基于虚拟机监控技术的可信虚拟域

邢 彬, 韩 臻, 常晓林, 刘吉强

北京交通大学 计算机与信息技术学院 北京 中国 100044

摘要 针对云计算中带内完整性度量方案存在的依赖操作系统安全机制、部署复杂和资源浪费等问题, 提出了基于虚拟机监控技术的带外完整性度量方案, 可用于为云计算基础设施即服务(IaaS)的租户提供可信的虚拟域。该方案包括域外监控方案和域内外协同监控方案两部分。前者可对开源 Linux 虚拟域实现完全透明的完整性度量, 同时弥补了其他基于系统调用捕获的域外方案所存在的不足。后者将实时度量与预先度量方法、域内度量与域外度量方法、细粒度的注册表度量方法和基于系统调用的域间信息传输方法相结合, 可对不完全开源的 Windows 虚拟域实现完整性度量。实验证明了方案的度量能力是完备的、性能影响是可接受的。

关键词 云计算; 虚拟化; 虚拟机监控; 可信计算; 完整性度量
中图法分类号 TP309.1

Trusted Virtual Domain based on Virtual Machine Introspection Technology

XING Bin, HAN Zhen, CHANG Xiaolin, LIU Jiqiang

School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China

Abstract In-band integrity measurement schemes in cloud computing have some weak points, such as OS security mechanism dependency, deployment complicacy, and computing resource waste. In this paper, an out-of-band integrity measurement scheme based on virtual machine introspection technology is proposed, which can be used for providing trusted virtual domains for the tenants of Infrastructure as a Service (IaaS). This scheme consists of two parts. One is Out-of-the-Box Monitoring sub-scheme, which can not only achieve fully transparent integrity measurement for Linux virtual domains, but also make up the shortcomings of the hypervisor-based schemes which use system call interception method. The other is In-and-Out-of-the-Box Monitoring sub-scheme, which is composed of real-time and beforehand measure methods, in-the-box and out-of-the-box measure methods, fine-grained registry measure method and system-call-based inter-domain information transmission, and has the ability to measure the integrity of Windows virtual domains. Evaluation experiments show that the proposed scheme has complete measurement ability as well as acceptable performance impact.

Key words cloud computing; virtualization; virtual machine introspection; trusted computing; integrity measurement

1 引言

云计算是一种新兴的基于互联网的计算方式和服务模式, 是多种计算技术和网络技术融合的产物。凭借便利、易扩展和按需计费等优点, 越来越多的企业和公众开始接受和使用云计算, 但是对云计算的安全仍存有疑虑。云计算的服务模式是一种资源租用模式, 租户在租用云服务提供商(Cloud Service Provider, CSP)的资源时, 需要将自己的数据和程序等存储在云端。在这种模式下, 租户失去了对云计算平台中资源的绝对控制权, 无法知道 CSP 是否会忠实按照约定的条款处理自己的数据和程序。同时作为一种信息系统, 云计算平台本身也会存在安全漏洞和隐患。当云平台被攻击时, 攻击者同样可能窃取或篡改租户的数据和程序。云计算中的安全问题已经成为了云计算推广和使用中最大的障碍, 如何保证云服务的可信、保护云中数据和程序的安全已经成为云计算发展中亟待解决的问题。

本文将主要讨论在云计算中为租户提供可信的基础设施即服务(Infrastructure as a Service, IaaS)。

通讯作者: 邢彬, 博士研究生, Email: xingbin@bjtu.edu.cn; 韩臻, 博士, 教授, Email: zhan@bjtu.edu.cn。
本课题得到国家自然科学基金项目(No.61572066); 高等学校博士学科点专项科研基金(No.20120009110007); 发改委信息安全专项(No.[2013]1309)资助。收稿日期: 2015-11-23; 修改日期: 2015-12-16; 定稿日期: 12016-01-05

IaaS 服务能够为租户提供虚拟域的租用服务，租户能够在所租用的虚拟域内安装和运行所需的操作系统和数据库、部署自己的应用和服务，同时还可以较为灵活地按需设置 CPU、内存、磁盘和网络连接的性能或容量等等。可信的 IaaS 服务，就是要使租户租用的虚拟域能够按照租户自己期望的方式运行。

目前关于可信虚拟域的研究有很多，文献[1, 2]都提出为虚拟域提供虚拟安全芯片[3]以实现对可信计算[4]支持，再在此基础上为虚拟域提供安全性增强方案，例如将原有为普通计算机设计的完整性度量框架应用于虚拟域中，以此建立一条从虚拟安全芯片到虚拟域应用程序的信任链。IMA[5]和 PRIMA[6]是目前研究较多的完整性度量框架，它们可以实现对 Linux 操作系统启动和运行过程中加载的内核、内核模块、可执行程序等文件的完整性度量。在信任链建立后，租户将可以通过远程证明(Remote Attestation)协议根据信任链验证虚拟域的完整性。虽然将传统的安全方案应用到虚拟域中能够解决一定的安全问题，但是这种做法会面临以下问题:

(1) 带内管理。租户通过远程证明协议验证虚拟域的完整性时，需要在虚拟域中部署远程证明服务器以接收租户发起的挑战(Challenge)。而该服务器需要使用虚拟域内的网络连接，因此这种管理方案属于带内管理，即管理控制流和业务数据流使用同一网络链路。在这种情况下，恶意程序和虚拟域自身的配置都可能造成虚拟域对外的网络中断，导致租户无法进行有效地管理或获得有效的结果。

(2) 依赖系统安全机制。IMA 和 PRIMA 等完整性度量框架都需要依赖操作系统自身提供的安全机制，例如 Linux 安全模块(LSM)和 Windows 的相应安全接口等等。一旦有 Rootkit 破坏了操作系统内核或这些安全接口，这些安全机制将可能被旁路，从而导致上述方案失效。

(3) 部署复杂。在虚拟域内部署完整性度量框架时都需要重新编译内核或设置内核参数，同时还需要 CSP 为虚拟域提供虚拟安全芯片，这增加了租户的使用难度。

(4) 资源浪费。在云计算 IaaS 服务中，主机运行着多个虚拟域，每个虚拟域内都需要部署远程证明服务器，造成在同一主机中部署了同一服务器的多个实例，产生资源浪费。如果由 CSP 统一提供远程证明服务则只需运行一个实例，从而节省资源。

产生上述问题的根源在于租户的操作仅限于虚拟域内部，无法对虚拟域进行带外安全管理；而 CSP 也没有能够为租户提供统一的、带外的完整性度量服务。事实上虚拟机技术为 CSP 和租户提供了带外管理的可能性，而且目前已经有一些基于虚拟化技术的安全研究，例如在虚拟域外检测虚拟域中应用程序的完整性[7-10]、文件系统的完整性[11]、进程的执行状态[12]、Rootkit 的行为[13]、恶意程序[14, 15]等方面。其中关于完整性方面的研究主要关注操作系统中可执行程序和内核模块等文件的完整性，缺乏对以下三类文件的考虑:

(1) 系统配置文件: 这类实体不具有可执行性，但是对这类文件的修改可能会影响操作系统和可执行程序的行为;

(2) 脚本和字节码等文件: 这类实体同样不具有可执行性，但是可以被脚本解析器、Java 虚拟机等程序加载运行，应当和可执行程序一样被度量;

(3) 不触发系统调用的文件: 这类实体无法被域外方法直接捕获，例如在内核空间中加载的可执行程序等。

为了解决上述问题，我们在文献[16]中给出了一个 OB-IMA 完整性度量方案，在 Xen 虚拟机中实现了对使用 Linux 操作系统的半虚拟域进行域外完整性度量。在 OB-IMA 方案的基础上，本文提出了域外监控方案和域内外协同监控方案，分别用于度量使用 Linux 和 Windows 操作系统的虚拟域的完整性，并详细介绍了这两种方案在 KVM[17]虚拟机中的实现，还通过实验对方案的完备性和性能等进行了验证。同时，这些方案的原理同样也适用于其他虚拟机架构。本文主要关注虚拟域的完整性，假设宿主机和 KVM 是安全和可信的。

域外监控方案的特点是对虚拟域中的 Linux 操作系统和应用程序完全透明，不需对虚拟域内做出任何修改，而且能够对虚拟域中可执行程序、系统配置、脚本以及不触发系统调用的文件进行度量。域内外协同监控方案只需要在虚拟域中做出很小的修改，就能够度量使用不完全开源的 Windows 操作系统的虚拟域的完整性，同样能够实现对虚拟域中上述类型的文件进行度量，而且还支持对注册表配置进行细粒度的检测。

本文的主要贡献有:

(1) 进一步分析了基于系统调用的完整性度量方案所存在的缺陷，实现了对 Linux 虚拟域完全透明的基于域外监控的完整性度量方案。

(2) 提出了预先度量的方法和使用范围，提出了对注册表的细粒度完整性度量方法，利用预先度量和实时度量相结合、域内度量和域外度量相结合等方法实现了对 Windows 虚拟域的完整性度量。

(3) 提出了基于系统调用的域间信息传输方法。

在接下来的内容中，第 2 节将介绍和本文有关的研究背景和相关工作，第 3 节讨论实现可信虚拟域时需度量的实体、可能的度量方法和安全目标，第 4 节和第 5 节将分别介绍域外监控方案和域内外协同监控方案，第 6 节将对全文进行总结。

2 研究背景和相关工作

2.1 虚拟机和虚拟域

虚拟化是指通过资源整合、抽象与仿真等手段进行资源模拟的方法或技术，并且虚拟化得到的资源应具有与相应真实资源相同或相似的接口和使用方法。类似的，虚拟机技术是一种利用计算资源和虚拟化方法模拟计算机的技术。它通过统一管理、分配和调度计算平台、集群中的 CPU、内存、磁盘和网络等计算资源，将这些资源抽象并分割为若干个彼此间隔离且独立的计算域。这些通过虚拟机技术得到的计算域被称作虚拟域(Virtual Domain，又称客户域，即 User Domain、Domain U、DomU)。每个虚拟域都可以像普通计算机一样进行计算、存储或访问网络。资源的动态分配和调整能力使虚拟机技术在云计算中得到了广泛的应用。

