出了整个系统功能模块图，如图 4-5 所示：

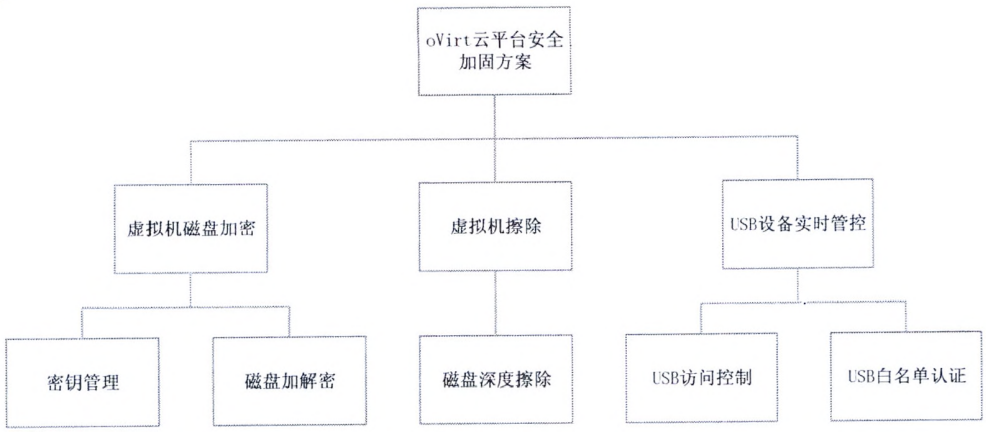


图4-5 系统功能模块

从上图 4-5 可以看出，本 oVirt 云平台安全加固方案中，各个模块被划成分为了三个子系统，分别是：虚拟机磁盘加密、虚拟机擦除、USB 设备实时管控。

虚拟机磁盘加密子系统中主要包括了密钥管理模块和磁盘加解密模块，完成了在 Qemu 中对虚拟机磁盘文件进行透明加解密，使得虚拟机磁盘文件存储在服务器中更加安全可靠。

虚拟机擦除子系统只含有磁盘深度擦除模块，完成了多级虚拟机磁盘安全擦除，起到平衡数据擦除安全性和性能需求的作用。

虚拟机 USB 设备实时管控子系统包含了 USB 访问控制模块和 USB 白名单认证模块，完成了有效地对远程虚拟机桌面上的 USB 设备的实时管控，企业能够很好的控制员工的 USB 设备使用情况。

4.4 本章小结

本章在第三章对 oVirt 云平台系统及其关键技术分析基础上，阐述了 oVirt 云平台功能需求和本文的实验环境，接着对本文的 oVirt 云平台安全加固决解方案提出了四点设计原则，同时分别对虚拟机磁盘加密、虚拟机磁盘深度擦除、USB 设备实时管控提出了设计思路。最后，总结提出了 oVirt 云平台安全加固设计的总体框架。

第五章 方案详细设计及实现

根据上一章的解决方案的总体设计，本章将从整体框架着手，详细介绍磁盘加解密子系统、磁盘擦除子系统和 USB 设备实时管控子系统三大子系统的实现过程。

5.1 磁盘加解密子系统

虚拟机磁盘加解密主要目的是为了保护虚拟机用户数据安全，使用户能够放心地将自己的数据信息存放在 oVirt 云平台服务器上。磁盘加解密子系统包括了密钥管理和磁盘加解密两个模块，他们均部署于 Qemu 中。如图 5-1 所示：

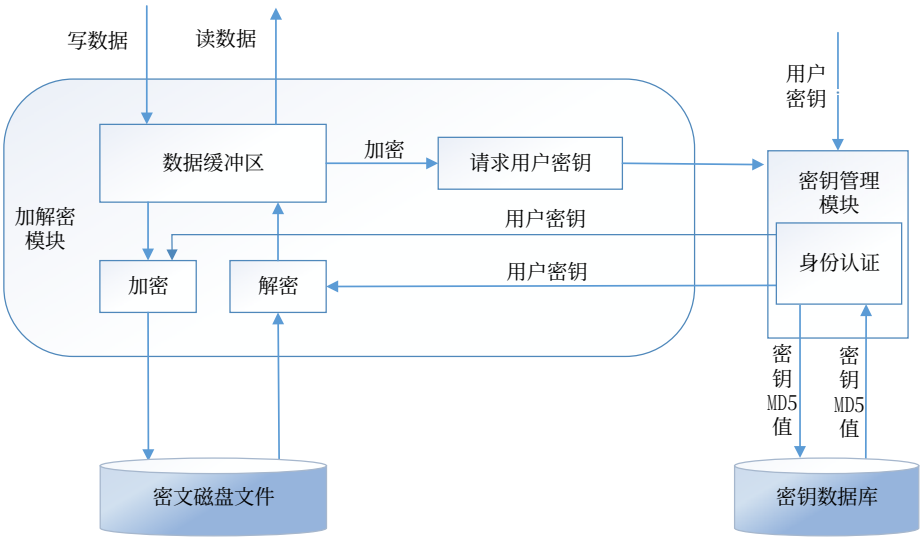


图5-1　磁盘加解密子系统加解密流程

上图 5-1 可以看出，用户登录虚拟机时，需要用户输入虚拟机密码，用户密钥传递给密钥管理模块，在密钥管理模块中把用户密钥通过 MD5 加密，并与保存在数据库中用户最初设定的密码 MD5 值进行比对，鉴别是否是合法用户。通过身份认证后，我们才允许用户登录到远程虚拟机桌面对虚拟机进行相关操作，否则用户无法登录到虚拟机远程桌面。在加解密过程中，对虚拟机磁盘文件写数据时加密，首先需要向密钥管理模块请求用户密钥，获取到密钥后，利用密钥对虚拟机文件进行加密。对虚拟机密文磁盘文件读取数据时解密，同样是通过密钥管理模块获取用户密钥对虚拟机文件进行解密。下面我们将对密钥管理模块和磁盘加解密模块详细设计和实现过程进行介绍。

5.1.1 密钥管理

密钥管理模块在虚拟机磁盘加解密子系统中相对次要的一个模块，其主要职责是对用户身份认证和密钥存储。首先介绍本模块的原因是只有通过先理解了虚拟机磁盘加密方案中密钥的管理保护机制，才能更好地理解虚拟机磁盘加解密模块中的运行原理。

5.1.1.1 用户身份认证

在 oVirt 云平台创建一个新的虚拟机时，虚拟机用户需要为其设定一个密码，该密码用于对虚拟机磁盘文件的加解密和远程登录到虚拟机桌面。然而其中存在两处隐患：首先是客户机和服务器之间是通过网络来通信的，只要黑客通过网络截取到客户机和服务器之间传送的数据包，通过解包分析就能够获取到用户密码；其次考虑到用户密码存放在服务器的数据库中，oVirt 平台的管理员可以获取到用户密码，通过用户密码来获取加密的虚拟机用户数据。如果存储在数据库的用户密码没有受到保护，虚拟机用户数据泄漏的可能性大大增加，因此，本文采用在客户机上对用户密码进行 MD5 加密后，再通过网络传输到服务器上，存储于服务器数据库中。MD5 是单向不可逆的加密算法，具有以下三个非常重要的特性：其一是单向加密性，明文数据经过加密后，无法通过加密后的密文逆向解密得到其明文数据；其二是数字指纹性，任何一个相同明文数据加密过后，得到的密文都是永恒不变的，并且任何两个不同明文数据加密过后，会得到两个不同的密文；其三是安全性，MD5 是单向不可逆的，经过 MD5 加密后的数据具有很高的安全性

从以上 MD5 特性中看出, 十分适合用 MD5 加密用户密码。用户密码经过 MD5 加密以后保存在数据库中，即使 oVirt 云平台管理员获取到了保存在数据库中经过 MD5 加密后的用户密码, 并不能解密出用户真实密码, 增加了用户密码破解难度，从而保护了虚拟机用户数据安全。所以,本文采用了对用户密钥多次 MD5 算法加密来实现用户身份认证，由于 MD5 是单向不可逆加密算法，我们不能将 MD5 加密后的密码转化为用户密码，所以只能将用户密码使用 MD5 加密后，然后通过查询数据库该用户保存的密码 MD5 值，对其进行比对，如果成功，通过了身份认证，给用户反馈成功登录信息。否则，用户身份认证失败，给用户反馈登录失败信息。同时用户密码采用多次 MD5 算法加密后，大大地增加了黑客通过穷举法来获取用户密码的难度和成本。其用户身份认证过程如图 5-2 所示：

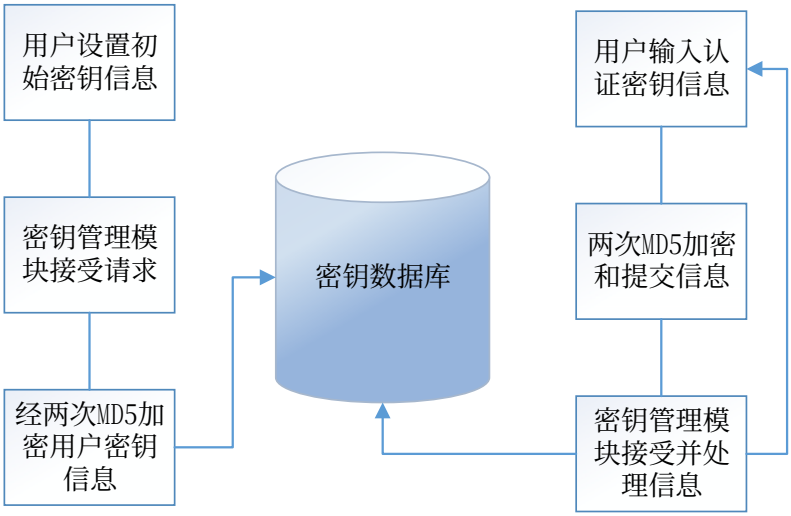


图5-2 用户身份认证过程

在用户身份认证过程中，首先当用户创建一个新的虚拟机时，用户需要为该虚拟机设定一个密码，然后传递给密钥管理模块，在密钥模块中默认设计了一个复杂密码，将它的 MD5 算法加密后的 MD5 值与用户密码 MD5 算法加密后的 MD5 值相加，再用一次 MD5 算法加密，经过两次 MD5 算法加密后，然后将其保存到服务器数据库中。连续使用两次 MD5 算法加密的目的是在于防止黑客采用“跑字典”等方法来破解用户密钥理论上的可能性。即使黑客破解了存放在数据库中加密后的用户密码，但是不知道我们采用了几次 MD5 加密，所以并不能获取到真正的用户密码，从而有效地保护了用户密码安全。

然后，用户利用远程 PC 机登录到虚拟机远程桌面时，用户需要输入虚拟机用户密码，点击确定后，客户端先将用户密钥通过网络发送给密钥管理模块。密钥管理模块接受用户密码后，将用户密码连续进行两次 MD5 加密后，从密钥数据库中查询该用户创建虚拟机时候保存加密后的用户密码，并将它们进行比对，如果匹配成功，用户是合法虚拟机用户，向用户返回登录成功的消息，如果匹配失败，则向用户返回登录失败的消息。

5.1.1.2 密钥存储

密钥存储是密钥管理模块中一个重要的组成部分，存储了用户密钥和虚拟机的相关信息。在上一小节中，已经详细阐述了用户密钥经过多次 MD5 算法加密存储，达到了保证用户密钥安全的目的，在本小节中就不再提及，下面将设计用户密钥数据库。

由于在 oVirt 云平台安全加固解决方案中密钥数据库设计并不是重点，只需要建立一张密钥数据表，主要包含了虚拟机 UUID 与虚拟机所属存储池 spUUID、虚

拟机所属磁盘域 sdUUID 和虚拟机用户密钥的一一对应关系。如果 oVirt 云平台中虚拟机数量非常多的情况下，可以根据用户虚拟机访问量情况，挑选出经常访问的虚拟机并为其再单独建立一张经常访问的虚拟机密钥数据表，用于提高查询和比对虚拟机用户密钥加密后的 MD5 值。其密钥数据表结构如表 5-1 所示：

表 5-1 密钥数据表结构

编号	字段名	数据类型	说明	允许为空
1	UUID	varchar(255)	虚拟机 UUID	否
2	spUUID	varchar(255)	存储池 spUUID	否
3	sdUUID	varchar(255)	磁盘域 sdUUID	否
4	userKey	varchar(32)	用户加密密钥	否
5	ctime	Date time	虚拟机创建时间	是
6	ltime	Data time	虚拟机最后访问时间	是
7	PV	int	虚拟机访问量	是

在密钥存储中，用户密钥数据表是非常重要的数据表，它保存了用户虚拟机相关密钥信息，如果密钥数据表出现偶然数据丢失情况下，将会严重影响到 oVirt 云平台的正常运行。因此，通常情况下，我们需要对密钥数据库进行备份，用于恢复用户密钥数据库使用。从表 5-1 中，我们可以看出用户密钥数据表包含了虚拟机 UUID、存储池 spUUID、磁盘域 sdUUID、用户加密密钥、虚拟机创建时间、虚拟机最后访问时间和虚拟机访问量七个虚拟机基本信息。其中虚拟机 UUID、虚拟机存储池 spUUID 和虚拟机磁盘域 sdUUID 是用于定位虚拟机磁盘存储在那个 Node 节点上，用户加密密钥则用于虚拟机磁盘加解密，虚拟机创建时间和虚拟机最后访问时间用于 oVirt 管理维护虚拟机使用，最后的虚拟机访问量是用于当 oVirt 云平台中虚拟机过多情况下挑选出经常使用的虚拟机，提高用户密钥 MD5 值查询和比对效率。