KVM(Kernel-based Virtual Machine)是开源虚拟机之一。如图 1 所示, KVM 由宿主机(Host)操作系统中的 KVM 内核模块、硬件仿真程序、管理工具和被创建的一个或多个虚拟域组成。宿主机即实际物理存在的计算机，其操作系统内核中的 KVM 模块是 KVM 虚拟机的一个重要组成部分，它本身不负责设备的模拟，而是为宿主机用户空间提供了一系列接口，提供虚拟域的分配内存、I/O 访问处理等功能。经过修改的 QEMU(Quick EMUlator)硬件仿真程序是 KVM 虚拟机的另一个组成部分，它能够为虚拟域模拟 x86、amd64 等不同的 CPU 架构和多种硬件设备并通过调用 KVM 模块的接口以实现相应的功能。KVM 管理工具在宿主机中，用于管理虚拟域和 KVM 配置。虚拟域运行在硬件仿真程序之上，云计算 IaaS 服务中为租户提供的服务就是虚拟域的使用权。

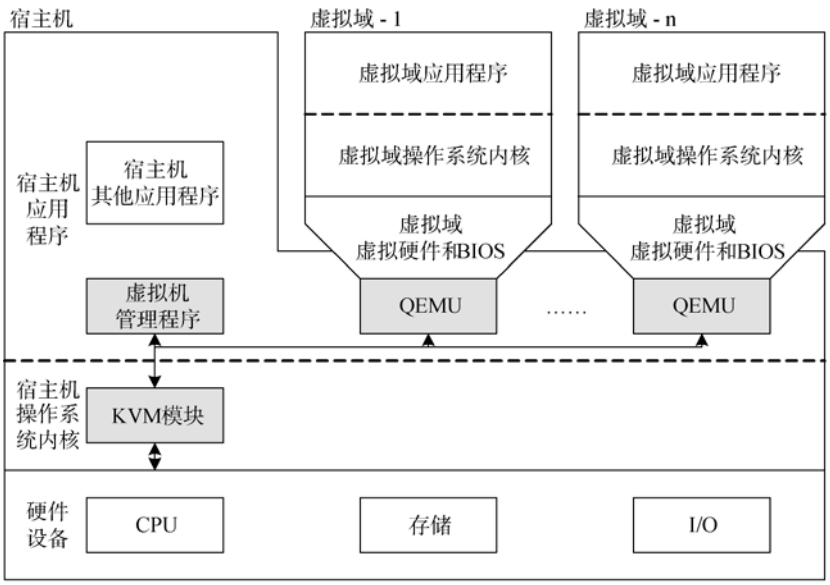

图 1　KVM 虚拟机结构

2.2 域内、域外和域内外协同方法

2.2.1 域内方法

在云计算 IaaS 服务中，租户所租用的计算资源就是虚拟域。在虚拟机隔离机制的保护下，各个虚拟域可以在同一个计算平台中独立运行且互不干扰。因为虚拟域内的操作系统和普通的操作系统通常没有颠覆性的差异，所以原操作系统内的安全机制能够直接或经移植后用于虚拟域中。这类方法可被称为“域内(in-the-box)方法”。原有普通操作系统中的杀毒软件、软件防火墙等都属于域内方法。

域内方法工作在操作系统内，在系统接口的支持下可以获得具有丰富语义的信息，这将有益于分析系统内的行为和事件。但是恶意程序能够通过对域内方法、操作系统及其组件等发起攻击而导致域内方法失效或被旁路。

2.2.2 域外方法

KVM 虚拟机负责虚拟域 CPU、内存等计算资源的分配、控制和调度，而宿主机中的工具又可以管理

KVM 虚拟机和各个虚拟域, 因此 KVM 虚拟机和宿主机中的安全机制能够在虚拟域中发挥一定的作用。由于这些机制部署在虚拟域外, 因此可被称为“域外(out-of-the-box)方法”。

和域内方法不同的是, 域外方法不需要对虚拟域内的操作系统和应用程序做任何的修改, 即对虚拟域是完全透明的。在虚拟机隔离机制的保护下, 虚拟域内的恶意程序无法破坏域外方法, 这就使域外方法不会被旁路, 因此具有更好的安全性。但是域外方法也存在一定的局限, 例如虚拟机带来的语义空白(Semantic Gap)问题。语义空白问题是由虚拟域内外的语义视图不一致导致的, 例如, 如果无法得到虚拟域内操作系统的页表信息, 就无法从内存中直接获得有意义的数据, 而即使获得虚拟域内的页表, 内存地址也需要在虚拟域视图内的虚拟地址、物理地址与虚拟机监视器视图内的虚拟地址之间进行转换。目前一些研究已经取得了很好的成果并得到了较为广泛的应用, 例如针对 VMware 的语义视图重构[14]和针对 Xen 虚拟机的 XenAccess[18]等, 这为域外方法的实现提供了很大的便利。

2.2.3 域内外协同方法

域内方法和域外方法除了各自独立工作之外, 也可以相互配合协同工作, 本文将该类方法称为“域内外协同方法”。常见的应用包括在域内外同时捕获同类信息并对比以检测系统是否存在异常, 或者分别捕获不同的信息实现互补。例如, Wang 等人提出的 NumChecker[19]就是一种通过对比域内外的硬件计数器来检测 Rootkit 的方法。

域内外协同方法包含了分别在域内和域外工作的模块, 域内模块可以获得富有语义的信息, 而域外模块能够观察到域内模块是否工作在正常状态, 实现域内方法和域外方法的优势互补。但是域内外协同方法也存在一定的缺陷, 例如不能对虚拟域完全透明, 需要保证域间通信的安全, 同时还需要对虚拟机做出较大的改造。

表 1 对上述三种方法进行了总结和对比。

表 1 域内方法、域外方法和域内外协同方法对比

	原理	优点	缺点
域内方法	利用操作系统已有的安全机制 或通过修改操作系统增加新的安全机制	语义信息丰富	易被攻击或旁路
域外方法	利用虚拟化技术在操作系统外实现安全机制	对虚拟域操作系统透明	需虚拟化支持, 语义空白导致语义缺失
域内外协同方法	结合虚拟化技术和操作系统自身的安全机制, 在操作系统内外同时实现安全功能	语义信息丰富, 攻击易被发现	修改范围较大, 涉及组件较多

2.3 可信计算

本文所讨论的可信计算限于可信计算组织(Trusted Computing Group, TCG)提出的概念, 即如果一个实体的行为总是以所期望的方式达到预期的目标, 那么该实体就是可信的。因此, 可信计算的目的是保护和度量实体的完整性。为了建立可信的计算平台, TCG 提出了改造计算机体系结构、增加硬件安全芯片等方法, 实现对计算平台完整性的度量、存储和报告等功能。安全芯片主要包括随机数发生器、密钥生成与管理、加解密、安全存储等功能, 使记录的计算平台完整性状态真实、准确和可验证。TCG 提出的可信平台模块(Trusted Platform Module, TPM)[20]和我国提出的可信密码模块(Trusted Cryptography Module, TCM)等安全芯片都具有这样的功能。

如果计算平台是可信的, 就需要该计算平台的行为与其设计预期相符。但是计算平台涉及大量的实体, 因此判断该平台是否可信并不容易。以普通计算机为例, 如果该计算机是可信的, 需要计算机中的硬件及其固件、引导程序、操作系统和应用程序等都是可信的, 因此硬件设备和型号、软件程序和动态链接库等等都需要与预期相同。为了使验证过程更加清晰和有效, TCG 提出通过建立信任链的方法进行判断: 从计算平台中初始且可信的信任根出发, 在每一次平台状态转换时, 如果这种转换能够让信任传递下去而不被破坏, 那么计算平台就始终是可信的。这需要首先在计算机中建立一个可信的信任根, 再按照计算平台启动和实体加载的顺序建立一条覆盖硬件、固件、引导程序、操作系统内核直到应用程序的信任链, 并在建立过程中逐级进行完整性度量, 将度量结果扩展到安全芯片的平台配置寄存器(Platform Configuration Register, PCR)中, 同时将度量结果记录到存储度量列表(Stored Measurement Log, SML)中。

PCR 寄存器是一种特殊的单向寄存器, 它的数值会在两种情况下发生变化: 一是在安全芯片随计算平台加电而初始化时被清零; 二是通过扩展操作,

如下式所示,

$$PCR\_new = Hash(PCR\_old \mid digest)$$

其中 *PCR_old* 和 *PCR_new* 分别表示“扩展”操作前后的 PCR 寄存器的值, *Hash* 表示安全芯片中的散列算法, *digest* 表示被扩展的实体度量结果, “|”表示连接运算。由于 PCR 寄存器的值不能被直接设置为特定的数值, 因此 PCR 寄存器能够真实记录信任链的建立过程。如果有攻击者修改了实体或在系统中加载了新的实体, 则相应实体的度量结果将是异常的, 同时 PCR 寄存器的值将必然和预期不同, 而且该值是无法被篡改的。因此, 重要的数据可以根据 PCR 寄存器的值进行密封, 使这些数据在平台状态不可信时无法解封, 从而保证数据的安全。

远程证明协议同样需要使用 PCR。这一协议将使挑战者有能力验证计算平台的完整性状态, 判断其是否可信。远程证明协议的核心是将 PCR 真实、安全地发送给挑战者。如果计算平台中各个实体具有固定的加载顺序且这些实体的完整性没有改变, 那么计算平台每次完成实体加载后的 PCR 将是相同的, 挑战者可以直接根据 PCR 判断计算平台是否可信; 如果计算平台中的实体加载与否与执行的任务相关, 加载顺序又可能因系统的调度而不同, 那么挑战者在判断计算平台完整性时, 还需要通过远程证明协议获得 SML 并逐项检验 SML 中的每一个实体的完整性记录, 同时根据 PCR 的扩展计算方法对 SML 列表中的记录进行计算, 检验计算结果和实际的 PCR 是否一致。

2.4 相关工作

第 1 节曾提到了与本文相关的一些研究工作, 表 2 对这些研究进行了总结和对比。

表 2 相关工作对比

方案	主要工作	原理	实验环境	和本文工作对比
方案[7]	对虚拟域的完整性进行度量, 阻止不可信的应用访问内存和磁盘数据	系统调用捕获等	Xen, Linux	未考虑配置文件和不触发系统调用的程序的完整性
HIMA[8]	在 VMM 中对虚拟域的内存和应用程序的完整性进行监控和度量以应对 TOC-TOU 攻击	系统调用捕获、内存保护等	Xen, Linux	未考虑配置文件和不触发系统调用的程序的完整性
SecVisor[9]	将系统内核修改为轻量级的虚拟机监视器, 实现对内核空间代码执行的完整性保护	内存管理和指令控制等	Linux	对系统改造较大, 关注重点为内核空间的程序
方案[10]	对虚拟域操作系统启动和运行时的关键文件进行完整性度量	系统调用捕获等	Xen, Linux	未考虑配置文件和不触发系统调用的程序的完整性
OB-IMA[16]	对虚拟域的完整性进行度量, 还特别对不触发系统调用的文件和配置文件等进行了分析	对系统调用进行捕获和分析等	Xen, Linux	工作在 Xen 虚拟机, 不支持 Windows 操作系统
XenFIT[11]	对虚拟域内的文件操作进行监控以保护虚拟域文件系统的完整性	系统调用断点处理等	Xen, Linux	需要修改虚拟域操作系统内核, 研究内容有所不同
方案[12]	利用虚拟化和进程嫁接技术对虚拟域内的进程进行监控	内存控制等	KVM, Linux	研究内容不同
方案[13]	利用虚拟化技术检测虚拟域操作系统内的 Rootkit	签名比较等	vIPS	研究内容不同
VMwatcher[14]	利用虚拟化技术检测虚拟域中的恶意程序	语义视图重构等	VMware、Xen 等虚拟机, Windows、Linux 等操作系统	研究内容不同
Ether[15]	利用硬件虚拟化检测虚拟域中的恶意程序	监控指令和系统调用执行等	Xen, Windows	研究内容不同
本文	对虚拟域操作系统启动和运行中的关键文件进行完整性度量, 能够度量不触发系统调用的文件和系统配置、检查注册表项的完整性	对系统调用进行捕获和分析	KVM, Linux 和 Windows	-

本文的方案利用了 OB-IMA 的设计思想。OB-IMA 是一种利用域外方法对虚拟域进行完整性度量的方案, 该方案能够很好地支持 Xen 虚拟机和使用 Linux 操作系统的半虚拟域。系统调用是操作系统用户空间的进程请求内核空间服务的接口。通过捕获系统调用的参数, OB-IMA 能够获得虚拟域操作系统内加载和运行的内核模块、可执行程序等会影响虚拟域完整性的实体并对它们进行度量。针对不触发

系统调用的程序加载器等实体，该方案提出通过分析程序的执行过程对其进行度量。此外该方案还能够度量会影响虚拟域完整性的配置文件等实体。

3 可信的虚拟域

实现可信的虚拟域，就需要对虚拟域的完整性进行度量。这主要涉及两项内容，一是确定需度量的实体，二是选择合适的度量方法。本节将对这两个问题进行详细讨论。

3.1 需度量的实体

虚拟域中具有执行能力的实体通常包括虚拟 BIOS、虚拟硬件设备、引导程序、操作系统内核和应用程序等，如果它们被恶意篡改，那它们的加载和运行将会影响虚拟域的完整性。此外，不具有直接执行能力的配置文件也会对计算平台产生间接的影响，例如 Linux 操作系统中的 inittab 文件、fstab 文件就会决定系统启动的终端界面功能和挂载的磁盘分区等。这些直接或间接影响虚拟域自身完整性的实体主要分为以下几类:

(1) 虚拟域的硬件配置、虚拟 BIOS、硬件设备等。

(2) 操作系统内核及其相关文件。例如 Linux 的内核和 ramdisk 文件、Windows 的 Ntoskrnl 文件等;

(3) 内核模块文件。例如 Linux 的.ko 内核模块文件、Windows 的.sys 驱动程序文件等;

(4) 可执行程序及其动态链接库文件。例如 Linux 的 ELF 文件和.so 库文件、Windows 的.exe 文件和.dll 文件;

(5) 需要解析器或运行时环境辅助执行的脚本文件、字节码文件。例如 Python 文件、Java 文件等;

(6) 系统配置文件。例如 Linux 中/etc 下的配置文件、Windows 中的注册表项等。

上述文件中，第 1 类文件实际存储在宿主机中，例如虚拟域的硬件配置通常以文件形式保存，宿主机中的管理工具会根据该文件来配置虚拟域的 CPU、内存和网络等硬件环境，虚拟 BIOS 和硬件设备则是 KVM 通过 QEMU 程序及其相关文件进行的模拟和仿真，这些文件的完整性可以在宿主机中直接度量。第 2~6 类文件则在虚拟域中，需要采用适当的方法对它们进行度量，这也是本文研究的重点。为了讨论方便，我们将上述这 6 类文件称为“关键文件”，将执行程序、加载配置文件等对关键文件的操作称为“关键操作”。称虚拟域中除“关键文件”之外的文件为“普通文件”，用户创建的文本文件和数据文件都属于普通文件。

3.2 度量方法

可信计算技术要求关键文件应当在其实际加载运行时被度量，但是有时度量结果并不能真实反映实体的实际加载和运行情况。例如，某实体在 t_1 时刻被度量，在 t_2 时刻被加载运行，如果存在恶意程序在 t 时刻($t_1<t<t_2$)修改了该实体，那么该实体在 t_1 和 t_2 时刻的完整性将是不一样的。这会导致实际运行的是 t 时刻修改后的实体，而 PCR 和 SML 中记录下的是实体在 t 时刻之前的完整性状态，从而导致挑战者无法准确得到平台的完整性状态，这就是 TOC-TOU 问题[21]。在可信计算的框架下，TOC-TOU 问题难以完全避免。因此，应当在实体加载时进行实时度量，即尽可能地减少实体度量与加载之间的时间差；或者在实体加载前预先度量，再通过其他手段确保实体在度量和加载间的完整性不会发生变化。

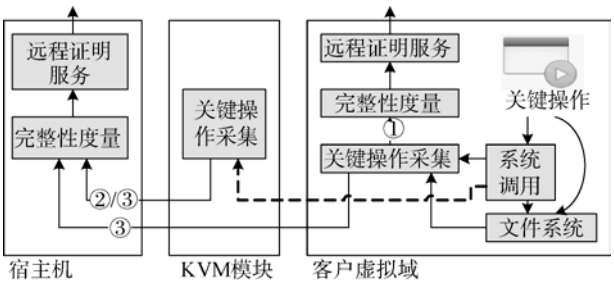

图 2 域内、域外和域内外协同方法原理图

实时度量需要能够实时截获关键操作，同时对涉及的关键文件进行度量。根据关键操作采集模块所在的位置，可以将实时度量的方法分为域内方法、域外方法和域内外协同的方法。图 2 是这三种方法的原理图，图中的①②③分别表示这三种方法。域内方法指关键文件的捕获和度量等操作全部都在虚拟域内进行，这类方法和是否是虚拟机环境没有关系，这里将不再讨论。

除了实时度量外，如果能够通过对系统和配置文件的分析而作出实体是否需要被加载的判断并得出实体的加载顺序，就有可能对这些实体进行预先度量。接下来将分别讨论实时度量中的域外方法、域内外协同的方法以及预先度量的方法。

3.2.1 域外方法

域外方法，即对关键操作的实时采集和处理模块都位于虚拟域外。实时捕获信息通常有两种方法，一是轮询，二是事件响应。

轮询方法能够获得某一时刻非常全面的信息，例如 CPU 的状态、寄存器的值、内存页面的数据等。但是轮询方法存在比较大的缺陷，即需要设置一定的采样间隔。如果间隔过长，将导致两次数据采样间

发生的事件被遗漏; 如果间隔过短, 又会对系统的性能造成较为严重的影响。此外, 轮询方法本身也需要占用一定的 CPU 资源, 影响系统的性能。

事件响应是另一类捕获动态信息时常用的方法, 这需要将需捕获的信息与特定事件相关联, 还需要设置该事件触发时的响应操作。当事件发生时, 事件响应操作将被触发, 并即时采集与事件有关的 CPU 的状态、寄存器的值、内存页面的数据等等。这类方法具有较高的运行效率, 能够很好地减少对系统性能的影响, 但是难点在于怎样确定事件、怎样观测事件以及怎样实现事件的响应操作。

表 3 行为事件和观察方法

虚拟域内行为	引发事件	虚拟域外的观察方法
用户空间操作	系统调用	软中断
文件 I/O	文件系统 I/O	存储介质 I/O 操作
进程切换	页表地址改变	CR3 寄存器改变
网络访问	网络协议栈操作	虚拟网络数据流

表 3 列出了操作系统内常见的行为、可引发的事件以及在虚拟域外的观察方法。需要特别说明的是, 虚拟域内的行为所引发的事件通常并不是单一的, 例如, 在用户空间内的文件 I/O 将会引发系统调用和文件系统 I/O, 在用户空间内运行程序将会引发系统调用、文件系统 I/O 和页表地址改变。因此某一行为对应的观察方法也不是唯一的。

观察虚拟网络数据流是指在虚拟机监视器或宿主机中通过截获网络数据包的方法来分析虚拟域的网络访问行为, 再根据数据包的 IP 地址、端口、协议等信息进行进一步分析。但是这种方法难以分析加密传输的数据, 同时仅能分析访问外部网络的行为。因此, 没有网络连接和仅进行本地进程间 Socket 通信的进程都无法通过这种方法进行监控。

观察存储介质 I/O 操作是指在截获虚拟域中存储设备的读写请求, 进而分析虚拟域中对文件的读、写或执行等操作。但是由于存储介质位于系统中较为底层的位置, 因此能够观察到的内容是数据块的物理存储位置和读写操作等信息, 缺少上层文件系统中的路径、文件名等重要信息。

CR3 寄存器是 CPU 的控制寄存器之一, 操作系统通常用它存储当前进程的页表地址, 观察 CR3 寄存器可以较为容易地获得与当前进程的内存数据。但是操作系统管理内存时可能会为新建进程分配一个已结束的进程曾使用过的页表地址, 因此无法仅通过 CR3 判断当前进程是否为新建进程。同时有些和进程相关的信息是由操作系统维护的, 并不在该进程的虚拟内存空间中。此外, 主流操作系统都支持多进程、多任务功能, 多个进程会通过时间片切换等方法共同使用一个 CPU, 这将导致进程调度时 CR3 寄存器数据频繁变化, 监控该寄存器将对系统性能造成较大的影响。

系统调用捕获(即软中断捕获)能够捕获用户空间请求的所有特权操作, 因此用户空间中的关键操作都能够被拦截, 而且可以根据需要阻止已拦截的操作继续执行。但是由于内核空间的程序在执行时已具有特权级, 不需要通过系统调用来执行特权操作, 因此这种方法无法捕获内核空间的关键操作。

3.2.2 域内外协同的方法

域内外协同的方法主要有两类, 第一类是通过在域内外捕获同类信息并进行对比以检测系统是否存在异常, 第二类是分别在域内外捕获不同的信息以实现互补。这两类方法都涉及在域内和域外采集信息、在域间传递信息、在域内或域外处理信息。

在采集信息方面, 域外实时采集已经在 3.2.1 节中进行了讨论, 而域内采集, 即在操作系统内进行采集, 则主要依靠操作系统自身的接口或安全机制来实现。例如通过 Win32 API 获取 Windows 系统内的进程、注册表配置等信息, 也可以通过自定义的过滤器驱动程序从内核中获取信息。