5.1.2 磁盘数据加解密

磁盘数据加解密模块是磁盘加解密子系统的重要模块，它负责了对虚拟机用户磁盘数据的加解密功能。

在 oVirt 云平台中，虚拟机磁盘镜像格式包含了 raw[30]和 qcow2[31]两种格式磁盘。raw 格式磁盘的有通过 ISO 镜像新创建的虚拟机磁盘和模板磁盘，其中通过模板创建出来的虚拟机共享该模板中 raw 格式的磁盘，通过硬连接的方式指向同一块数据文件，对于该虚拟机新增或改变的数据，统统保存在一个 qcow2 格式的磁

盘中。因此，我们需要分别对这两种格式磁盘进行加解密。在详细介绍磁盘数据加解密之前，为了更好地理解对这两种格式磁盘加解密详细过程，我们先简单分析比较一下 raw 和 qcow2 这两种格式：

- raw 的优势

1） raw 格式是原始镜像格式，比较简单，可以直接当作为一个块设备给虚拟机使用。

2） raw 格式磁盘的虚拟机比 qcow2 格式磁盘的虚拟机 IO 效率更高。

3） raw 格式根据实际的使用量来占有空间使用量，并非是设定 raw 格式磁盘的最大值，linux 下的文件系统能够很好地支持空洞这一特性，磁盘文件的空洞是交由宿主机上的文件系统来管理的。

4） raw 格式磁盘可以改变磁盘空间最大值（qcow2 格式也可以改变磁盘空间最大值，不过需要通过转化为 raw 格式后才能改变其大小）。

5） raw 格式磁盘能够直接挂载到宿主机上，在没有启动虚拟机的情况下，就能够在宿主机和虚拟机之间进行数据传输。

- qcow2 的优势

1） qcow2 格式占有更小的磁盘空间，即使是宿主机文件系统不支持空洞的情况下。

2） qcow2 格式支持写时拷贝（copy-on-write），一个 qcow2 镜像可以保存其它模板镜像的增量或者变化部分，并不修改原始的模板镜像文件。

3） qcow2 格式支持快照。

4） qcow2 格式支持基于 zlib 的压缩方式。

5） qcow2 格式在 Qemu 中可选择 AES 加密。

通过上面的分析比较，我们大概对 raw 和 qcow2 这两种格式磁盘及其各自的优势有所了解。在 Qemu 源代码中，已经基本实现了 qcow2 格式磁盘加密，所以我们只需要在 Qemu 中做轻微的修改，便能完成在 oVirt 云平台中对 qcow2 格式磁盘加密底层功能。但是，在 Qemu 中并没有实现 raw 格式磁盘的加密，下面将对实现 raw 格式磁盘加解密过程详细介绍。

在 Qemu-KVM 中，虚拟机的所有磁盘文件的 I/O 操作都是通过 KVM 来截获，然后将其传递给 Qemu 来模拟实现。因此，我们根据第三章虚拟机镜像文件加密关键技术中研究的切入点对虚拟机磁盘进行加解密。

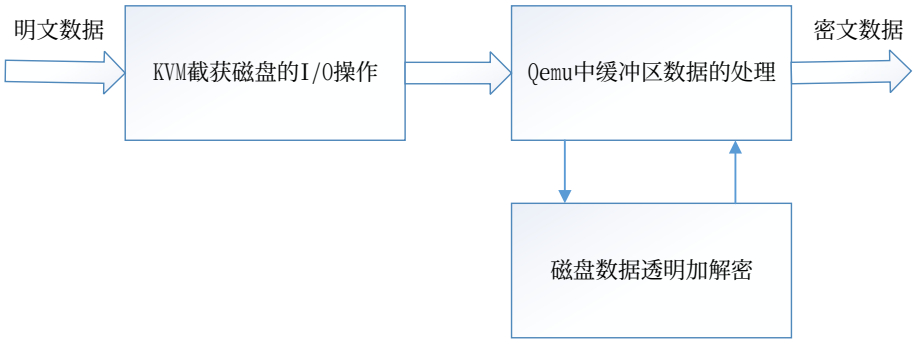


图5-3　虚拟机磁盘加密关键点

从上图 5-3 可以看出，对虚拟机磁盘加解密的实现过程中主要有两个关键点：

1） Qemu 中 Buffer 数据的处理：该方面将根据第三章虚拟机镜像文件加密关键技术进一步具体实现。

2） 磁盘数据透明加解密：在数据写入磁盘之前进行加密，读数据之后进行解密，对于用户而言，这两个过程均是透明的。

5.1.2.1 缓冲区数据处理

结合第三章虚拟机镜像文件加密关键技术分析，在对缓冲区数据进行加解密之前，我们需要先了解下面两个重要的缓冲区数据存储结构。

```
struct iovec {
    void *iov_base;   //缓冲区地址
    si ze_t iov_len;   / /表示数据大小
};
```

在 iovec 结构体中，iov_base 指向了一个缓冲区，这个缓冲区是存放了磁盘读写时的数据，成员 iov_len 表示了该缓冲区存放的数据实际大小。

```
typedef struct QEMUIOVector {
    st ruct iovec *iov;    //表示为struct iovec类型的数组
    int niov;              //实际向量元素个数
    int nalloc;            //数组元素个数
    size_t size;           //实际数据总大小
} QEMUIOVector;
```

QEMUIOVector 是在 Qemu 中读写所用的 I/O 向量，所以我们需要对这个向量所有元素指向的数据进行加解密处理。本文采用的是 AES 加密算法，也可以使用其它加密算法。由于结构体 QEMUIOVector 并不能直接加解密算法中函数进行加解密，因此，我们需要对缓冲区数据进行相应的处理。

根据前面找到的虚拟机磁盘加密切入点 raw_co_ready 和 raw_co_writev 两个函

数，分别对其缓冲区数据进行处理。

1）读操作缓冲区数据处理

在 raw_co_ready 函数中，首先定义了两个新的缓冲区 QEMUIOVector hd_qiov 和 uint8_t cluster_buf，然后，对函数中的 qiov 缓冲区数据进行相应的处理后，方便 AES 算法解密。其相应的处理函数如下：

```
qemu_iovec_init(&hd_qiov,qiov->niov); //初始化hd_qiov缓冲区
qemu_iovec_reset(&hd_qiov); //重置hd_qiov缓冲区
buf = qemu_blockalign(bs,MAX_CRYPT_BUF_SIZE) //申请缓冲区cluster_buf，应足够大
qemu_iovec_add(&hd_qiov, cluster_buf,512*nb_sectors); //关联hd_qiov与cluster_buf
bdrv_co_readv(bs->file, sector_num, nb_sectors, &hd_qiov);//读取数据到hd_qiov中
qemu_iovec_from_buf(qiov,0, cluster_buf,512*nb_sectors); //解密后的数据保存到qiov中，
                                                          返回
qemu_iocec_destroy(&hd_qiov);//释放hd_qiov缓冲区
qemu_vfree(cluster_buf);//释放cluster_buf缓冲区
```

2）写操作缓冲区数据处理

在 raw_co_writev 函数中，也定义了两个新的缓冲区 QEMUIOVector hd_qiov 和 uint8_t buf，然后，对函数中的 qiov 缓 s 冲区数据进行相应的处理后，方便 AES 算法加密。其相应的处理函数如下：

```
qemu_iovec_init(&hd_qiov,qiov->niov); // 初始化hd_qiov缓冲区
 cluster_buf = qemu_blockalign(bs,MAX_CRYPT_BUF_SIZE) //申请缓冲区cluster_buf，
                                                       应足够大
qemu_iovec_to_buf(&hd_qiov,0, cluster_buf,512*nb_sectors); //将hd_qiov的数据线性保
                                                            存到buf
qemu_iovec_reset(&hd_qiov); // 重置hd_qiov缓冲区
qemu_iovec_add(&hd_qiov, cluster_buf,512*nb_sectors); // 将加密的数据保存到hd_qiov
bdrv_co_writev(bs->file,sector_num,nb_sectors,hd_qiov); // 加密后的数据写入磁盘
qemu_iocec_destroy(&hd_qiov);//释放hd_qiov缓冲区
qemu_vfree(cluster_buf);//释放cluster_buf缓冲区
```

在上面的读写操作缓冲区数据处理中，缓冲区数据处理次序有细微区别，分别是在向磁盘写入数据前和在磁盘读取数据后，但是原理基本相同。均是采用先将申请的新数据缓冲区 buf 和 QEMUIOVector 结构体 hd_qiov 关联，接着把数据先拷贝到 buf 缓冲区中，经过 AES 算法加解密后，再进行相应的读写操作，最后释放 hd_qiov 和 buf 两个数据缓冲区。

5.1.2.2 磁盘数据透明加解密

在处理完缓冲区数据操作之后，接着就是对硬盘数据进行加解密的过程，首先来介绍一个非常重要的数据结构 BlockDriverState，它是由 Qemu 所定义的，表示了一个虚拟机磁盘设备，当 Qemu 接受到磁盘 I/O 请求时，Qemu 会把磁盘 I/O 请求转化为文件读写请求，并向 BlockDriverState 结构中传入设备初始化时注册的磁盘读写函数, 不同的虚拟磁盘设备对应不同的磁盘读写函数。该 BlockDriverState 结构贯穿了 Qemu 虚拟机磁盘读写的整个过程，所以本文也对此结构增添了用于磁盘数据加密的用户密钥信息和加解密控制信息，便于密钥和加解密控制信息的传递。BlockDriverState 数据结构如下：

```
struct BlockDriverState {
    in t64_t                total_sectors; /*扇区数目*/
     int                    read_only;   /*如果为真，磁盘为只读设备*/
     int                    open_flags;   /*  标志文件是否打开*/
     int                    encrypted;   /*如果为真，磁盘已加密*/
     int                    valid_key;   /*如果为真，磁盘已经设置了一个有效密钥  */
    in t                    sg;            /* if true, the device is a /dev/sg* */
     int                    copy_on_read;
     /*  一个引用计数，如果不为零，表示有磁盘对后备镜像的引用*/
     int                    enc                 /*表示磁盘是否加密，为本文添加*/
     AES_KEY                encKey;             /*加密密钥，为本文添加  */
     AES_KEY                decKey;             /*解密密钥，为本文添加  */
     BlockDriver            *drv;       /*如果为空，表示没有设备  */
     void                   *opaque;
     void                   *dev;
    c onst BlockDevOps      *dev_ops;
     void                   *dev_opaque;
     char                   filename[1024];         /*虚拟磁盘文件名*/
    c har                   backing_file[1024];   /*如果不为空，表示磁盘镜像的差量*/
    c har                   backing_format[16]; /*如果不为空,表示存在后备镜像差量*/
     int                    is_temporary;
    B lockDriverState        *backing_hd;     /*指向设备的描述队列头*/
     （此处省略若干定义）
    c har                     device_name[32];       /*设备名称*/
    HBi tmap                  *dirty_bitmap;
     int                      in_use;
```

```
        QTAILQ_ENTRY(BlockDriverState)     list;     /*设备队列*/
        QLIST_HEAD(, BdrvTrackedRequest)   tracked_requests;
        BlockJob                   *job;
        QDict                      *options;
};
```

其中设备初始化注册的 raw 格式磁盘读写操作的各种处理函数如下所示：

```
static BlockDriver bdrv_raw = {
    .format_name                = "raw",
    .instance_size              = 1,
    .bdrv_open                  = raw_open,           //打开磁盘镜像文件
    .bdrv_close                 = raw_close,          //关闭磁盘镜像文件
    .bdrv_reopen_prepare        = raw_reopen_prepare,
    .bdrv_co_readv              = raw_co_readv,       //读磁盘镜像文件
    .bdrv_co_writev             = raw_co_writev,      //写磁盘镜像文件
    .bdrv_co_write_zeroes       = raw_co_write_zeroes,
    .bdrv_co_discard            =  raw_co_discard,
    .....
};
```

接着我们介绍磁盘镜像加解密过程，磁盘数据透明加解密是在 Qemu 层，当 Qemu 获取到磁盘 I/O 请求后，磁盘加解密模块根据相应的虚拟机用户密码对虚拟机磁盘进行加解密，如图 5-4、5-5 所示：