在传递信息方面, 虚拟域与 KVM 内核模块之间传递信息具有一定的挑战性, 常见的方式有以下四种:

(1) CPU 传输。可利用虚拟域 CPU 中没有使用的寄存器传输信息。该方法会受到寄存器位长的限制, 每次能传输 2~4 字节的信息, 不适宜传输较长的信息。

(2) 虚拟网络传输。可利用网络在虚拟域与特权域间传输信息。该方法具有较高的传输效率, 但是需要虚拟域必须配置虚拟网卡, 而且传输的数据存在被虚拟域内恶意程序篡改的可能。

(3) 虚拟设备传输。可在虚拟域中添加自定义的字符型设备或块设备(即前端设备), 同时在宿主机中为该设备提供软件形式的仿真实现(即后端设备), 然后通过设备 I/O 传输信息。该方法具有较高的效率, 但是对系统的改动较大。

(4) 内存传输。有些虚拟机在设计隔离机制的同时也提供了共享方法, 虚拟域可以授权宿主机或其他虚拟域读写自己指定的内存区域。即使虚拟域没有授权, 但是由于虚拟域的内存是宿主机内存的一部分, 同时受到 KVM 的管理和分配, 因此可以通过查找内存映射关系读取虚拟域特定的内存区域。通过内存传输同样具有较高的效率。

在完成域间信息传输后，在域内或域外工作的模块即可对信息进行处理并完成相应的工作。

3.2.3 预先度量的方法

预先度量，即根据实体的路径、文件名等定位信息在其加载和运行前进行度量。预先度量必须满足两个前提条件，一是能够确定该实体一定会被加载，二是能够保证实体在度量到真正加载间的完整性不会发生变化。虚拟域中的一些实体是满足上述条件的，例如，如果虚拟域中的虚拟 BIOS、硬件设备和操作系统引导程序都是经过认证的，那么在操作系统内核加载之前恶意程序将无法被任何实体加载和运行，因此操作系统的内核也就不会被篡改。

预先度量可以作为实时度量的辅助手段。如果一些实体的加载和运行不能通过实时度量的方法捕获，而同时又满足上述两个前提条件，就可以通过预先度量的方法对这些实体进行度量。

3.3 安全目标

本方案假设攻击者能够在虚拟域对外提供服务时利用虚拟域内核或程序中存在的漏洞对虚拟域中的文件进行读写，例如攻击者可以利用这些漏洞修改虚拟域内的配置文件和可执行程序等关键文件。但是攻击者不具有破坏宿主机的能力，即宿主机中的 KVM 模块、虚拟机管理工具等都是安全的。

本方案的安全目标是能够真实记录虚拟域中被加载和执行的关键文件的度量结果，并将结果正确地提供给挑战者。

4 域外监控: 对 Linux 虚拟域的完整性监控和度量

本节将提出域外监控方案，这是一种基于域外方法对使用 Linux 操作系统的虚拟域的完整性进行度量的方案。虽然 3.2.1 节中列出的域外方法都存在各自的缺陷，但是系统调用捕获的方法能够拦截的关键操作较多，因此本节提出的方案将以系统调用捕获的方法为主，结合对开源操作系统内核的分析找出所有的关键操作涉及的关键文件，通过对这些文件进行度量来评估虚拟域的完整性。域外监控方案参考了 OB-IMA 方案的原理，对该方案进行了修改和完善使其能够应用于 KVM 虚拟机。本节将首先介绍域外监控方案面临的问题及解决方案，然后提出详细的系统设计方案，最后通过实验验证方案的完备性和性能并分析方案的安全性。

4.1 面临的问题及解决方案

为了能够在虚拟域外准确地获得关键文件操作并真实地反映虚拟域完整性状态，域外监控方案需要解决以下三个问题:

(1) Linux 操作系统中的系统调用有很多，对所有系统调用都进行监控将严重影响系统的性能，怎样有针对性地选择需监控的系统调用是本方案需解决的第一个问题。

(2) 有的关键操作不触发系统调用，怎样找到这些操作及其涉及的实体是本方案需解决的第二个问题。

(3) 对关键文件和普通文件的某些操作会使用同一种系统调用，但是由于访问普通文件并不影响虚拟域的完整性，因此怎样区分这些文件是本方案需解决的第三个问题。

接下来将介绍如何解决这三个问题。

4.1.1 选择需要监控的系统调用

操作系统中的系统调用有很多，但是并非所有的系统调用都是关键操作，也并非所有关键操作都会加载或运行新的关键文件。涉及新的关键文件操作的系统调用主要包括: 加载新程序的 sys_execve、加载库文件的 sys_uselib、打开或加载新文件的 sys_open 和 sys_mmap2 以及加载新内核模块的 sys_init_module。其中后两项系统调用依赖于 sys_open，即它们只能对 sys_open 打开后的文件进行操作。因此需要监控的系统调用主要有 sys_execve、sys_open 和 sys_uselib。以下称这三项系统调用为关键系统调用。

在 Linux 的用户空间发起系统调用时需要通过寄存器传输系统调用的参数，其中 EAX 存储系统调用的编号，EBX、ECX 和 EDX 等则依次存储系统调用的各个参数。sys_execve、sys_open 和 sys_uselib 的第一个参数都是指向文件名的字符串指针，sys_execve 的另两个参数分别是命令行参数和环境变量，sys_open 的另两个参数分别是打开方式和文件权限。当捕获这些关键系统调用时，可以通过寄存器中存储的数据或地址获得关键操作的相关信息。

4.1.2 找出不触发系统调用的实体

Linux 操作系统中只有内核和内核模块在内核空间运行，虽然它们的关键操作不能直接在域外通过系统调用捕获的方法发现，但是 Linux 的内核是开源的，这为分析内核的行为提供了可能。通过对 Linux 内核的分析可以发现，内核中直接涉及关键文件加载和运行的操作非常少，但是也有例外。例如，Linux 能够支持 ELF 文件、脚本文件等多种格式的可执行程序，当内核处理用户空间通过 sys_execve 系统调用发起的执行请求时，需要找到合适的程序处

理器(Handler)解析、加载和运行它们[①]。程序处理器由内核完成加载和运行操作，不需要用户干预。

程序处理器可以是特定的动态链接库、脚本解析器或提供特定模拟运行环境的可执行程序。常见的程序处理器有:

(1) 对于ELF格式的可执行程序，程序处理器是/lib/ld-linux.so.x (i686 架构)或/lib64/ld-linux-x86-64.so.x (x86_64 架构)等。GNU 系统和其他绝大多数使用 Linux 内核的系统中的基础可执行程序都是 ELF 格式的。编译器将在编译生成 ELF 文件时将文件头部的 INTERP 段设置为上述程序处理器，它的作用是为 ELF 程序查找和加载所需的动态链接库。

(2) 对于 Bash、Python、PHP 等脚本文件或解释型文件，程序处理器是相应的脚本加载器或解释器。脚本文件或解释型文件通常是纯文本文件，没有严格意义上的头部数据结构。Linux 内核会根据它们的头部魔幻码(magic code)进行判断: 如果文件首行以魔幻码“#!”开头，则魔幻码后必须指明脚本解析器的路径，系统将加载该脚本解析器来解析执行它；否则系统将加载默认的脚本解析器来解析执行它，默认的脚本解析器可以通过查找用户账户设置来获得。

(3) 对于现已少见的 a.out、som 和 em86 等格式的文件，Linux 内核也会根据其头部魔幻码判断文件格式并加载相应运行环境。

如果尝试直接执行其他格式的文件，Linux 内核将返回错误信息。

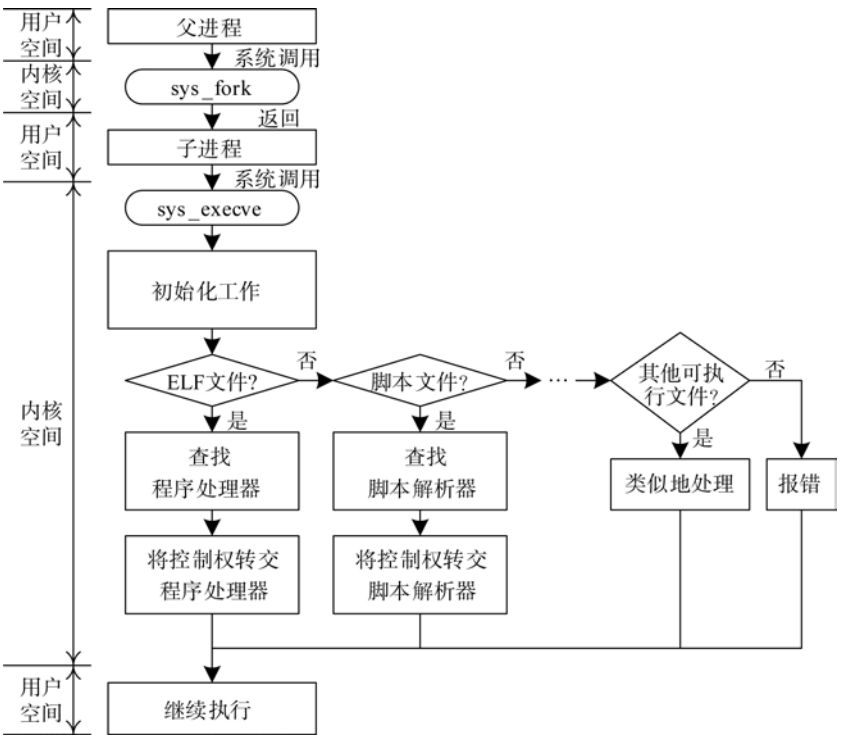

图 3　Linux 程序运行流程图

Linux 中加载和运行程序的流程如图 3 所示。当在用户空间中运行一个新的程序时，首先会由处理该操作的进程(父进程)发起 sys_fork 系统调用创建一个新的进程(子进程)，然后子进程再通过 sys_execve 来执行这个程序。内核处理 sys_execve 请求时会根据程序的头部信息加载相应的程序处理器，最后返回用户空间继续执行程序代码。虽然域外监控方案不能直接捕获到对程序处理器的加载操作，但是能够捕获 sys_execve 系统调用，这时可以按照内核中的处理流程找到相应的程序处理器。

除程序处理器外，Linux 内核中也存在访问其他关键文件的情况，但是涉及的文件几乎都是设备文件，对这些文件读写操作的本质是对设备的 I/O 操作。在云计算环境下，虚拟域的设备都是虚拟化技术得到的，它们在宿主机中的软件仿真和在虚拟域中的前端驱动都能够被度量。此外还有极个别不常见的设备的驱动程序会加载特定文件，如果虚拟域中没有这些设备则不会触发这些文件的加载操作，这

①以 2.6.32 版 Linux 源代码为例，该操作定义在 fs/exec.c 文件的 search_binary_handler 函数中。

里将暂时忽略它们。

4.1.3 过滤触发系统调用的普通文件

sys_open 是另外一个非常关键的系统调用, 在用户空间中发起插入内核模块、打开文件等请求时都需要调用它。由于打开配置文件、脚本文件等关键文件和打开普通文件都需要调用 sys_open, 因此怎样对它们进行区分是一个需要解决的问题。

Linux 操作系统中的文件存在一些固定的特征, 例如, ELF 文件、脚本文件和 Java 类库文件或字节码文件等关键文件可通过文件的魔幻码或头部数据结构等信息进行判断; 存储在/etc、/sbin、/bin、/lib、/usr/bin 和/usr/lib 等目录中的文件是系统的配置文件、可执行程序和动态链接库, 这些关键文件可通过文件的存储位置进行判断; 而存储在/home 等目录中的文件则很可能是普通文件。根据这些特征, 本方案提出通过策略对 sys_open 打开的文件进行区分。策略由白名单列表和运行时规则等组成。

白名单列表: 所有满足白名单规则的文件都不需要被度量。白名单由完整的路径和文件名指定。

运行时规则: 所有满足运行时规则的文件一旦通过 sys_open 系统调用打开就需要被度量, 除非它们满足白名单列表规则。运行时规则的设置非常灵活, 即可指定完整的路径和文件名, 也可仅指定路径, 还可通过文件的魔幻码或头部数据结构等特征来指定文件。

4.2 系统设计

域外监控方案主要包括三个组成部分, 分别是位于 KVM 内核模块中的系统调用捕获器、位于宿主机中的完整性管理器和位于虚拟域镜像文件存储服务器中的文件度量器。为便于讨论, 称宿主机和运行在其上的虚拟域为计算结点, 称虚拟域镜像文件存储服务器为存储结点。图 4 给出了本方案的架构图。以下将分别介绍它们的功能。

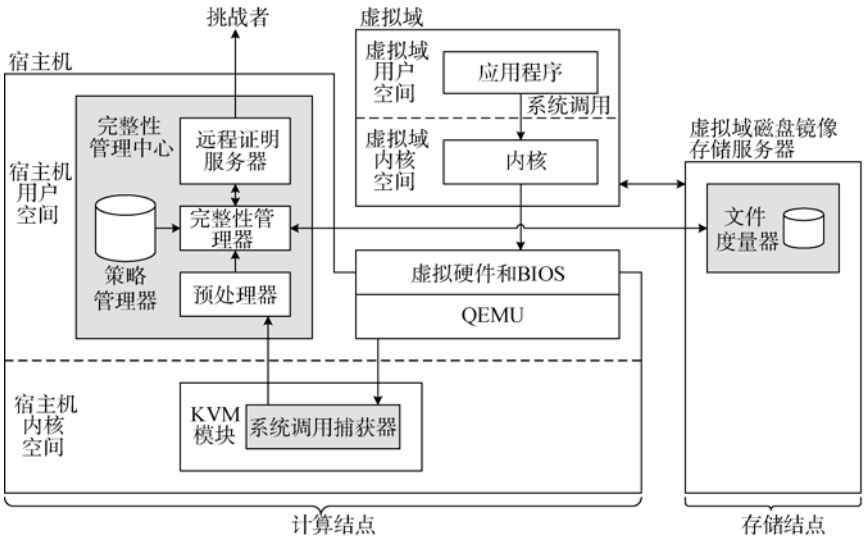

图 4 域外监控方案架构图

4.2.1 系统调用捕获器

系统调用捕获器负责捕获虚拟域发起的关键系统调用, 记录相关的信息并将这些信息传递给完整性管理中心。

在 KVM 内核模块中捕获系统调用的方法有多种, 本文在实现时参考了 Pfoh 等人提出的 Nitro 方法[22]。系统调用捕获器能够捕获虚拟域发起的所有系统调用, 然后过滤出 sys_execve、sys_loadlib、sys_open 等关键系统调用, 同时通过参数表得到系统调用涉及的关键文件的路径和文件名等信息。为了使 KVM 内核和虚拟域保持高效运行, 系统调用捕获器在获得上述信息后并不直接查找和度量相应的文件, 而是将这些工作交由宿主机中的完整性管理中心进行处理。为此, 系统调用捕获器需要将上述信息传递给完整性管理中心。传递的信息是一个三元组(*domID*, *filename*, *path*), 分别表示虚拟域的标识、关键文件的文件名和路径。

4.2.2 完整性管理中心

完整性管理中心位于宿主机操作系统中, 由预处理器、完整性管理器、策略管理器和远程证明服务器等模块组成。

预处理器负责接收系统调用捕获器传来的信息, 再根据 4.1.2 节的讨论找出程序处理器等不触发系统调用的关键文件, 最后将系统调用捕获器传来的关

键文件和通过分析得到的关键文件的信息一并传递给完整性管理器。

完整性管理器有两个主要功能，一是维护虚拟域的 PCR 和 SML，二是在度量结果出现异常时作出应急响应。当收到来自预处理器的关键文件信息时，完整性管理器会首先根据策略判断该文件是否需要度量。如果需要度量，则根据 *domID* 查找虚拟域存储空间(如镜像文件或逻辑卷等)所在的存储结点，然后向该存储结点中的文件度量器发起度量请求，并在文件度量器返回文件摘要值 *digest* 后将文件的 *path*、*filename*、*digest* 等信息记录到该虚拟域的 SML 中，同时将 *digest* 扩展到相应的 PCR 中。在应急响应方面，虚拟域的租户可以预先设置期望的实体度量结果和应急影响处理方案。如果文件度量器返回的 *digest* 和期望结果不同，表明虚拟域内的文件已经被修改，继续加载和运行此文件将导致虚拟域的完整性被破坏。在这种情况下，完整性管理器可以和 KVM 虚拟机联动，按照租户的设置作出应急响应，如断开网络、暂停虚拟机运行或直接关闭虚拟机等等。

策略管理器负责维护 4.1.3 节中提到的文件度量策略，向完整性管理器作出文件是否需要被度量的判断。策略管理器能够同时支持系统预设策略和虚拟域租户的自定义策略，这也体现了本方案的灵活性。

远程证明服务器为虚拟域的租户(挑战者)提供远程证明服务，向他们提供虚拟域的 PCR 和 SML 信息，由挑战者检验虚拟域的完整性与自己的期望是否相同。

4.2.3 文件度量器

文件度量器位于存储结点中，它从存储空间中找出完整性管理器指定的文件，对它进行度量并返回度量结果。实际的文件度量操作在存储结点进行，这样将可以减少因文件在存储结点和计算节点之间传输而造成的内部网络流量。

文件度量器中内置有一个用于保存度量记录的数据库，这将可以提高文件度量的处理速度。当接收到完整性管理器发来的文件度量请求后，文件度量器将首先从数据库中查找所请求的文件之前是否被度量过，然后在打开磁盘镜像文件或逻辑卷并检查所请求文件的状态：(1)如果数据库中没有度量记录，则直接对该文件进行度量，将文件的 *path*、*filename*、*digest* 和度量时间 *time* 等信息存储到数据库中，并将度量结果返回给完整性管理器；(2)如果数据库中已有记录，但文件在上次度量后已被修改，则对该文件重新度量，在数据库中存储新 *digest*，并将 *digest* 返回给完整性管理器；(3)如果数据库中已有记录且该文件在上次度量后未被修改，则直接返回“确认”信息给完整性管理器，这表明该文件的加载和执行不会再次影响虚拟域的完整性，完整性管理器将不改变 SML 和 PCR；(4)如果指定的文件不存在，文件度量器返回“文件不存在”的错误信息给完整性管理器，完整性管理器将根据系统调用的返回码判断系统调用 sys_open 是否正常打开了文件，并将异常情况记录到日志文件中。

4.3 实验和结果分析

接下来将通过实验对域外监控方案的完备性和性能进行分析，实验环境如表 4 所示。实验中使用本地磁盘替代专用的存储结点，因此文件度量器也运行在计算结点中。

表 4 域外监控实验环境

	宿主机	虚拟域
CPU	Intel Core 2 Q8300 2.5GHz	虚拟单核
内存	2.5G	1G
磁盘	500G SATA2 3.0Gb/s 桌面级硬盘	5.0G
操作系统	CentOS 6.3 i686	CentOS 5.9 i686
内核版本	Linux 2.6.32	Linux 2.6.18.8
虚拟机	KVM 0.13.0	-

4.3.1 完备性测试

该实验将对比同一虚拟域下域外监控方案和 IMA 方案能够度量的文件，以此来验证域外监控方案的完备性。实验中虚拟域操作系统的内核略有区别，在域外监控方案下将使用未开启 IMA 的 Linux 内核，而在 IMA 方案下则在该版本内核中以 Test 模式开启 IMA 功能。表 5 给出了两种环境下客户虚拟机启动过程的统计信息。

表 5 域外监控方案和 IMA 方案度量能力对比

方法	度量的关键文件数量	相同的文件		不同的文件
		相同文件名	软连接	
IMA	258	186	72	0
域外监控	1823			1565

从表 5 中可以看出，IMA 方案度量的 258 个文件都能够被域外监控方案度量，其中 186 个文件的访问路径完全相同，另外 72 个虽然文件名不同但是指向相同的软连接。此外，域外监控方案还比 IMA 多度量了 1565 个文件，这些文件可被分为以下 4 类:

(1) 系统配置文件，例如: /etc/inittab;

(2) 共享动态链接库文件，例如: /lib/libc.so.6;

(3) 脚本模块文件，例如：/usr/lib/python2.4/os.

pyo;

(4) 不可直接执行而需手动运行脚本解析器加载执行的脚本文件, 例如: /home/hvmuser/nonexecutable_helloworld.py。

根据之前的分析, 这些文件都是会影响虚拟域完整性的文件, 对这些文件的度量将使对系统平台状态的完整性判定更加准确。

4.3.2 性能测试

下面将通过 Benchmark 测试和虚拟域操作系统启动时间测试两种方法来检测域外监控方案的性能。

首先使用 libMicro 0.3[23]对域外监控方案和 IMA 分别进行 Benchmark 测试。libMicro 是用于 Linux 的一系列 Benchmark 测试的合集, 可以用于测试多种系统调用接口和链接库的性能。libMicro 0.3 共含有 259 个 Benchmark 测试项, 其中在本实验环境中有效的测试项为 229 个。这些测试项包括内存、字符串、进程和 I/O 操作等内容。和 IMA 方案相应测试项的执行时间相比, 域外监控方案: (1)有 69 个测试项的性能影响在 10%以下, 其中 25 项的性能影响在 2%以下; (2)有 79 个测试项的性能影响超过了 50%, 但其中 59 项的实际执行时间少于 2 微秒, 即对实际运行的影响很小; (3)总体性能影响为 14.82%, 虽然这一结果略高于 Xen 半虚拟域环境下 OB-IMA 方案 10.82%的性能影响, 但仍属于可以接受的范围。这一差异与 Xen、KVM 虚拟化实现不同有关, 因为 Xen 半虚拟化比 KVM 全虚拟化的性能要高。限于篇幅, 表 6 只给出了涉及 sys_execve 和 sys_open 的测试项以及其他常见的测试项。