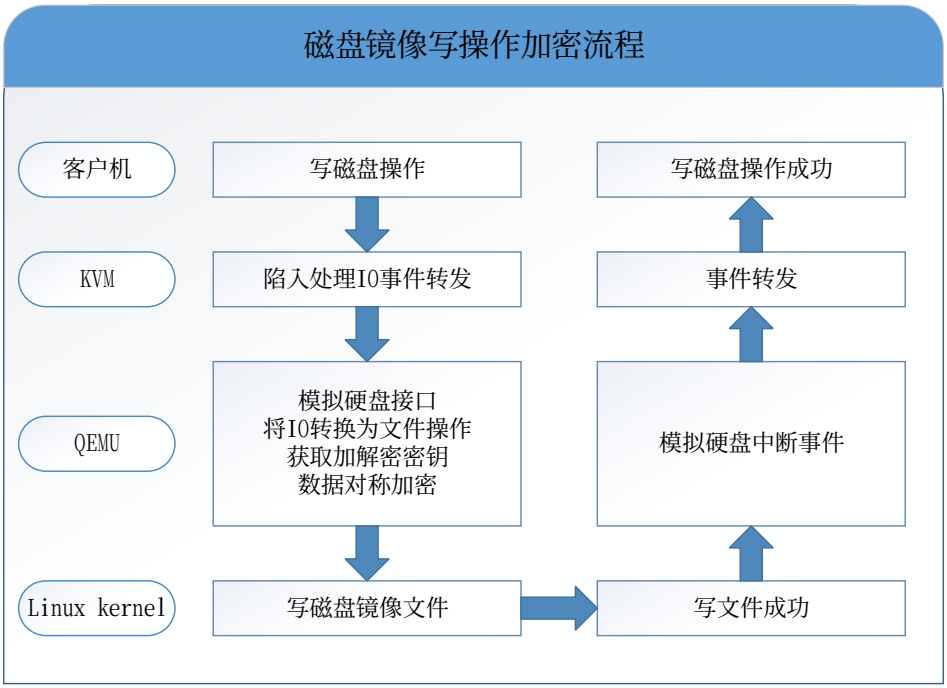


图5-4 磁盘镜像写操作加密流程

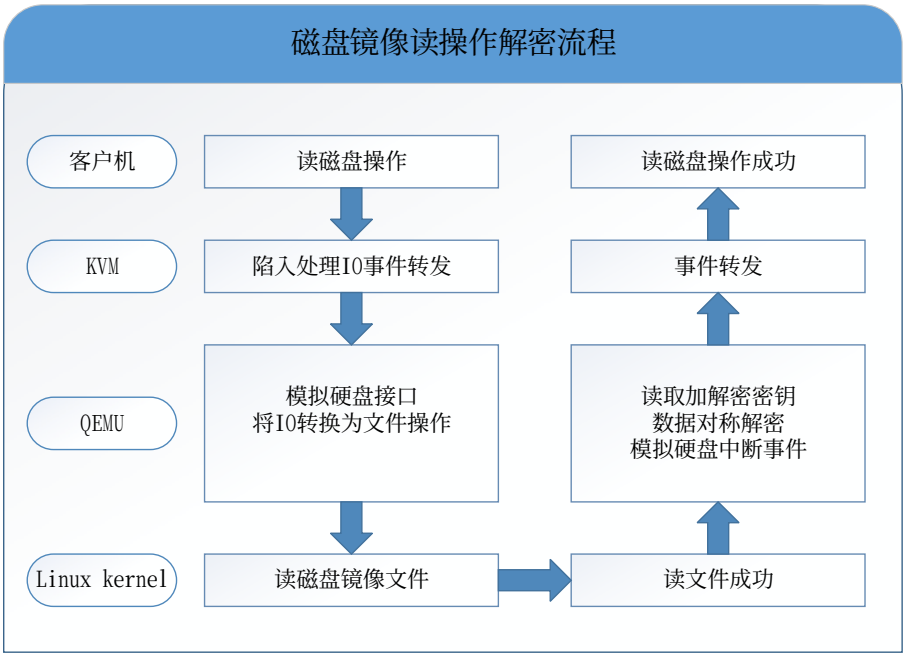


图5-5　磁盘镜像读操作解密流程

从上面图 5-4、图 5-5 可以看出，虚拟机磁盘透明加解密过程如下：

1） 用户通过虚拟机远程桌面操作虚拟机，发起客户机磁盘读写操作。

2） KVM 通过 VMCS 结构截获到磁盘 I/O 操作，并将其转发给 Qemu。

3） Qemu 为用户虚拟机模拟了硬盘设备，将虚拟机磁盘的读写操作转换成为了虚拟机磁盘镜像文件的读写操作。

4） 磁盘加解密模块获取到虚拟机相应的虚拟机用户密码，调用算法加解密函数对虚拟机磁盘数据进行加解密。

5） 数据经过磁盘加解密模块处理后，向客户机返回磁盘读写结果。

根据前面的加解密切入点 raw_co_ready 和 raw_co_writev 两个函数，并对缓冲区数据进行处理后，剩下的就是对 buffer 数据加解密。如图 5-6 所示：

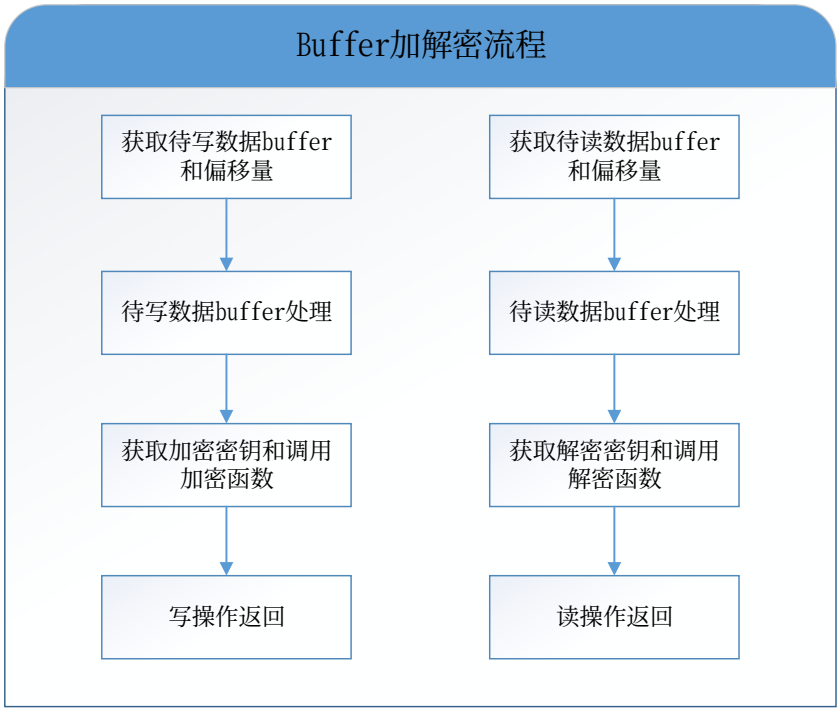


图5-6 Buffer 加解密全过程

从上图 5-6 可以看出，在对切入点 buffer 加解密全过程有以下步骤：

1） Qemu 调用 raw_co_ready 和 raw_co_writev 函数对磁盘镜像文件进行读写操作。

2） 获取到待读写数据 buffer、偏移量以及数据长度等信息。

3） 根据上一节对读写数据 buffer 进行处理。

4） 从 BlockDriverState 虚拟机磁盘设备结构中获取到用户加解密密钥和加解密控制信息，调用 AES 加解密函数接口对 buffer 数据进行加解密。加解密 buffer 中的数据是 512 的整数倍，其数据大小小于 MAX_CRYPT_BUF_SIZE。对 buf 中每个 512 字节进行加解密，根据 AES 算法特性，实质是每 16 字节进行一次加解密。

5） 最后读写操作返回。

5.1.2.3 磁盘加解密算法接口

从上一小节可以看出，我们还需要实现磁盘加解密算法接口，本文在磁盘数据透明加解密中采用的是 AES 加密算法，也可以采用其它磁盘加密算法。AES 算法具有加密强度高、效率高、灵活性好等特点，它的主要工作流程在第二章有所介绍，下面将介绍磁盘加解密算法接口的实现过程。

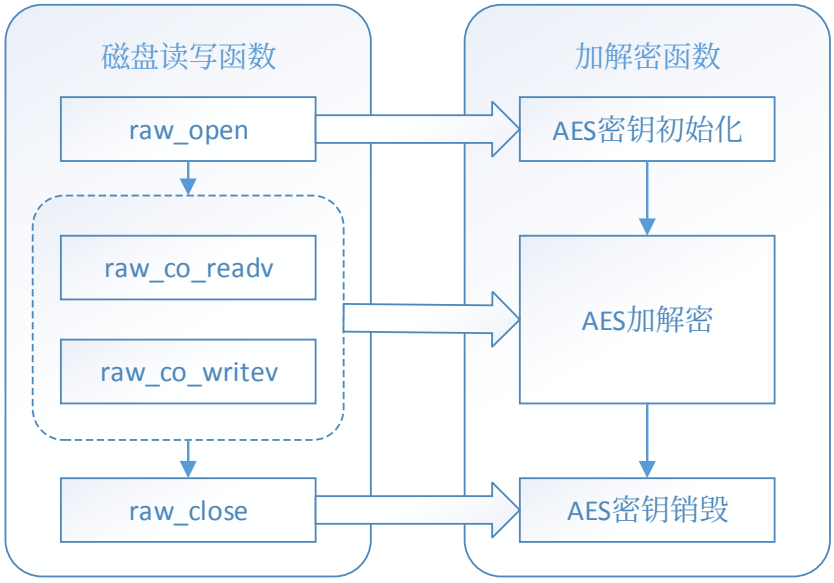


图5-7　磁盘读写函数与加解密函数关系

上图 5-7 描述了磁盘读写函数和加解密函数之间的关系，可以从中看出，细箭头代表的是函数执行过程，粗剪头代表的是磁盘读写函数所调用加解密函数接口的关系。因此，我们需要设计实现 3 个接口：密钥初始化接口、加解密接口和密钥销毁接口，下面将分别具体介绍各接口定义。

```
//密钥初始化
//设置加密密钥
int AES_set_encrypt_key(
    const unsigned char *userKey,      //用户密钥
    const int bits,                    //密码长度，以bit为单位（可以是128、192、256比特）
    AES_KEY *key                       //AES_KEY对象指针
);
//设置解密密钥
int AES_set_decrypt_key(
    const unsigned char *userKey,      //用户密钥
    const int bits,                    //密码长度，以bit为单位（可以是128、192、256比特）
    AES_KEY *key                       // AES_KEY对象指针
);
//AES加解密
void raw_encrypt_sectors(
    int64_t sector_num,                //磁盘偏移量，单位为扇区
    uint8_t *out_buf,                  //计算后输出的数据
    const uint8_t *in_buf              //需要加密/解密的数据
```

```
    int nb_sectors,                    //需要加解密数据的大小，单位为扇区
    int enc,                           //0表示解密，1表示加密
    const AES_KEY *key                 // AES_KEY对象指针
);
//密钥销毁
bool raw_destory_key(
    AES_KEY *key                 //销毁对应的指针对象，使指针指向为NULL
) ;
```

当 AES_set_encrypt_key 和 AES_set_decrypt_key 函数被调用时，把生成的 AES_KEY 对象保存在 BlockDriverState 结构新增成员 encKey 和 decKey 中，因为 BlockDriverState 结构贯穿了 Qemu 虚拟机磁盘读写全过程，在对数据进行加解密时，AES_set_decrypt_key 函数可以根据传入的 BlockDriverState 结构找到对应的 AES_KEY，并对 buffer 数据进行相应的加解密。当虚拟机被关闭的时候，会调用 raw_destory_key 函数销毁对应的 AES_KEY 对象。

5.2 磁盘擦除子系统

通过第三章的系统分析研究，从源码分析和实验证明了，在 oVirt 云平台中，虚拟机删除后的虚拟机磁盘数据并没有彻底销毁，虚拟机用户数据还在服务器存储介质上。因此，本文虚拟机磁盘深度擦除设计的目的就是为了保护虚拟机用户数据，使得用户虚拟机删除后，窃取者无法从服务器存储介质上恢复删除的虚拟机数据。在虚拟机磁盘擦除子系统中，也提出多级安全数据擦除机制概念，以供用户根据自己虚拟机资料重要性来选择相应的磁盘数据擦除方式，在用户数据安全性和磁盘数据擦除性能需求之间找到平衡点。

5.2.1 权限认证和多级安全数据擦除机制

本文采用的是以软件方式对存储介质上的虚拟机数据深度擦除，通过覆写存储介质数据区中数据的方法擦除虚拟机数据，由于虚拟机磁盘镜像文件一般都较大，少则几十 GB，多则几百 GB 以上。因此，虚拟机镜像文件擦除是一个非常耗时的操作过程，所以，给用户提供了不同的擦除方式，不同的擦除方式擦除的次数不同，从而擦除消耗的时间也有所不同，同时，为了防止被恶意用户利用这一点，通过虚拟机磁盘数据擦除模块擦除大规模虚拟机数据，使系统不断地覆写大量数据信息，造成系统资源浪费和过度使用。本文设计了权限认证和多级安全数据擦除机制，见流程图 5-8 所示：