从表 6 的测试结果可以看出, 域外监控方案对 open_usr 和 exec 操作的性能影响相对较高, 这是由于这两项操作涉及的系统调用被捕获后, 需要进一步地分析和处理。

表 6 域外监控方案和 IMA 方案 Benchmark 测试对比

测试项	执行时间(微秒)		性能影响
	IMA	域外监控	
open_usr	1.30298	2.08814	60.259%
exec	1677.3451	1884.7703	12.366%
read_t100k	20.443	22.169	8.443%
write_t100k	26.558	30.624	15.310%
memcpy_1k	0.12378	0.12785	3.288%
fork_100	724.79568	806.72781	11.304%
exit_100	590.5455	592.0773	0.259%

接下来测试对虚拟域启动过程造成的性能影响。本实验分别启动 IMA 和本方案的虚拟机各 50 次并分别计算平均启动时间。启动时间通过虚拟域操作系统启动完成后自动读取/proc/uptime 获得。图 5 列出了这 50 次实验的结果, 其中横坐标表示实验序数, 纵坐标表示虚拟域启动的时间。测试结果显示, IMA 方案和域外监控方案下虚拟域操作系统的平均启动时间分别为 46.062 秒和 51.569 秒, 性能影响约为 11.956%, 属于生产环境中用户可以接受的范围。需要说明的是, 域外监控方案在虚拟域操作系统启动过程中度量的文件数量远超过 IMA 方案, 同时实验环境中使用了桌面级硬盘而非企业级硬盘和网络存储服务, 大量且琐碎的 I/O 操作会对硬盘的性能产生一定的影响。

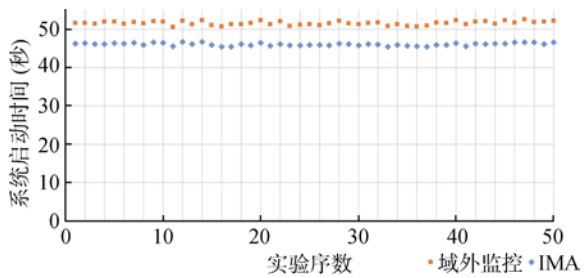

图 5 域外监控方案和 IMA 方案下虚拟域操作系统启动时间对比

4.3.3 安全性分析

在 3.3 节的安全假设中, 攻击者可能利用漏洞在虚拟域内执行程序或修改系统文件, 但不能破坏宿主机。本方案对虚拟域完全透明, 各组件均在宿主机的用户或内核空间中运行, 因此攻击者在虚拟域内加载和运行恶意或被篡改的程序的行为将能够被发现, 相关文件也将被度量, 但攻击者无法通过破坏该方案的各个组件而改变度量结果。

系统调用捕获是本方案的核心, 本方案为了减少对系统性能的影响, 仅监控了 3 项关键系统调用。如果攻击者获得了系统权限, 那么攻击者将能够通过修改系统调用表或中断向量而改变中断处理过程, 从而使关键系统调用不被捕获。攻击者可以通过加载内核模块等方法实现这一修改, 但是加载内核模块等操作会因触发系统调用而被捕获和度量。因此挑战者依然能够通过远程证明协议发现这一影响系统完整性的操作。

在 3.3 节的安全假设之外, 攻击者存在不通过加载内核模块、执行程序或修改文件等方法而实现攻击的可能性, 例如仅仅利用系统漏洞进行内存操作实现对系统的破坏。这种攻击不涉及修改或运行新的关键文件, 因而不能被本方案发现。这也是可信计

算技术所面临的挑战之一。内存监控、指令执行检测等方法会有助于解决该问题。

5 域内外协同监控: 对 Windows 虚拟域的完整性监控和度量

上一节提出的域外监控方案通过系统调用捕获、内核代码分析等方法获得系统内的关键操作, 再对这些操作涉及的关键文件进行度量, 使租户能够判断其使用的 Linux 虚拟域的完整性。但是这种方法不能适用于使用 Windows 操作系统的虚拟域, 这主要有三点原因。第一, 虽然在 Windows 用户空间中加载和执行可执行程序、读写文件等操作依然需要通过系统调用, 但是 Windows 中的系统调用数量远远超过 Linux 系统, 很多同类操作还可以通过不同的系统调用实现, 如果对所有相关的系统调用都进行捕获和分析将会对系统性能造成较大的影响。第二, Windows 内核是不完全开源的, 其内核空间中的关键操作既无法在域外通过系统调用的方法获得, 也不能通过分析源代码获得。第三, 与 Linux 将系统配置存放在固定目录中有所不同, Windows 中的系统和应用程序的配置都存储在注册表中, 注册表文件可以被视为一个小型的数据库, 但是注册表文件在系统每次运行时都会被修改, 直接度量注册表文件的完整性是没有意义的。针对这些问题, 本节将提出域内外协同监控方案对 Windows 虚拟域的完整性进行监控和度量。本节将首先介绍该方案面临的问题及解决方案, 然后提出详细的系统设计方案, 最后通过实验验证方案的完备性和性能并分析方案的安全性。

5.1 面临的问题及其解决方案

和域外监控方案相比, 域内外协同监控方案需要在域内和域外同时采集和处理信息, 这将面临以下问题: (1)在域内工作的模块如何采集所需的信息(包括不触发系统调用的关键操作)? (2)如何将域内模块获得的信息高效地传递到域外? (3)域内模块只有被虚拟域内的操作系统加载后才开始工作, 应该如何监控在此之前虚拟域的完整性? (4)Windows 的系统配置存储在注册表中, 怎样度量这些配置的完整性? 接下来将详细介绍这四个问题及其解决方案。

5.1.1 在域内采集关键文件操作信息

虽然捕获系统调用的方法在监控 Windows 虚拟域完整性时存在不足, 但是在虚拟域内能够通过其他方法进行弥补。例如, 不论是用户空间还是内核空间, 加载和执行程序或读写文件的操作都离不开存储介质和文件系统。操作系统中所有的文件访问请求都需要由内核传递给文件系统进行进一步处理, 文件系统负责找出文件在磁盘或逻辑卷中的存储位置, 最后经由磁盘驱动程序在存储介质中进行读写。和磁盘驱动相比, 文件系统处理文件访问请求时具有更加丰富的语义信息, 如文件的路径和文件名、文件对象属性、读写方式、共享权限等等。

Windows 的文件系统提供了过滤接口, 通过在文件系统过滤器驱动程序中实现相应的接口能够改变文件系统原有的行为。文件系统过滤器工作在内核空间, 能够在文件访问操作前后添加新的功能或修改原有行为。因此, 为了捕获系统中的关键文件操作, 可以在文件系统的文件访问操作前过滤出会影响虚拟域完整性的关键文件, 同时获得这些文件的文件类型、文件名、路径和访问权限等信息。由于存储系统配置信息的注册表文件的完整性需要单独考虑, 因此这里只需要过滤得到可执行程序、动态链接库等关键文件。过滤操作主要包括以下两项:

(1) 根据文件对象类型过滤, 去除目录、命名管道等非实体文件;

(2) 根据文件的访问权限过滤, 得到申请了执行权限的文件访问操作。

通过过滤得到的文件是具有执行权限的实体文件。这些文件的信息将被传输到宿主机中, 由宿主机中的完整性管理中心对相应的文件进行度量并记录到日志中。

5.1.2 域间信息传输

在 3.2.2 节中提到了在域间传输信息的方法, 但是这些方法都有各自的局限性。在域内外协同监控方案中, 域间传输的信息是关键文件的路径和文件名等信息, 这些信息具有长度有限、数量较大、实时性要求较高等特点。针对这种需求, 本文提出了基于系统调用的域间信息传输方法。

这种方法在虚拟域的 Windows 操作系统中放置了一个随系统自动启动的数据传输触发器。它工作在用户空间, 可以创建与文件系统过滤器的通信管道, 实时获得从文件系统过滤器中传来的信息。当它收到信息时, 它将发起一个系统调用, 并将这些信息作为系统调用的参数。在域外工作的模块将能够捕获到虚拟域发起的系统调用, 并通过参数获得所需的信息, 这就完成了信息从虚拟域到宿主机的传输过程。

这一方法基于事件机制, 因此具有较好的实时性和性能, 能够满足域内外协同监控方案的需求。不过, 该方法在实现时需要注意以下两个问题。

(1) 数据传输触发器发起的系统调用不能涉及

文件系统操作，或者在文件系统过滤器中通过过滤去除需涉及的文件。否则数据传输触发器发起系统调用时涉及的文件将被文件系统过滤器捕获，而该信息又将被发送给数据传输触发器从而导致循环依赖。

(2) 需要防止多个数据传输触发器实例同时运行。数据传输触发器通过建立通信管道与内核空间运行的文件系统过滤器进行通信，多个数据传输触发器的实例同时运行时将导致通信管道工作发生异常。针对这一问题，可以通过在数据传输触发器中添加系统全局信标等方法保证实例唯一性。

5.1.3 域内模块工作前虚拟域的完整性度量

在域内外协同监控的方案中，域内工作的文件系统过滤器、信息传输触发器在开始工作后才能将从文件系统中实时获得的关键文件的信息传送到域外，但是它们并非在操作系统一启动时就立即开始工作。因此，如何度量操作系统加载和启动过程中的完整性是需要解决的问题。

Windows 操作系统的常规启动过程大致可分为三个阶段。第一阶段是操作系统引导和驱动程序加载阶段。在该阶段中，操作系统引导程序将控制权转移给 Windows 加载程序(Winload.exe)，它将根据系统的配置加载内核、硬件抽象层和所需的驱动程序，完成内核中对象、电源、I/O 等多个管理器的初始化工作，然后调用会话管理器(SMSS)。第二阶段是操作系统用户空间初始化阶段。在该阶段中，SMSS 将加载通用的动态链接库、完成注册表设置中挂起的操作，然后再为每个会话创建一个 SMSS 子进程。会话 0 的 SMSS 子进程将加载客户端服务器运行时子系统(CSRSS)、Win32 子系统并调用 Wininit 完成对服务控制管理器(SCM)和本地安全权限子系统(LSASS)的初始化，每个用户会话的 SMSS 子进程将显示登录界面(LogonUI)。第三阶段是用户操作阶段，系统在用户登录后将进行用户环境的初始化(UserInit)、加载桌面程序(Explorer)并等待用户的操作。