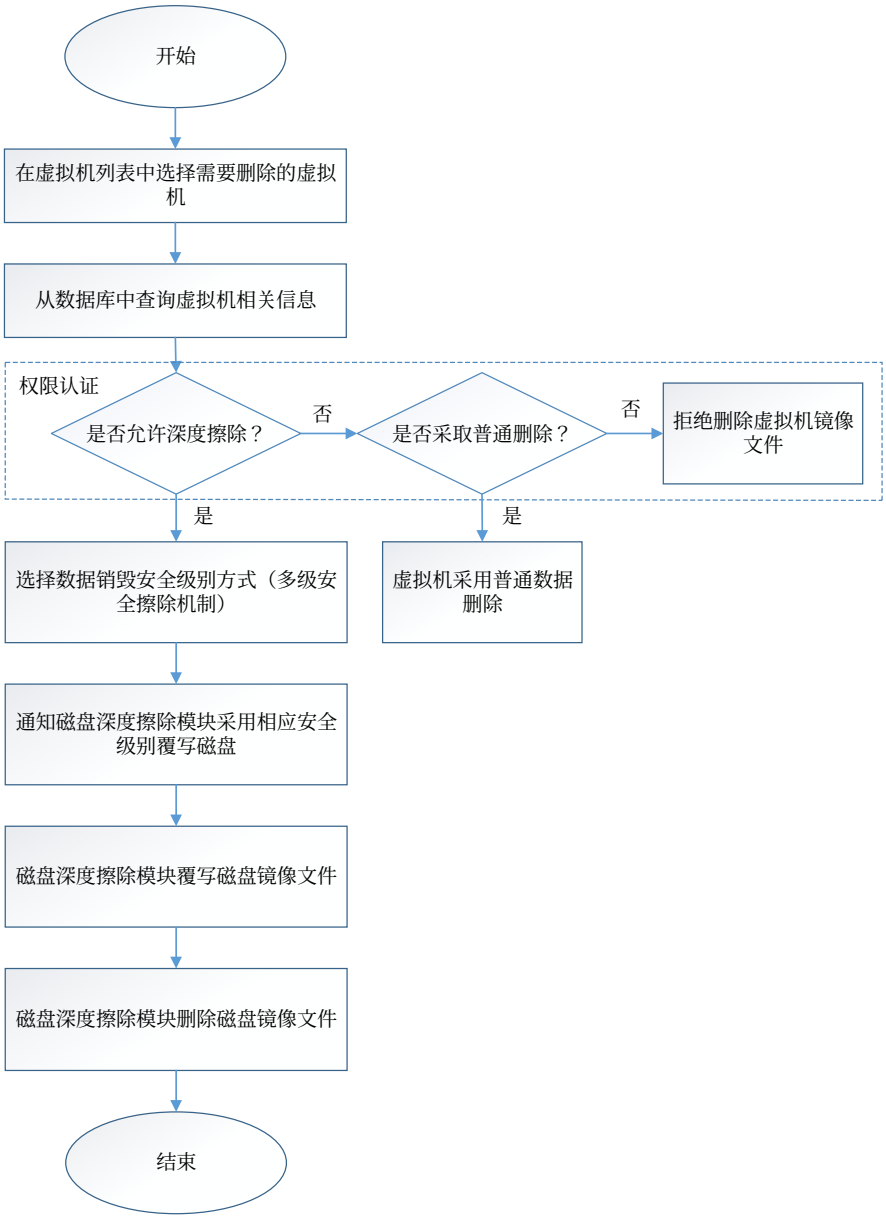


图5-8　虚拟机磁盘深度擦除流程

5.2.1.1 权限认证

从上图 5-8 中可以看出，权限认证首先查看该虚拟机是否具有深度擦除功能的权限，如果该虚拟机具有这权限，便可以使用磁盘深度擦除功能，如果该虚拟机没有这权限，接着查看虚拟机是否能够采用普通删除方式删除虚拟机数据，如果能够，虚拟机采用普通数据删除，否则，拒绝删除虚拟机数据文件。

权限认证能够有效地防止恶意用户通过利用磁盘深度擦除功能，不坏好意地删除虚拟机磁盘镜像文件，致使大规模对虚拟机磁盘镜像文件进行多次覆写，造成系统资源过度浪费。权限认证是通过 oVirt 云平台管理对每个虚拟机设定是否具

有使用深度擦除功能和普通方式删除数据的权限，然后保存于虚拟机删除功能权限数据表中，数据库设计并不是本方案的重点，所以内容比较简单，其删除功能权限数据表入表 5-2 所示：

表 5-2 虚拟机删除功能权限数据表

编号	字段名	数据类型	说明	允许为空
1	UUID	varchar(255)	虚拟机 UUID	否
2	depthErase	boolean	深度擦除	否
3	erase	boolean	普通删除	否

虚拟机删除功能权限数据表是磁盘擦除子系统中的重要数据表，控制虚拟机是否采用深度擦除或普通删除功能。其结构如上图，包括了虚拟机唯一标识符 UUID、是否采用深度擦除和是否采用普通删除这三个字段。

5.2.1.2 多级安全数据擦除

多级安全数据擦除机制主要是针对不同用户有不同的安全需求，用户根据自己不同安全需求采用不同的数据覆写方式。当用户虚拟机没有重要敏感数据时，用户可以选择全零覆写或者 DOD5220.22-M 简单的覆写标准来擦除虚拟机数据，如果用户虚拟机有重要敏感数据时，则采用 BMB21-2007 标准或者更高安全级别的标准。

本文多级安全数据擦除机制中采用了四种覆写算法标准：全零覆写、BMB21-2007 标准、DOD5220.22-M 标准和 DOD5220.22-M7 覆写标准。在本方案中并没有采用目前最安全的数据擦除 Gutman 覆写标准，因为在 oVirt 云平台中用户虚拟机磁盘镜像文件一般数据量很大，少则几十 GB，多则几百 GB 甚至 TB 以上，而 Gutman 覆写算法要求对擦除数据覆写 35 次，覆写速度慢，耗时长，这使得其擦除效率太低，将可能对 oVirt 云平台系统造成不良影响。其本文采用擦除覆写算法如下：

- 全零覆写标准

对虚拟机镜像文件采用全零方式填充。

- DoD5220.22-M 覆写标准

对虚拟机镜像文件采用一个数据覆写后，再用该数据反码进行覆写，最后再采用随机码进行覆写操作。这种覆写方式可以使其在前两次覆写过程中的是相反的磁化方向，最后再通过覆写随机码，能够有效地对虚拟机镜像文件数据进行覆盖。本文采用的覆写序列如下：

1）采用 0x00 填充虚拟机镜像文件。

2）采用 0xFF 填充虚拟机镜像文件。

3）生成一个随机数，接着用该随机数填充虚拟机镜像文件。

- BMB21-2007 覆写标准

对虚拟机镜像文件采用三次随机数覆写，两次指定数据覆写，一次上一次指定数据的反码覆写。本文采用的覆写序列如下：

1）采用 0xFF 填充虚拟机镜像文件。

2）生成一个随机数，接着用该随机数填充虚拟机镜像文件。

3）再次生成一个随机数，接着用该随机数填充虚拟机镜像文件。

4）再次生成一个随机数，接着用该随机数填充虚拟机镜像文件。

5）采用 0xFF 填充虚拟机镜像文件。

6）采用 0x00 填充虚拟机镜像文件。

- DoD5220.22-M7 覆写标准

对虚拟机镜像文件采用三次随机数覆写，两次指定数据覆写，两次上一次指定数据反码覆写。本文采用的覆写序列如下：

1）生成一个随机数，接着用该随机数填充虚拟机镜像文件。

2）采用 0x00 填充虚拟机镜像文件。

3）采用 0xFF 填充虚拟机镜像文件。

4）再次生成一个随机数，接着用该随机数填充虚拟机镜像文件。

5）再次生成一个随机数，接着用该随机数填充虚拟机镜像文件。

6）采用 0x00 填充虚拟机镜像文件。

7）采用 0xFF 填充虚拟机镜像文件。

从上面本文采用的覆写标准序列可以看出，其中第一遍或是最后一遍采用的全 1 和全 0 的方式填充，主要目的是为了进一步净化噪声，使得虚拟机镜像文件数据擦除具有更好的效果。

5.2.2 磁盘数据深度擦除

经过以上的权限认证和多级安全数据擦除机制的设计，下面将介绍磁盘数据深度擦除详细设计及其实现过程。

结合第三章的关键技术分析, 磁盘数据深度擦除并不需要 Qemu 和 KVM 的介入，因此，磁盘数据深度擦除模块设计位于 VDSM 层。磁盘数据深度擦除模块是一个独立的模块, 我们用 C 代码实现了多级安全数据覆写基础功能, 然后编译成.so 链接库供 VDSM 中 python 调用的。其中最为重要的数据结构是 srm_target 结构,

它包含了虚拟机磁盘数据擦除的内容及其采用的擦除方式。

```
struct srm_target{
   int fd;                          //文件描述符fd，表示打开的文件
   const char* file_name;           //虚拟机磁盘镜像文件名字
   long long file_size;             //虚拟机磁盘镜像文件大小
   unsigned char *buffer;           / /覆写的数据
   unsigned buffer_size;            //覆写数据的大小
   int options;                     //数据覆写标准方式
};
```

srm_target 贯穿了磁盘数据深度擦除始终，包括了虚拟机磁盘镜像名字、虚拟机磁盘镜像文件大小、覆写磁盘采用的数据已经覆写采用擦除标准等重要信息。磁盘数据深度擦除相应的主要处理函数如下：

```
//磁盘数据深度擦除最重要函数，根据srm->options不同，执行不同的擦除方式
static int overwrite_selector (struct srm_target *srm)
 {
     if(!srm) return -1;
      ……
     if(srm->options & SRM_MODE_SIMPLE)
          ……
     else if(srm->options & SRM_MODE_DOD)
          ……
     else if(srm->options & SRM_MODE_BMB)
      ……
     else if(srm->options & SRM_MODE_DODM7)
      ……
     return 0;
   };
//其它主要处理函数定义
//指定字节覆写
static int overwrite_byte(
     struct srm_target *srm,          //srm_target擦除信息结构体
     const int pass,                  //第几次擦除
     const int byte                   //覆写的数据
);
//产生随机数覆写
static int overwrite_random(
```

```
        struct srm_target *srm,          //  srm_target擦除信息结构体
        const int pass,                  //第几次擦除
        const int num_passes             //连续随机覆写几次
);
//覆写函数，供上两个函数调用
static int overwrite(
        struct srm_target *srm,        // srm_target擦除信息结构体
        const int pass                 //第几次擦除
);
```

接着我们介绍磁盘数据深度擦除过程，磁盘数据深度擦除是在 VDSM 层，当 VDSM 获取到 oVirt-Engine 管理界面的删除虚拟机请求后，磁盘数据深度擦除模块会根据相应的虚拟机信息进行判断，然后对虚拟机磁盘镜像文件做相应的操作处理，磁盘数据深度擦除逻辑图如图 5-9 所示：

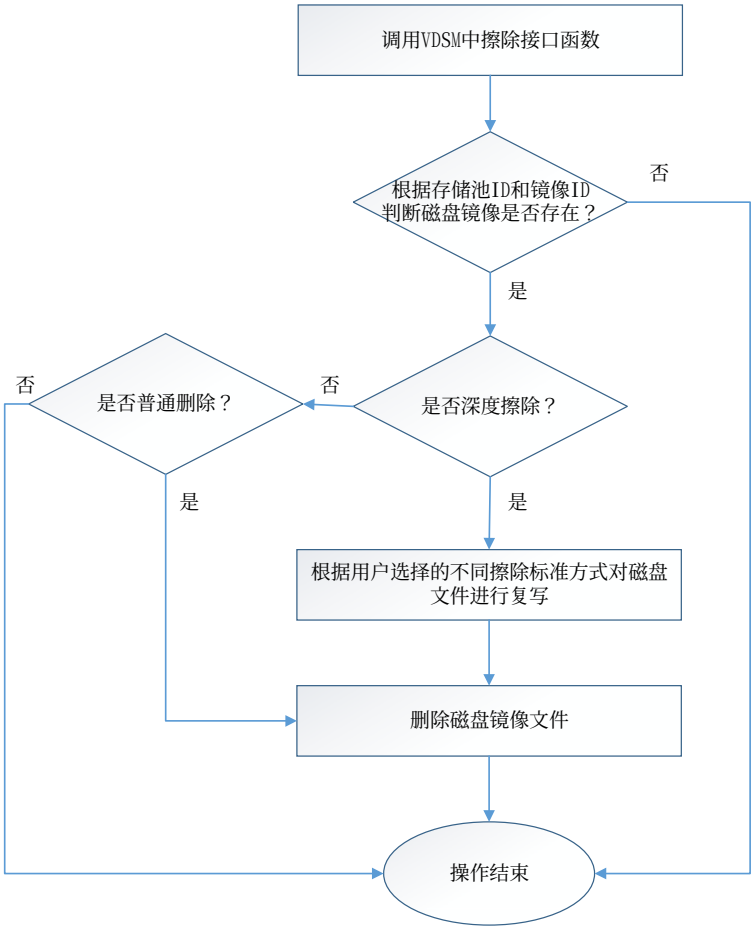


图5-9　磁盘数据深度擦除逻辑图

从图 5-9 可以看出，磁盘数据深度擦除全过程如下：

1）　VDSM 接受到 oVirt-Engine 管理界面删除虚拟机请求，调用 VDSM 中虚

拟机擦除接口函数。

2） 根据虚拟机存储池 ID 和虚拟机 ID 查找对应的虚拟机镜像文件，如果存在该虚拟机镜像文件，继续下一步操作，否则，直接结束操作。

3） 读取存于数据库中的虚拟机删除功能权限数据表信息，根据虚拟机信息，判断虚拟机是否拥有深度擦除功能的权限，如果有权限，继续下一步操作，否则，继续判断虚拟机是否拥有普通删除的权限，如果有权限，则跳至第 5 步，否则，直接结束操作。

4） 调用多级安全数据覆写基础功能的.so 动态链接库中 overwrite_selector 函数，根据用户选择虚拟机的数据擦除方式，对虚拟机磁盘镜像文件进行覆写处理。

5） 用普通方式删除服务器上的虚拟机磁盘镜像文件。

6） 操作结束，并返回结果。

至此，通过前面的权限认证、多级安全数据擦除机制和磁盘数据深度擦除的设计和实现，我们已经完成了对虚拟机磁盘镜像文件的多级安全深度擦除效果。

5.3 USB 设备实时管控子系统

USB 设备实时管控设计的目的是公司能够有效地管理员工的 USB 设备使用情况，防止公司不怀好意的员工盗窃公司内部敏感数据。主要包括了 USB 设备授权认证模块和 USB 设备访问控制模块。如图 5-10 所示：

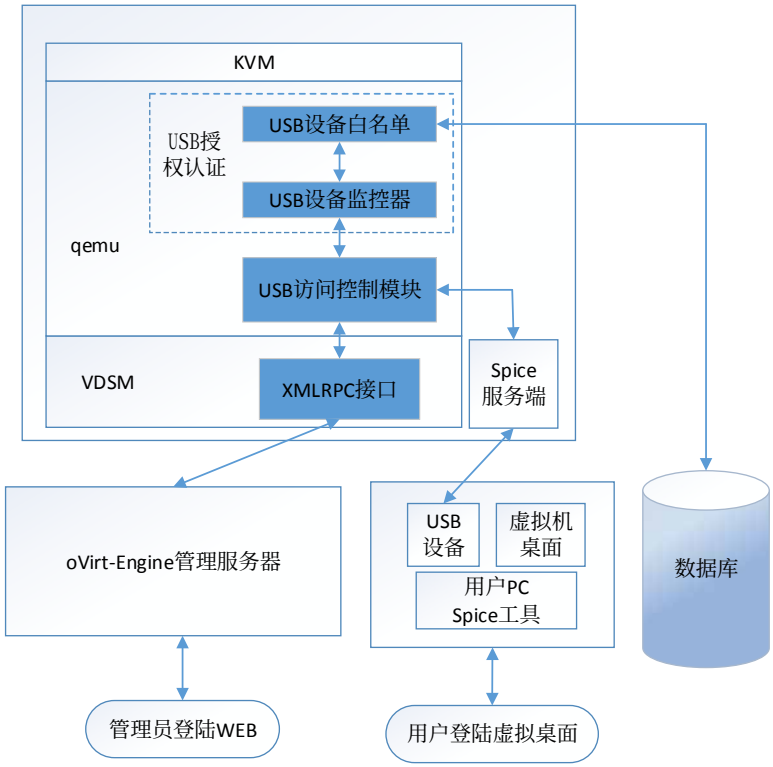


图5-10 USB 设备实时管控结构图

5.3.1 USB 设备授权认证

如图 5-4 所示, 当 oVirt 云平台用户通过一台 PC 机远程登录到虚拟机桌面后，用户 PC 机上的 USB 设备，如 U 盘、鼠标等，通过 Spice 协议重定向到远程的虚拟机桌面，最终会由 Qemu 来模拟实现。在 oVirt 云平台中，并没有对 USB 设备加以任何控制，这样使得公司内部数据存在严重的安全威胁：其一是公司内部员工可以随意通过 USB 设备拷贝数据，给公司带来严重安全隐患；其二是一旦黑客获得用户账户密码，就能进入虚拟机桌面随意拷贝数据。因此，本文采用了 USB 设备白名单的方式来对 USB 设备授权认证。每个虚拟机都对应了一张 USB 设备白名单，USB 设备白名单储存于数据库中。只有 USB 设备通过了授权认证，才能重定向到对应的远程虚拟机桌面上，否则，USB 设备不能重定向到对应的远程虚拟机桌面上。USB 设备白名单表结构如表 5-3 所示：

表 5-3 密钥数据表结构

编号	字段名	数据类型	说明	允许为空
1	UUID	varchar(255)	虚拟机 UUID	否
2	PID	varchar(5)	USB 设备供应商 ID	否
3	VID	varchar(5)	USB 设备产品识别码	否

oVirt 云平台管理员负责管理云平台 USB 设备插入情况, 为每个虚拟机定制允许访问的 USB 设备 PID 和 VID 白名单，给虚拟机允许访问的 USB 设备授权。用户在远程桌面使用 USB 设备时，需要认证该 USB 设备是否授权，认证流程如图 5-11 所示：

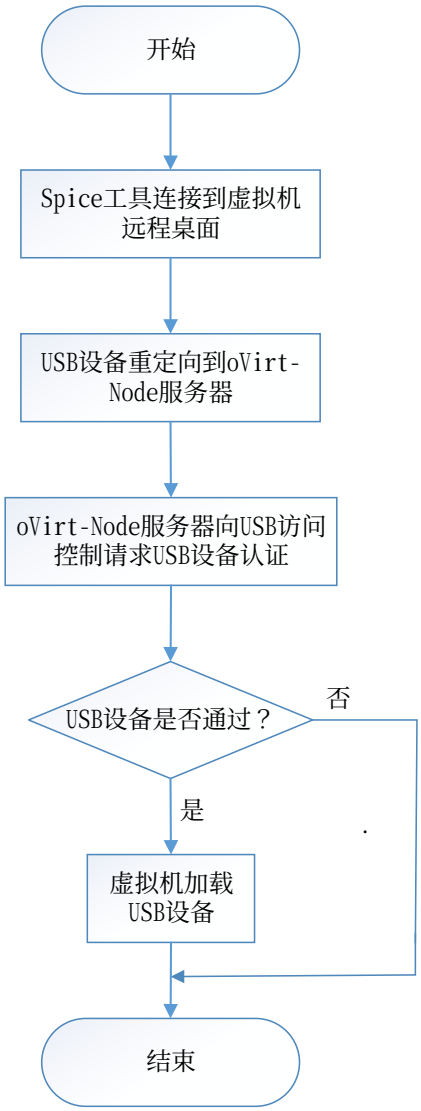


图5-11 USB 设备授权认证流程图

从图 5-11 可以看出，USB 设备授权认证过程如下步骤：

1） 用户通过一台 PC 机使用 Spice 工具远程连接登陆到远程虚拟机桌面上。

2） 在用户在 PC 机上使用 USB 设备时，USB 设备会通过 Spice 协议重定向到虚拟机 oVirt-Node 服务器上。

3） oVirt-Node 服务器在接受到 USB 设备重定向的消息后，会向 USB 访问控制模块发送认证该 USB 设备请求。

4） USB 访问控制模块会通过对应虚拟机的 USB 设备白名单来对该 USB 设备进行认证，如果 USB 设备在白名单中，虚拟机则加载该 USB 设备，否则，虚拟机不加载该 USB 设备。

通过以上的流程，在 oVirt 云平台中达到了对 USB 授权管控的目的，但是还

没有达到管控的实时性。为了实现达到对 USB 设备管控实时性效果，我们添加了 USB 设备监控器对 USB 设备白名单进行监控。USB 设备实时监控流程如图 5-12 所示：

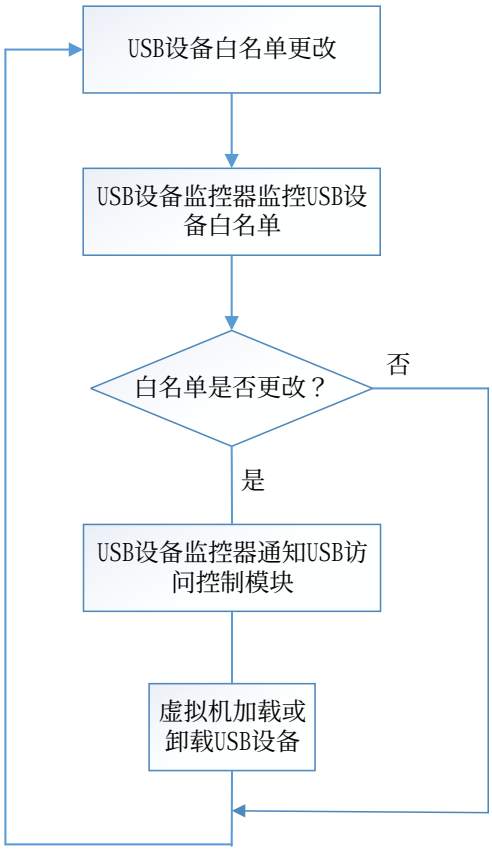


图5-12　USB 设备白名单实时监控流程

在 oVirt 云平台管理人员修改某虚拟机允许访问 USB 设备白名单时，USB 设备监控器会监视到这一变化，如果 USB 设备白名单有改动，USB 设备监控器会通知 USB 访问控制模块对虚拟机中的 USB 设备进行相应的加载或卸载。否则，不做任何处理。虚拟机监控器会不停地监视虚拟机的 USB 设备白名单，从而达到对 USB 设备实时监控的目的，实现了实时性。

通过以上过程我们已经实现了 USB 设备的授权认证和白名单实时监控，能够有效地实时管理 USB 设备的访问权限，下面将详细介绍 USB 设备访问控制的具体过程。

5.3.2 USB 设备访问控制

经过以上的 USB 设备授权认证的设计，下面将详细介绍 USB 设备访问控制的设计与实现。

结合第三章关键技术分析，USB 设备实时管控子系统设计于 Qemu 层中。经过 Spice 协议重定向后，在 Qemu 中使用 USBRedirDevice 数据结构来表示远程 PC 机上的重定向的 USB 设备，如果有多个设备，则是以链表的形式串连起来。

为了实现以 USB 设备 VID 和 PID 作为认证条件, 本文添加了自定义的两个重要 USB 设备访问控制数据结构, 分别是 myusbredirfilter_rule 和 usb_filter_tmp 数据结构。

```
//重定向USB设备过滤
struct myusbredirfilter_rule{
    int device_class;                   //设备类型
    int vendor_id;                      / /供应商ID
    int product_id;                     //产品识别码ID
    int device_version_bcd;             / /设备版本号
    int allow;                          //是否授权，如果为非0，表示授权
};
//USB设备实时管控
struct usb_filter
{
    char pid[5];                        //产品识别码ID
    char vid[5];                       / /供应商ID
    int allow;                          //是否授权，如果为非0，表示授权
    USBRedirDevice *dev;                //重定向USB设备数据结构，包括对USB设备的各种操作
    bool should_reject;                / /是否卸载设备
    struct usb_filter *next;           // 下一个usb_filter结构
};
```

上面的两个数据结构贯穿了 USB 设备实时管控子系统始终, 具有 USB 设备管控信息，为了实现 oVirt 云平台 USB 设备访问实时管控，本文自定义了几个 USB 设备访问控制接口，其自定义主要 USB 设备访问控制接口如下：

```
//向USB设备授权认证模块验证USB设备是否授权
static int check_usb_filter(struct usb_filter_tmp *usb_head,char pid[],char vid[] )
//更新虚拟机USB设备链表
void update_usb_filter(struct usb_filter_tmp *usb_head, const char *usbinfo)
//向虚拟机USB设备链表插入一个USB设备
void insert_usb_filter(struct usb_filter_tmp *usb_head, char pid[],char vid[],
                       USBRedirDevice *dev)
//卸载虚拟机USB设备链表中标记为未授权的设备
void reject_usb_old(struct usb_filter_tmp *usb_head)
```

```
//监控虚拟机所授权的USB设备是否发生改变
void *usb_control_realtime_thread(void *vmuuid)
```

为了实现 oVirt 云平台 USB 设备管控的实时性，我们在 USB 设备访问控制模块中，为每个虚拟机开启了一个 usb_control_realtime_thread 线程，用于实时地监控虚拟机白名单的变化。其 USB 设备实时管控逻辑图如图 5-13 所示：

1）在虚拟机硬件初始化的时候启动 usb_control_realtime_thread 线程，接着开始检查 USB 设备白名单是否发生，如果白名单有改动，则调用 update_usb_filter 进行处理。否则，继续循环执行检查 USB 设备白名单。

2）当 update_usb_filter 被调用时，会检查虚拟机所授权的 USB 设备是否增加，如果增加，调用 insert_usb_filter 接口将新配置的 USB 设备插入到可用链表中。否则，调用 reject_usb_old 接口强制卸载虚拟机正在使用的 USB 设备。

3）执行完成 update_usb_filter 或 reject_usb_old 函数接口后，返回到 usb_control_realtime_thread 线程中循环执行以上过程。