根据上面的分析，在域内工作的文件系统过滤器将在第一阶段的驱动程序加载阶段被加载和运行，而信息传输触发器将在第二阶段由 SCM 加载和运行。这表明实时度量的方法无法捕获这两个阶段涉及的关键操作。然而，当 Windows 操作系统以常规方式启动时，这两个阶段是无用户交互的运行阶段，各个实体间的加载和调用关系是通过程序硬编码和注册表中的配置决定的，不接受用户的任何输入。因此，在不考虑外界的干扰的情况下，这两个阶段所加载的各个实体是可以分析和预测的。如果这些实体都不会改变其他实体的完整性，就能够满足 3.2.3 节中提到的预先度量方法所需的条件，也就可通过预先度量的方法验证这两个阶段 Windows 操作系统的完整性。在第二阶段结束前文件系统过滤器、信息传输触发器将开始协同工作，它们将能够对第三阶段的关键操作进行实时捕获，这就使完整性度量操作能够实现无缝衔接。图 6 是这三个阶段和所适用度量方法的示意图。

5.1.4 注册表的完整性

注册表由 BCD、System、SAM、Security、Software 等注册表文件和一些运行时生成的易失数据组成，它们是 Windows 操作系统中存储启动信息、系统、用户账户和应用程序等配置和信息的数据库，因此注册表对 Windows 操作系统的完整性非常重要。但是由于注册表中也存有很多并不影响操作系统完整性的信息，例如系统的关机时间、应用程序的临时数据和用户的个性化配置等，因此直接验证注册表文件的完整性是没有意义的。

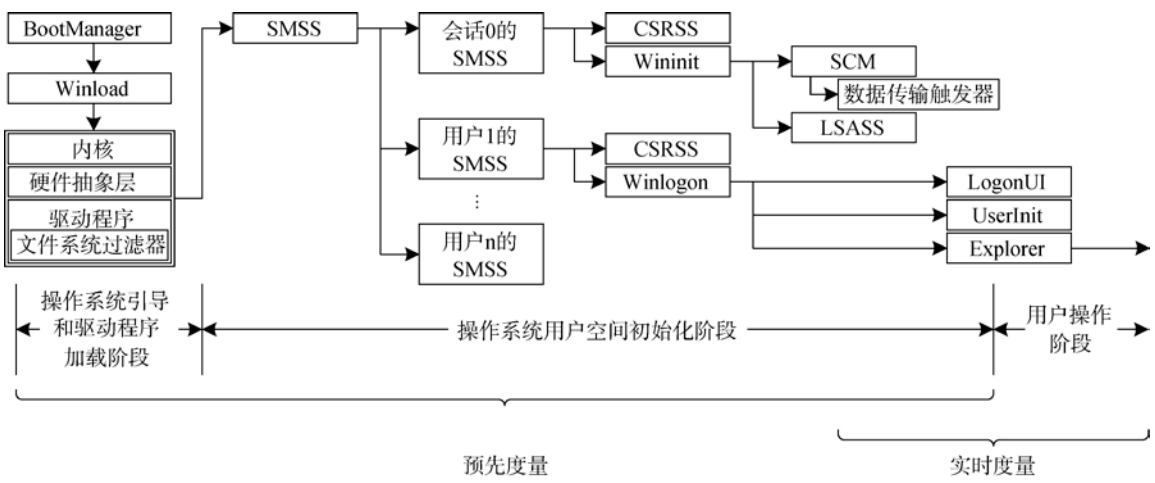

图 6 Windows 启动过程及其适用的度量方法

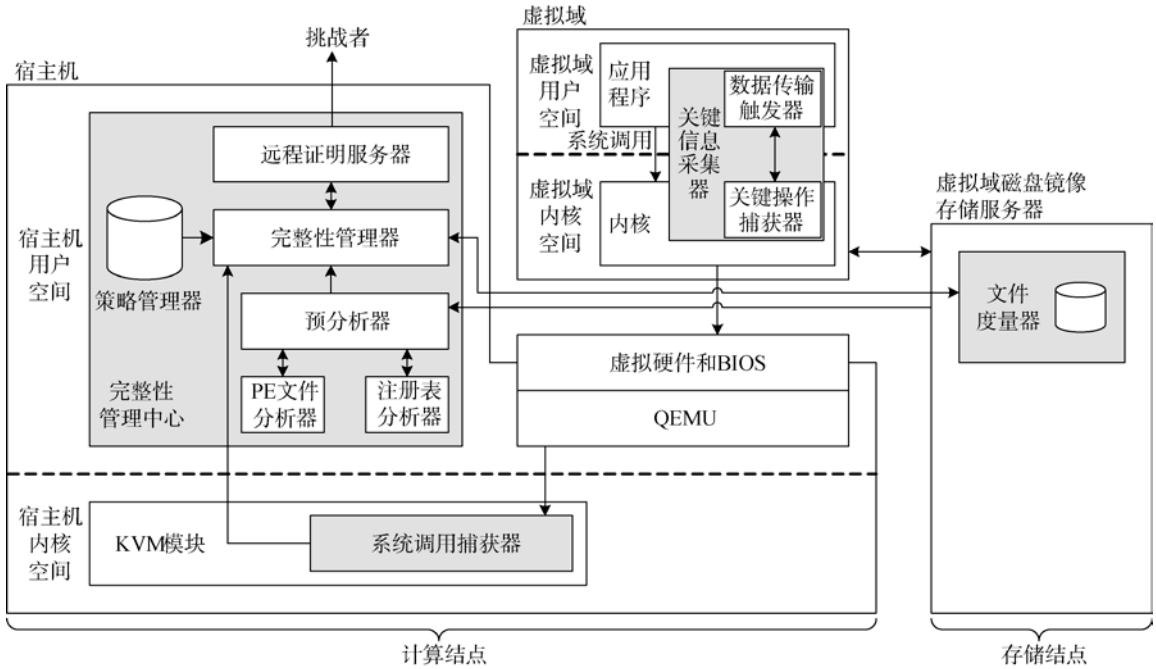

图 7　域内外协同监控方案架构图

表 7　注册表项的存储文件及其意义

注册表项	存储文件	意义
HKLM\BCD00000000	BCD	操作系统启动项
HKLM\SYSTEM\ControlSet001\Services	SYSTEM	驱动程序和服务配置
HKLM\SYSTEM\ControlSet001\Control\Session Manager	SYSTEM	会话管理器配置
HKLM\SYSTEM\ControlSet001\Control\ServiceGroupOrder	SYSTEM	驱动程序和服务加载顺序
HKLM\SYSTEM\ControlSet001\Control\GroupOrderList	SYSTEM	驱动程序和服务组内的加载顺序

针对这一问题, 可以采用更细粒度的注册表项完整性检验方法。由于虚拟域操作系统加载注册表时, 用户空间还不具有运行应用程序的能力, 因此需要在虚拟域外对注册表项进行完整性检测。这就需要在域外解析注册表文件、读取注册表项。注册表是一个树形结构, 存储着键(Key)-值(Value)形式的数据, 每个键下都可以存储若干个子键和值项, 根据由键和子键构成的路径和值名(Value Name)可以直接定位到需要读取的值。根据文献[24]中对注册表的介绍, 本方案将对表 7 列出的注册表项及其子键进行完整性检测。

虽然本方案仅对上述较为重要的注册表项进行完整性检测, 但是也可以根据需要对其他注册表项进行检测。

5.2　系统设计

如图 7 所示, 域内外协同监控方案主要包括四个组成部分, 除了域外监控方案中已有的位于 KVM 内核模块中的系统调用捕获器、位于宿主机中的完整性管理中心和位于虚拟域镜像文件存储服务器中的文件度量器之外, 还新增了位于虚拟域中的关键信息采集器。本节将对与域外监控方案中存在差异的部分进行详细的介绍。

5.2.1　关键信息采集器

关键信息采集器工作在虚拟域中, 包括运行在内核空间的关键操作捕获器和运行在用户空间的数据传输触发器两个模块。前者是一个文件系统过滤器驱动程序, 后者是一个以守护进程形式运行的系统服务。它们负责实时采集虚拟域内对关键文件的操作, 并将这些文件的信息通过基于系统调用的域间信息传输方法发送给系统调用捕获器。5.1.1 和 5.1.2 节已经详细介绍了它们的工作原理。此外, 数据传输触发器还会周期性地发起系统调用, 使完整性管理中心能够确认其处于正常工作状态。

5.2.2　系统调用捕获器

和域外监控方案中一样, 系统调用捕获器负责捕获虚拟域发起的关键系统调用, 但是有两点不同。第一, 它不需要捕获多种系统调用, 而只需捕获数据传输触发器使用的系统调用。第二, 它在捕获系统

调用时获取参数的方式也有所不同。因为 Linux 中系统调用的参数依次存放在 EBX、ECX、EDX 等寄存器中，而 Windows 中系统调用的参数则按照参数的类型和参数表的顺序依次压栈，需要从指向栈顶的 ESP 寄存器向下查找所需的参数。最后，系统调用捕获器将获得的信息传递给完整性管理中心中的完整性管理器。

5.2.3 完整性管理中心

完整性管理中心工作在宿主机中，由预分析器、完整性管理器、策略管理器、远程证明服务器等主要模块和注册表分析器、PE 文件分析器等辅助模块组成。

预分析器的工作是在注册表分析器和 PE 文件分析器的辅助下进行的。注册表分析器负责解析虚拟域注册表文件，它能够遍历指定键下的所有子键和值项，也可以从指定的键中根据值名获取值项的值。Windows 操作系统中的可执行程序、驱动程序和动态链接库程序等文件都是 PE 格式的文件，它们在运行时可能需要依赖其他的动态链接库等等。PE 文件分析器能够对 PE 文件进行静态分析，从它们的头部数据结构中找出所依赖的文件列表。