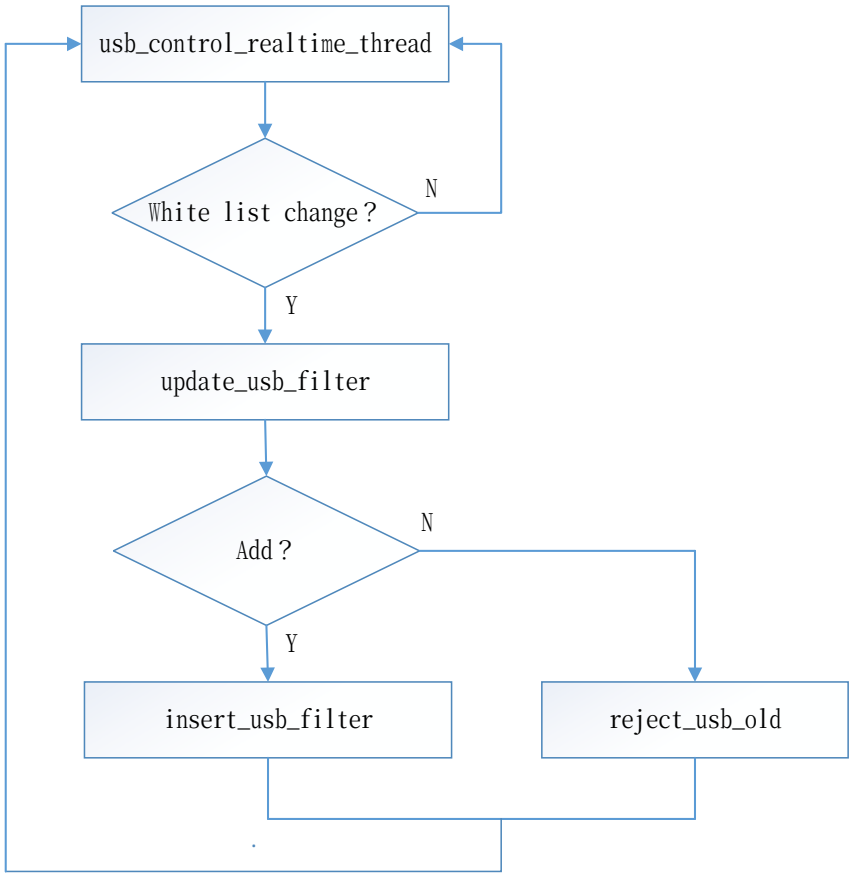


图5-13　USB 设备实时管控逻辑图

至此，我们实现了 oVirt 云平台上的 USB 设备实时管控，达到了实时有效地管理用户 USB 设备效果。

5.4 本章小结

本章结合第三章系统关键技术分析和第四章的设计思路和整体设计框架，对虚拟机磁盘加解密、虚拟机磁盘深度擦除、USB 设备实时管控三大子系统的设计与实现进行了详细阐述，提供了设计与实现的流程图，并给出了实现过程中的部分关键代码。

第六章 系统测试及结果分析

本章对 oVirt 云平台安全加固后进行测试, 验证其是否到达了设计目标的效果, 同时测试其对 oVirt 云平台系统性能影响 ，并对测试结果进行比较分析。

6.1 测试环境搭建

本系统设计到内核级的代码调式，所以需要专门为其搭建测试环境。

1）硬件系统的配置

由于 oVirt 云平台是基于硬件虚拟化技术的，首先处理器需要在硬件上支持硬件虚拟化扩展（Intel VT 和 AMD-V），本测试环境是使用是英特尔的 VT 技术。

首先，进入处理器的 BIOS，VT 的选项通过“ Virtualization Support ”的 Virtualization 选项来查看和设置，如果芯片支持 VT-d 技术，也需打开，它是设备 I/O 的虚拟化硬件支持技术，在 oVirt 云平台高级设备的直接分配会用到，是通过“Virtualization Support”下的 VT for Direct I/O 选项设置。

a）BIOS 中的 VT 选项设置，如图 6-1 所示：

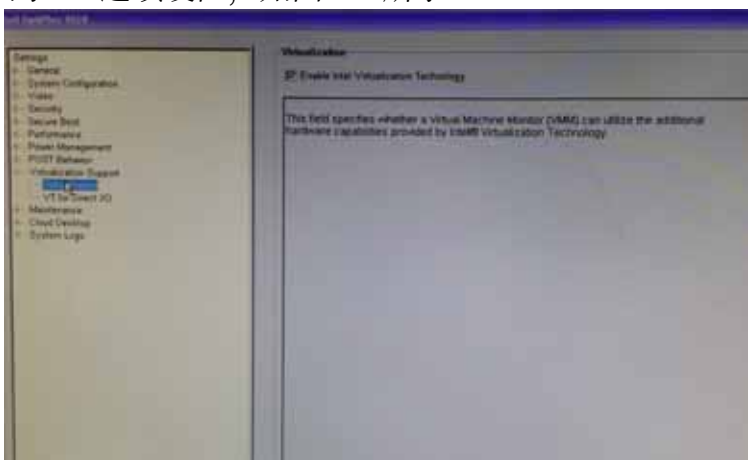


图6-1　BIOS 中的 VT 选项设置

b）BIOS 中 VT-d 选项设置，如图 6-2 所示：

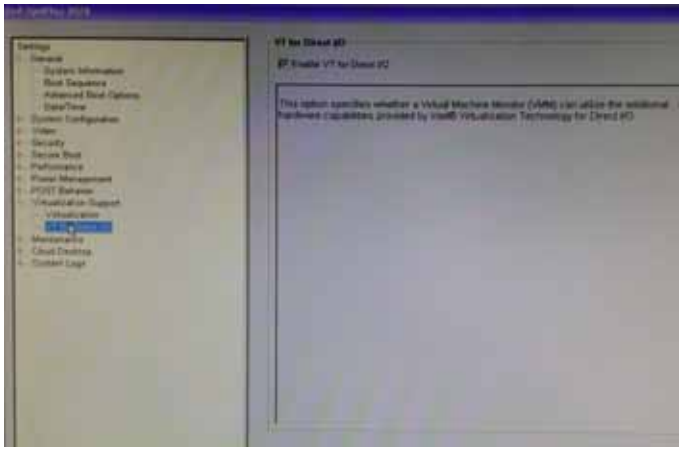


图6-2 BIOS 中的 VT-d 选项设置

2）编译安装带 KVM 的内核

KVM 是作为 Linux kernel 的一个模块存在, 所以只需重新编译下载的 linux 内核， 只是需要配置一下 KVM 相关配置选项，开启支持 KVM 虚拟化。

```
# cd linux-3.14.8
# make menuconfig
```

会显示出配置菜单选项界面，如图 6-3 所示：

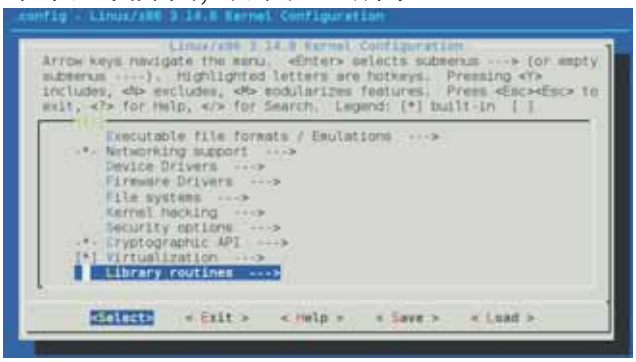


图6-3 make menuconfig 菜单配置界面

选择 Virtualization 进入, 进行详细配置。选中对处理器的支持, 如图 6-4 所示：

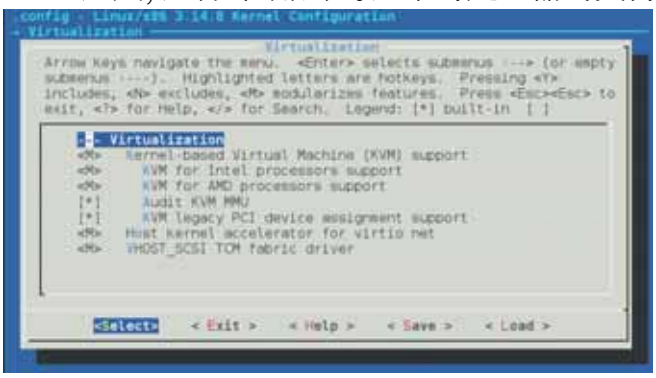


图6-4 Virtualization 中的配置选项

接着编译安装 linux 内核：

```
# make –j20 all
# make modules_install
# make install
```

重启系统, 选择刚才为了 KVM 而编译、安装的内核启动。查看 kvm 和 kvm_intel 两模块是否启动，如果没有启动，用 modprobe 命令一次加载 kvm 和 kvm_intel 这两模块。

```
# modprobe kvm
# modprobe kvm_intel
# lsmod|grep kvm
```

3）编译和安装 qemu

qemu 的配置并不复杂, 进入 qemu 代码文件夹, 通过 configure 文件进行配置。接着编译和安装。

```
# ./configure --enable-debug
Install prefix        /usr/local
BIOS directory     /usr/local/share/qemu
binary directory   /usr/local/bin
library directory   /usr/ local/lib
libexec directory   / usr/local/libexec
include directory   / usr/local/include
config directory    /usr/ local/etc
local state directory    /usr/local/var
Manual directory     /usr/local/share/man
ELF interp prefix    /usr/gnemul/qemu-%M
 ……
VNC support       yes
GUEST_BASE        yes
ATTR/XATTR support yes
Install blobs     yes
KVM support       yes
spice support     yes (0.12.6/0.12.4)
QOM debugging     yes
```

需要注意的是，上面命令输出的 KVM 相关选项配置是否是 yes，VNC support 和 spice support 是否是 yes，接着编译安装 qemu。

```
# make –j 20
# make install
```

4）编译和安装 VDSM

首先，安装 VDSM 所需的编译工具和环境。

```
yum install make autoconf automake pyflakes logrotate gcc python-pep8 libvirt-python
python-devel python-nose rpm-build sanlock-python genisoimage python-ordereddict
python-pthreading libselinux-python python-ethtool m2crypto python-dmidecode
python-netaddr python-inotify python-argparse git python-cpopen bridge-utils
libguestfs-tools-c pyparted openssl libnl libtool gettext-devel python-ioprocess
```

配置网桥, oVirt 云平台中网络资源共享是通过网桥来完成的, 需要创建网桥。

```
vim /etc/sysconfig/network-scripts/ifcfg-ovirtmgmt
```

写入精简的配置信息如下：

```
DEVICE=ovirtmgmt
BOOTPROTO=dhcp
TYPE=Bridge
ONBOOT=yes
```

配置完后，重启网络。

```
# service network restart
```

接着，配置和编译安装 VDSM。

```
$./autogen.sh --system && ./configure    --enable-hooks && make rpm
# cd ~/rpmbuild/RPMS
# yum install --enablerepo=ovirt-master-snapshot-static x86_64/*
noarch/vdsm-xml* noarch/vdsm-cli* noarch/vdsm-python-zombiereaper*
noarch/vdsm-*jsonrpc* noarch/vdsm-python*
# vdsm-tool configure --force
# systemctl start vdsmd
```

5）安装 oVirt-Engine 管理界面

由于 oVirt-Engine 管理界面的修改部分不是由我来完成的, 本文所完成的 oVirt 安全加固的底层功能代码实现。这里就不再使用 oVirt-Engine 源码来编译安装，而使用修改过的 oVirt-Engine 源代码打包成 RPM 包来安装, 会自动安装所需依赖包。

```
# yum install ovirt-engine
# ovirt-engine setup
```

6.2 测试内容

本次测试内容主要包括了两方面：一方面是功能测试，主要测试虚拟机磁盘是否加密成功、虚拟机删除后是否还能在磁盘上找到虚拟机数据、USB 设备是否能够实时管控。另外一方面是性能测试，测试系统经过安全加固后对系统性能的

影响。主要测试内容如下：

1）功能测试

虚拟机磁盘加密测试内容：通过 losetup 和 kpartx 工具对虚拟机的磁盘镜像分区创建映射，接着 mount 挂载相应分区，挂载了虚拟机磁盘镜像文件之后，就可以访问虚拟机文件，找到对应的文件看是否能够解析成功，验证虚拟机磁盘是否加密。

虚拟机磁盘深度擦除测试内容：通过 debugfs 工具查看虚拟机镜像对应磁盘数据块号的数据，对虚拟机深度擦除前后的对应磁盘数据块号上的内容是否相同。验证虚拟机磁盘深度擦除是否成功。

USB 设备实时管控测试内容：远程 PC 机上插入 U 盘，虚拟机桌面是否能加载 U 盘；然后，把 U 盘的 PID 和 VID 分别加入到虚拟机 USB 设备白名单中，再次测试虚拟机桌面是否能够正常加载该 U 盘；最后，验证实时性，在虚拟机桌面正常加载 U 盘的情况下，把 U 盘的 PID 和 VID 从虚拟机的 USB 白名单中删除，结果看到虚拟机桌面中的 U 盘是否能自动卸载。验证其是否能对 USB 设备进行实时管控。

2）性能测试

本次测试主要测试系统经过安全加固后对系统性能的影响，测试虚拟机磁盘加解密前后，对虚拟机磁盘读取速度比较；测试虚拟机磁盘深度擦除的速度快慢。通过测试结果的对比，得出 oVirt 云平台安全加固后对原系统的性能影响结果。

6.3 测试及结果分析

本节将介绍测试过程的一些典型测试用例，并通过 oVirt 云平台安全加固前后的测试数据结果进行对比分析，在实际测试过程中，我们采用了多种的测试用例，由于本节篇幅限制，在下面只介绍测试中主要的测试内容。

6.3.1 功能测试及分析

在 oVirt 云平台安全加固系统中，主要添加了三个功能模块：虚拟机磁盘加密，虚拟机、虚拟机深度擦除、USB 设备实时管控。下面将对这三大功能进行功能性测试，测试其是否满足设计目标的要求。

1）虚拟机磁盘加密功能测试

首先，通过 oVirt-Engine 创建两个虚拟机分别是 vm01 和 vm02，其中虚拟机 vm01 选择磁盘不加密，虚拟机 vm02 选择磁盘加密，并输出自己的密码。如图 6-5 所示：

图6-5 创建虚拟机操作界面

接着，用 Spice 工具连接到虚拟机 vm01 和虚拟机 vm02，分别两台虚拟机桌面上创建纯文本文件，写入文本内容为“This is a te st for virtual machine disk encryption!”，并保存。

然后通过 losetup 和 kpartx 工具对虚拟机的磁盘镜像分区创建映射，挂载分区后，便可以查看虚拟机文件内容是否加密。

```
# losetup –f
# losetup /dev/loop0 1c754e1e-14de-4040-8e5f-0e51a3a7fbaf
#mount /dev/mapper/loop0p2 /mnt
```

首先，看磁盘没有经过加密的虚拟机 vm01，能够从挂载虚拟机文件中正常解析到内容，能够看到先前写入的文本内容。如图 6-6 所示：

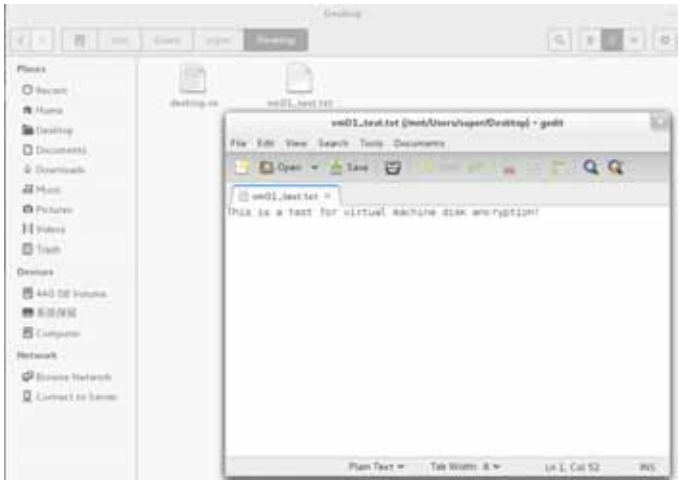


图6-6 虚拟机磁盘解析成功

然后，查看经过磁盘加密的虚拟机 vm02，kpartx 会解析失败，发现不了虚拟机文件内容，文件系统无法识别。

通过上面的比较测试可以看出，虚拟机磁盘加密效果很不错，不仅加密彻底，而且文件系统也无法成功解析，实现了虚拟机磁盘加密功能，虚拟机的数据内容得到了保护，在没有用户密钥的条件下，我们无法得到虚拟机磁盘的用户数据，所以到达了虚拟机磁盘保护的效果。

2）虚拟机深度擦除

虚拟机深度擦除，在删除虚拟机时，对虚拟机磁盘进行复写并删除。下面将以虚拟机 vm01 为例，来验证虚拟机是否深度擦除。

首先，通过 df 命令查看主机文件系统磁盘情况，找到虚拟机镜像文件存放在系统磁盘，接着用 debugfs /dev/mapper/fedora _home 命令，进入 debugfs 模式下。用 debugfs 指令找到虚拟机镜像文件。然后，通过 blocks 命令查看虚拟机镜像文件占有的数据块号，我们选择其中多块来查看其内容，本实验选取数据块 220677099 为例，用 bd 命令查看数据块里的内容。如图 6-7 所示：

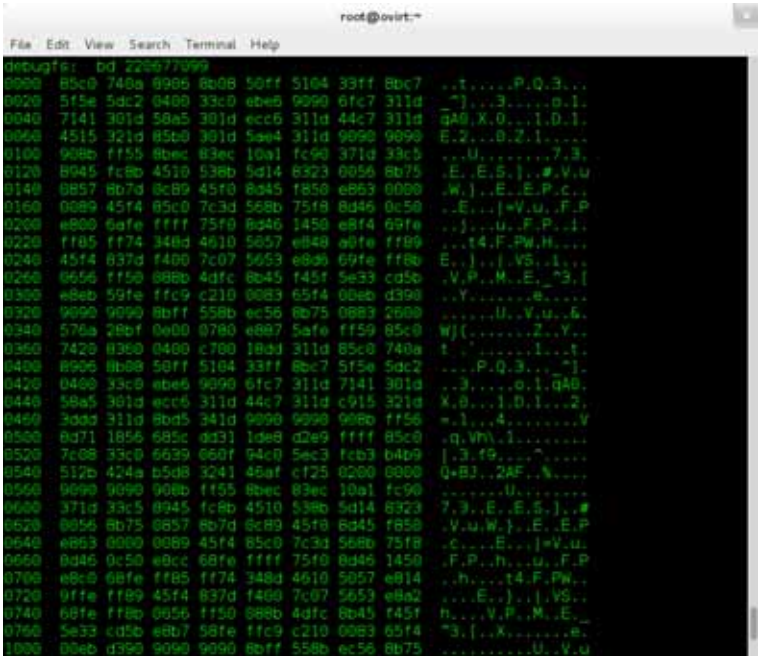


图6-7　虚拟机数据块内容

接着，登录 oVirt-Engine 管理界面，选择删除虚拟机 vm01，并在删除虚拟机界面中勾选深度擦除和一次擦除磁盘方式。如图 6-8 所示：

图6-8　虚拟机删除界面

虚拟机深度擦除完成后，虚拟机 vm01 对应的镜像文件已经被删除，我们再通过 bd 命令来查看虚拟机磁盘文件是否以一次擦除磁盘方式被复写。如图 6-9 所示：

```
debugfs:  bd 220677099
0000  0000 0000 0000 0000 0000 0000 0000 0000  ................
*

debugfs:
```

图6-9　擦除后的虚拟机数据块内容

通过上面的比较测试可以得到，在删除虚拟机后，虚拟机的磁盘内容被复写删除，虚拟机磁盘内容全部被复写为 0X00，达到了虚拟机深度擦除效果。

3）USB 设备实时管控

首先，我们登录到 oVirt-Engine 管理界面，开启一个虚拟机，通过 Spice 工具连接到虚拟机桌面，默认情况下新建的虚拟机的 USB 设备白名单为空，这时候不支持任何设备。我们在远程 PC 机上插入 U 盘，虚拟机桌面没能成功加载桌面。

接着，我们在 oVirt-Engine 管理界面上开启 USB 选项，然后添加本实验测试 U 盘的 PID 和 VID。如图 6-10 和图 6-11 所示：

图6-10　USB 设备管控界面

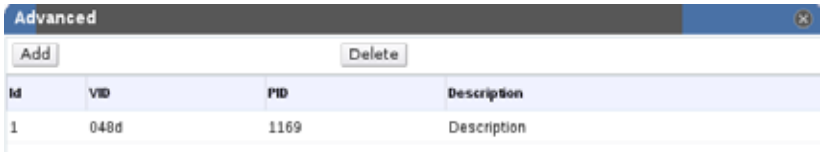


图6-11　USB 白名单设置

然后，再次测试 USB 是否能加载到虚拟机桌面，把 U 盘插入远程 PC 机上，发现 U 盘成功加载到了虚拟机桌面上。接着，再通过 oVirt-Engine 管理界面把 USB 设备白名单中删除掉本 U 盘，发现虚拟机桌面中的 U 盘也被自动卸载了。如表 6-1 所示：

表 6-1 USB 设备实时管控测试结果

虚拟桌面	U盘ID	操作	结果
Vm 01	048d:1169	未加入白名单	加载失败
Vm 01	048d:1169	加入白名单	加载成功
Vm 01	048d:1169	从白名单删除	自动卸载

通过上面测试可得，USB 设备实时管控达到了设计目标，我们能够实时的管控虚拟机的 USB 设备，控制了 USB 设备的使用情况。

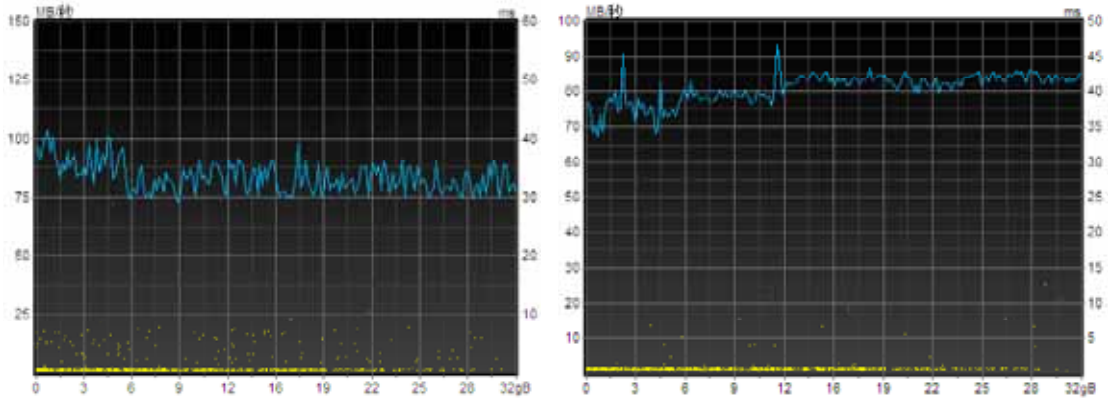
6.3.2 性能测试及分析

oVirt 云平台安全加固主要通过加解密、覆写以及访问控制的方式实现，这些操作必定会给系统性能带来损失。因此，下面我们将对 oVirt 云平台安全加固中的虚拟磁盘加解密、虚拟机磁盘擦除和 USB 设备实时管控对系统的性能影响。

1）磁盘加密后访问性能测试

测试内容：磁盘加密后访问性能测试是在相同的环境下，通过 HDTunePro 工具分别对没有磁盘加密和磁盘加密的两个虚拟机的硬盘读写速率性能测试。测试了硬盘的“ 文件读写速率 ”和“ 基准读取速率 ”两项指标，并对测试结果进行比对。

测试结果：基准读取速率性能测试结果如图 6-12 所示。



虚拟机磁盘加密前　　虚拟机磁盘加密后

图6-12　虚拟机磁盘加密前后基准读取速率对比

测试结果分析：在虚拟机磁盘加密前后基准读取速率对比中，加密前的平均传输速率为 83.6MB/秒，加密后的平均传输速率为 80.9MB/秒。虚拟机磁盘加解密模块对其性能的影响 1-80.9/83.6=0.0323，影响在设计目标的 10%范围内。

2）磁盘文件深度擦除性能测试

测试内容：磁盘文件深度擦除性能测试是在相同环境下，创建了四台大小为 20GB 的虚拟机，依次分别通过全零覆写标准、DoD5220.22-M 覆写标准、BMB21-2007 覆写标准、DoD5220.22-M7 覆写标准这四种擦除标准进行深度擦除，虚拟机擦除完成后，查看不同覆写标准所用时间。

测试结果：不同覆写标准性能测试结果如图 6-13 所示。

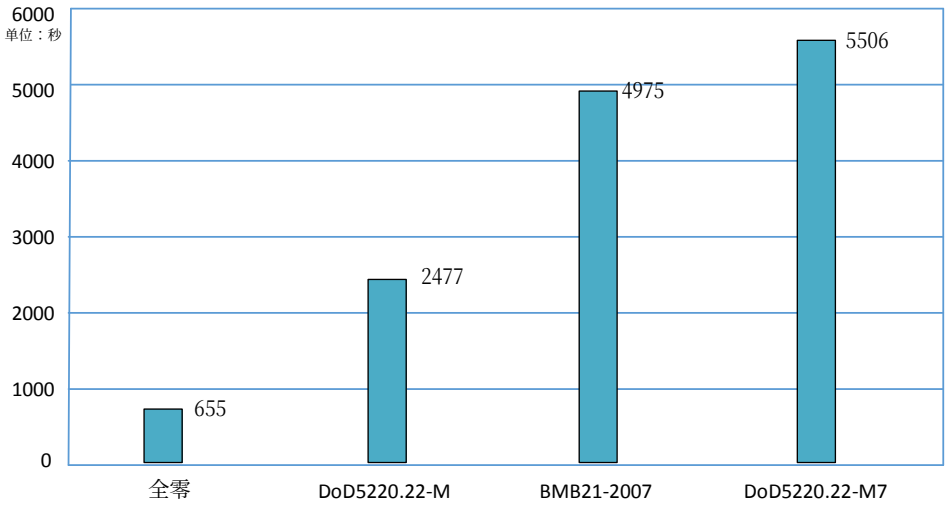


图6-13　不同覆写标准性能测试

测试结果分析：从图 6-13 可以看出，采用不同覆写标准深度擦除所用时间具

有很大差异的，全零的方式覆写能够达到 31M/s，速度还是比较快的，其它方式因擦除次数不同，相应消耗的时间有所增加，总体擦除性能是能够接受的。整个擦除操作相对于用户也是透明的，磁盘深度擦除性能达到了预期目标。