预分析器在 KVM 虚拟机收到虚拟域启动命令时开始工作，它将完成两项任务，一是利用注册表分析器对虚拟域中的关键注册表项的完整性进行检测，判断它们与预先设置的注册表项期望值是否相同；二是根据文献[25]中提出的 Windows 驱动程序加载原理和顺序，利用注册表分析器和 PE 文件分析器从注册表中找出虚拟域 Windows 启动过程需加载的内核及其相关文件、需加载的驱动程序项及其顺序和依赖关系、需加载的通用动态链接库、需运行的服务及其处理程序等信息，将它们涉及的关键文件组成列表发送给完整性管理器。等待这些工作全部完成后，KVM 将继续原有的虚拟域启动操作。

完整性管理器依然负责维护虚拟域的 PCR 和 SML 并在度量结果出现异常时作出应急响应。它将接收来自预分析器和系统调用捕获器发来的关键文件信息，根据策略管理器中的策略得到需要度量的关键文件，在存储结点中文件度量器的协助下完成对关键文件的完整性度量，将 *digest* 记录到 SML 中并扩展到 PCR 中。需要说明的是，由系统调用捕获器发来的关键文件不需要通过 PE 分析器查找其依赖文件，这是因为操作系统在执行这些关键文件时会自动加载它们的依赖文件，而这些文件的加载操作都离不开文件系统，因此能够直接被关键文件捕获器捕获。

策略管理器负责度量策略的管理工作，远程证明服务器负责为虚拟域的租户(挑战者)提供远程证明服务。它们的功能和 4.2.2 节域外度量方案中相应模块的功能一致，在此不再做更多介绍。

5.2.4 文件度量器

文件度量器的功能和 4.2.3 节域外度量方案中的文件度量器是一致的。需要特别说明的是，和 Linux 不同，Windows 文件系统中目录和文件名称的大小写是不敏感的，不影响文件的唯一性。因此在存储结点中根据指定的路径和文件名查找和度量文件时需要忽略大小写。

表 8 域内外协同监控实验环境

	宿主机	虚拟域
CPU	Intel Core 2 Q8300 2.5GHz	虚拟单核
内存	2.5G	0.5G
磁盘	500G SATA2 3.0Gb/s 桌面级硬盘	5.0G
操作系统	CentOS 6.3 i686	Windows 7 with SP1 Ultimate
内核版本	Linux 2.6.32	6.1.7600.17514
虚拟机	KVM 0.13.0	-

5.3 实验和结果分析

接下来将通过实验对域内外协同监控方案的完备性和性能进行分析，实验环境如表 8 所示。在本实验中依然使用本地磁盘替代专用的存储服务器。

5.3.1 完备性测试

由于目前没有用于 Windows 完整性度量的开源代码，因此不能直接比较域内外协同监控方案的完备性。这里将通过对比存储在 SML 列表中的文件信息与其他系统分析工具获得的信息的方法进行完备性测试。但是由于这些分析工具有着不同的适用环境，因此我们按照 5.1.3 节中提到的 Windows 启动过程的三个阶段分别进行测试。

(1) 操作系统引导和驱动程序加载阶段

通过开启 Windows 操作系统引导程序 BootManager 的调试参数能够记录这一阶段加载关键文件列表。表 9 对比了 BootManager 和域内外协同监控方案的预分析器分别得到的关键文件加载列表。

表 9 域内外协同监控实验环境

方法	关键文件数目	相同的文件数目	不同的文件数目
BootManager 日志	111	110	1
域内外协同监控的预分析器	129		19

从表 9 中可以看出, 本方案在此阶段得到的关键文件列表和 BootManager 记录的日志存在一定差异。BootManager 能够记录而预分析器没有得到的文件是 mcupdate_genuineintel.dll, 该文件和计算平台使用的处理器有关。通过预分析器得到的而不在 BootManager 日志列表中的文件有 19 个, 包括 videoprt.sys 和 watchdog.sys 等, 这些文件都是其他关键文件所依赖的文件, 应当由系统自动加载。在两种方法都能够得出的 110 个相同的文件中, 有 12 个文件的加载顺序存在差异。经检查, 虽然注册表中确定了驱动程序和服务的分组加载顺序和组内加载顺序, 但是也有一些驱动程序和服务的加载顺序并没有被明确标注, 这表示它们加载次序对系统不会造成影响, 而这 12 个文件均属于这类驱动程序。

(2) 操作系统用户空间初始化阶段

本实验将对比通过专用文件系统过滤器和域内外协同监控方案中的预分析器得到的关键文件加载列表。这里的专用文件系统过滤器是根据本方案中关键操作捕获器而制作的一个特殊版本, 它将在上一阶段被加载后立即开始工作, 把这一阶段操作系统执行或以执行权限加载的所有关键文件记录到日志文件。

经过对比, 本方案通过预分析器得到的文件共有 198 个, 比专用文件系统过滤器少 1 个, 该文件是操作系统的虚拟内存文件 pagefile.sys, 但是该文件应当对冷启动后的操作系统没有影响。两种方案得到的其余文件完全相同, 这表明预分析器能够正确分析和度量此阶段涉及的关键文件。

(3) 用户操作阶段

这一阶段域内外协同监控的实时度量将可以使用。为了检测用户的操作涉及的关键文件是否能够被关键操作捕获器捕获, 这里将对比分别使用 Process Monitor[26]和域内外协同监控方案采集运行同一应用程序时发生的关键操作。以运行记事本程序(Notepad.exe)为例, 表 10 给出了对比结果。

表 10 域内外协同监控方案和 Process Monitor 对比

方法	捕获关键文件数量	相同的文件数	差异	
			资源不存在	不是实体文件
Process Monitor	47	39	5	3
域内外协同监控关键操作捕获器	44		5	0

从对比中可以看出, Process Monitor 共监测到申请包括执行权限在内的关键文件访问请求涉及文件共计 47 个, 其中 39 个文件访问成功, 8 个文件访问失败。其中访问成功的 39 个文件均可被域内外协同监控方案捕获和度量。在差异部分, 有 3 个 Process Monitor 监测到的目录访问请求没有被域内外协同监控方案捕获, 这是因为本方案中的关键操作捕获器对命名管道和目录等非实体文件进行了过滤。

根据上述实验的结果可以看出域内外协同监控方案的预先度量和实时度量能够基本满足对虚拟域完整性的度量要求, 但是也遗漏了部分关键文件, 例如之前提到的 mcupdate_genuineintel.dll。如果操作系统启动过程中添加了即插即用设备, 那么这些操作涉及的关键文件也不能被本方案捕获和度量。对于前者这种属于与平台有关的文件, 可以通过在预分析器中添加相应的规则进行弥补。对于后者, 域内外协同监控方案则不能满足要求, 因为这种方案要求虚拟域 Windows 操作系统必须以常规方式启动过程, 而且启动阶段不能接受用户的输入和外界的干扰。在云计算服务中, 虚拟域的硬件配置是预先设置的, 在启动过程中不存在新的即插即用设备, 因此域内外协同监控方案在此条件下依然是有效的。

需要说明的是, 本实验使用的 Windows 是 Microsoft 为开发者提供的版本, 仅安装有与实验有关的调试和 Benchmark 软件, 没有安装其他驱动程序和应用程序。方案设计中对 Windows 内核和操作系统启动过程的分析都是通过参考已公开的文献和资料进行的。实验结果不排除 Windows 存在尚未公开的关键操作。

5.3.2 性能分析

下面将依然通过 Benchmark 测试和虚拟域操作系统启动时间测试两种方法进行性能分析。

首先使用 Windows 平台中的主流 Benchmark 测试工具 PCMark 进行对客户虚拟机的性能进行测试。PCMark 是 Futuremark 发布的用于 Windows 平台的 Benchmark 测试工具, 这里使用支持 Windows 7 操作系统的 PCMark 7(1.0.4 版)对客户虚拟域的性能进行测试, 表 11 给出了测试结果, 其中前 7 项给出的处理速度, 最后 1 项是评分。

表 11 域内外协同监控 Benchmark 测试结果

测试项	原始系统	域内外协同监控
Windows Defender	1.07MB/s	1.01MB/s
Importing pictures	1.23MB/s	1.21MB/s
Video editing	9.46MB/s	9.41MB/s
Windows Media Center	2.39MB/s	2.36MB/s
Adding music	0.28MB/s	0.28MB/s
Starting application	2.85MB/s	1.39MB/s
Gaming	3.83MB/s	3.69MB/s
System storage score	830	723

从测试结果可以看出，本方案对应用程序的启动影响较大，性能影响大约在 51.23%。这主要是因为每当运行应用程序时，都会涉及应用程序和动态链接库等多个关键文件加载和运行。而导入图片、编辑视频和添加音乐等测试项的性能影响则在 2%以内，这是因为这些操作更多涉及的是普通文件，而这些文件是不需要被捕获和度量的。Windows Defender 和 Gaming 等类别的性能影响稍大于数据文件，这是因为该类软件在运行时可能涉及的关键文件较多。

接下来对虚拟域操作系统的启动时间进行比较。虽然 Windows 操作系统提供了计算系统启动时间的接口，但是在本项测试中并没有直接使用该接口。这是因为本方案包括了在虚拟域启动前进行的预先度量操作，而该接口的计时器初始时刻是 Windows 操作系统内核中完成时钟初始化的时刻。为了使实验结果更加准确和客观，实验中将分别记录宿主机中发出虚拟域启动命令的时刻和 Windows 操作系统完成启动的时刻，再通过时间差来计算虚拟域操作系统启动时间。该实验共测试 50 次，图 8 给出了域内外协同监控方案下的操作系统和原始操作系统启动时间对比结果。

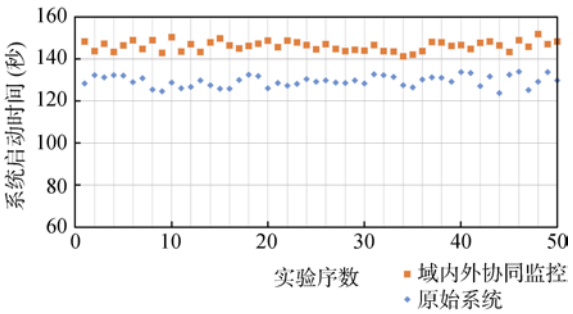

图 8　域内外协同监控方案下的操作系统和原始操作系统启动时间对比

从图 8 中比较的结果可以看出，域内外协同监控方案下操作系统启动时间平均为 146.16 秒，而原始操作系统启动时间平均为 129.40 秒，性能影响约 12.95%，属于可接受的范围。

5.3.3　安全性分析

本节将在 3.3 节和 4.3.3 节的基础上对域内外协同监控方案的安全性进行讨论。根据之前的分析，方案中位于虚拟域外的各个模块都是安全的。但是和域外监控方案相比，域内外协同监控方案在虚拟域内增加了关键信息采集器模块，因此保护该模块的安全是非常重要的。

由于虚拟域操作系统的引导和