3）USB 设备实时管控后性能测试

测试内容：USB 设备实时管控在相同的环境下，分别对没经 USB 设备实时管控的虚拟机和 USB 设备实时管控的的虚拟机进行性能测试，从远程 PC 机上插入 U 盘，测试拷贝大约 1GB 大小文件的时间，并对测试结果进行比对。

测试结果：对没经过 USB 设备实时管控的虚拟机拷贝 1GB 大小文件耗时 3 分 36 秒，经过 USB 设备实时管控的虚拟机拷贝 1GB 大小文件耗时 3 分 43。

测试结果分析：从上面的测试结果可以看出，USB 设备拷贝速率受到了网络传输速率的制约，因此 USB 设备拷贝文件速率并不是很快。USB 设备实时管控对 USB 拷贝数据传输影响 1-216/223=0.032，影响较小，到达了预期目标。

6.4 本章小结

本章对 oVirt 云平台安全加固系统进行了功能和性能测试。首先介绍了测试环境的搭建，其次分别对虚拟机磁盘加密、虚拟机深度擦除和 USB 实时管控对比测试和分析，说明了 oVirt 云平台安全加固的作用。最后，对虚拟机磁盘加密、虚拟机深度擦除和 USB 设备实时管控对系统的性能影响测试，说明了经过安全加固后的 oVirt 云平台性能影响在可接受范围之内。

第七章 总结与展望

7.1 工作总结

随着云计算产业迅猛发展，云计算应用已经逐渐走进了人们的视野，越来越多的公司和企业都把办公环境转移到云办公环境上。oVirt 云平台正是一个这样的云办公环境的管理平台，是一个基于 Qemu-KVM 的开源 IaaS 项目，受到了广大公司企业青睐。但在 oVirt 云平台设计中没有考虑到一些安全方面的需求，存在用户虚拟机磁盘以明文的形式储存在服务器上，并且在删除虚拟机时，虚拟机用户数据没有得到彻底地删除，用户虚拟机可以任意使用 USB 设备等安全问题，虚拟机用户数据存在严重的安全隐患。

本文正是以 oVirt 云平台安全为主线，分析研究 oVirt 云平台系统，针对虚拟机磁盘加密、磁盘深度擦除和 USB 设备实时管控这三大方面进行安全加固，起到保护用户数据的作用。本文的主要工作如下：

1）研究 oVirt 云平台安全加固相关内容

首先，我们阅读学习了大量有关虚拟化方面的资料，介绍了 KVM 和 Qemu 虚拟化技术，接着，研究学习了磁盘加解密算法以及目前常用的磁盘数据擦除技术，为后面的研究设计打下了扎实的基础。

2）研究分析 oVirt 云平台系统

有了理论基础铺垫后，我们结合 oVirt/Qemu/KVM 源码，分析 oVirt 云平台框架、磁盘存储管理、虚拟机镜像文件加密、虚拟机镜像深度擦除和 USB 设备实时管控等关键技术。并通过实验证明了 oVirt 云平台中存在的安全隐患，经过大量地源码分析和不断地实验，最终找到了安全加固的切入点。

3）oVirt 云平台安全加固方案的制定和实现

结合 oVirt 云平台关键技术的分析，制定了 oVirt 云平台安全加固方案。根据所制定的方案，我们分别实现了虚拟机磁盘镜像文件加密、虚拟机磁盘深度擦除和 USB 设备实时管控三个子系统，并且对其进行了详细介绍。

4）测试 oVirt 云平台安全加固的有效性

最后，通过对 oVirt 云平台安全加固前后进行功能和性能方面的测试，并对测试结果进行对比，验证了本 oVirt 云平台安全加固的有效性，对原 oVirt 云平台系统的影响在合理范围之内，达到了预期的效果。

7.2 工作展望

随着云平台如火如荼的发展，越来越多的公司和企业把传统的办公转移到云办公环境上，但是目前很多云平台还存在一些安全隐患，公司用户不愿意将重要数据存放在云平台中。为了保护云平台中用户数据安全，本文在研究分析基于oVirt/Qemu/KVM 云平台系统基础上，通过源码分析，设计和实现了 oVirt 云平台系统的安全加固。虽然在 oVirt 云平台中实现了虚拟机磁盘加密、虚拟机磁盘深度擦除和 USB 设备实时管控，在一定程度上起到了保护虚拟机用户数据的作用，但是对 oVirt 云平台安全加固是没有终点的，因此需要对 oVirt 云平台安全进一步研究。

总结 oVirt 云平台安全加固中的不足，根据目前现状，在此，我们做出了以下两点工作展望：

1）安全加固功能部署的优化

在本文中对 oVirt 云平台安全加固都是基于固定在某一版本上实现的，如果随着 oVirt 云平台版本更新，需要修改新版本相应的代码，这一过程的实现相当繁琐和复杂。下一步可以通过对新版本 oVirt 云平台以打补丁的方式增加安全加固功能，从而降低了新版本安全加固部署难度。由于时间和精力有限，本文并没有对此进行优化。

2）oVirt 云平台安全问题进一步研究

本文中的 oVirt 云平台安全加固解决方案虽然能够在一定程度上保护用户数据安全，但是黑客的攻击手段是变化多端的，设计一套绝对安全的 oVirt 云平台系统是不可能的，因此，还需要对 oVirt 云平台安全问题进一步研究。

致　谢

在论文即将完成之际，也标志着我在电子科技大学的三年研究生生涯即将结束。在此，我由衷地感谢这三年的时光里，在学习和生活中给予我帮助和支持的人们。

首先，我要感谢我的恩师刘丹副教授，在三年研究生学习中，给我们创造了良好的科研环境，在我们遇到学习和科研问题时，给予我们耐心的指导，传授给我们解决处理问题的思路，并悉心指导了论文选题和写作，使我们在科研能力和论文写作方面都有了质的提升。从刘老师那学习来的处理解决问题的能力和为人，使我终生受益。

同时，我也要感谢周明老师，在研究生三年科研工作中，对我们科研项目和论文指导提出了许多宝贵的建议，让我们在期间得到了很好的锻炼。

另外，感谢唐道平、张国伟师兄在学习和科研上对我的帮助，感谢徐亚运、王迪、彭春红、杨川等项目上的搭档，在研究生阶段给予我了很多帮助。

特别还要感谢我的父母这些年对我无微不至的关怀， 是他们的支持和鼓励才使我走到今天。

最后，我要对评阅本论文而付出宝贵时间和辛勤劳动的评审委员会老师们表示衷心的感谢。

参考文献

[1] Armbrust M, Fox A, Grith R, et al, Above the clouds: A Berkeley View of Cloud Computing[R] ,UCB/EECS-2009-28, Berkeley, USA : Electrical Engineering and Computer Sciences,University of California at Berkeley, 2009

[2] Vaquero L, Rodero-Marino L, Caceres J, et al, A break in the clouds : towards a cloud definition [J]. SIGCOMM Computer Communication Review, 2009, 39(1) : 50-55

[3] M Armbrust, A Fox, RG riffith, et al, A view of cloud compting[J]. Commun ACM, 2010, 53(4) :50-58

[4] Feng DG.Zhang M.Zhang Y.Xu Z.Study on cloud computingsecuritu[J].Journal of Software, 2011, 22(1):71-83

[5] KVM Sources and Documentation[EB/OL]. http://www.linux-Kvm.org/

[6] oVirt Sources and Documentation[EB/OL]. http://www.ovirt. org/.

[7] 王继刚，郑纬民，滕志猛，钟卫东. 虚拟化环境下的 USB 设备访问方法[J].计算机应用. 2011,31(5):1439-1442.

[8] 孔楠. 基于云计算平台的商业服务模式研究[D].上海外国语大学，2010

[9] 刘鹏．云计算[M].北京：电子工业出版社，2010

[10] Mell P and Grance T. The NIST Definition of Cloud Computing[R/OL]. http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[11] Cloud Security Alliance. Security Guidance for Crit ical Areas of Focus i n Cloud Computing V3.0[R]. The Cloud Security Applicance, 2011.

[12] 李玮. 云计算安全问题研究与探讨[J]. 电信工程技术与标准化，2012，04:44-49.

[13] 崔泽永,赵会群. 基于 KVM 的虚拟化研究及应用[J]. 计算机技术与发展,2011,06:108-111+115.

[14] 李胜召,郝沁汾,肖利民.KVM 虚拟机分析[J]. 计算机工程与科学，2008，30(A1):129-132.

[15] Kong Jinzhu. A Practical Approach to Improve the Data Privacy of Virtual Machines[C]. The 10th International Conference on Computer and Information Technology. Bradford, UK, 2010, 936-941

[16] Joshua Gold. Protection in the Cloud:Risk Management and Insurance for Cloud Computing[J]. 12J. on Internet Law. L. 15, 2012

[17] Xiao Zhang, Hong-tao Du, Jian-quan Chen, et a l. Ensure Data Security in Cl oud Storage. Network Computing and Information Security (NCIS), 2011,284-287

[18] Liu Hao,Dezhi Han. The study and design on secure-cloud storage system. Electrical and Control Engineering (ICECE), 2011,5126-5129

[19] Sengupta S., Kaulgud V., Sharma V.S. Cloud Computing Securit—Trends and Research Directions. Services (SERVICES), 2011,524-531

[20] Balduzzi M, Zaddach J,Balzarotti D, et al. A Security analysis of amazon's elastic compute cloud service[C]//Proceeding of the 27th Annual ACM Symposium on Applied Computing. ACM,2012: 1427-1434.

[21] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory[C]. In the Sixth USENIX Security Sympo-sium Proceedings, San Jose, California, July, 1996

[22] Peter Bennison. Data security for surplus or end of life hardware[C]. 2004 NYS Cyber Security Conference: New York, April21-22, 2004.

[23] Carbone Martim, Sharif Monirul I, Lee Wenke. Wenke lee; Lares: An Architecture for Secure Active Monitoring Using Virtualization Security and Privacy [C]. IEEE Symposium on 18-22 May 2008 Page(s): 233-247.

[24] Artem Dinaburg, Paul Royal, Monirul Sharif, et al. Ether: malware analysis via hardware virtulization extensions[C]. In Procee dings of the 15th ACM conference on Computer and communications security. Oct, 2008, 15-18.

[25] Fengzhe Zhang, Jin Chen, Haibo Chen, et al. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization[D]. Fudan University, 2011.

[26] 胡光永. 基于云计算的数据安全存储策略研究[J]. 计算机控制与测量，2011.19(10): 2539-2541

[27] 林秦颖,桂小林,史德琴,王小平. 面向云存储的安全存储策略研究[J]. 计算机研究与发展,2011,S1:240-243.

[28] Top Threats Working Group. The notorious nine cloud computing top threats in 2013[EB/OL]. http://www.chinacloud.cn/upload/2013-03/13030711513081.pdf.

[29] 冉旭,钟鸣,程放,许勇,刘佳. 云计算安全防护体系研究[J]. 信息通信,2013,01:81-82.

[30] The Raw Image Format[EB/OL]. http://en.wikipedia.org/wiki/Raw_image_format

[31] Gnome Org. The QCOW2 Image Format[EB/OL]. https://people.gnome.org/~markmc/qcow-image-format.html

[32] M. Schnjakin, R. Alnemr, C. Meinel. A security and high-availability layer for cloud storage[C]. In Proceeding(s) of the 2010 international conference on Web information systems engineering. Hong Kong, China, 2010:449-462.

[33] SPICE Sources and Documentation [OL]. http://ww w.cnnvd.org.cn/

[34] Damiani E, et al. Key management for multi-user encrypted databases //Proc of the 2005 ACM Workshop on Storage Security and Survivability. New York: ACM, 2005:81-86.

[35] Hogben G Privacy, Security and Indentity in the Cloud[C] ENISA OASIS/EEMA Identity conference, 2010:24-31

[36] Cloud Security Alliance Security Guidance for Cri tical Areas of Foc us in Cloud Computing V2.1. Technical report[R]. Cloud Security Alliance, 2009

[37] Archer J, Bo ehme A, Cullin ane D, Ku rtz P. Top Threats to Cloud Computing V1.0[R]. Technical Report. Cloud Security Alliance, 2010

[38] 胡光永. 基于云计算的数据安全存储策略研究[J]. 计算机测量与控制, 2011,10:2539-2541.

[39] 张艳,李舟军,何德全. 灾难备份和恢复技术的现状与发展[J]. 计算机工程与科学,2005,02:107-110.

[40] 王继刚,郑纬民,滕志猛,钟卫东. 虚拟化环境下的 USB 设备访问方法[J]. 计算机应用, 2011,05:1439-1442.

攻硕期间取得的研究成果

[1] 林雪峰，邵长庚，刘丹. 一种基于 XEN 平台桌面协议的隔离方法[P]. 中国，发明专利，201310278593.3，2013 年 7 月

电子科技大学
UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE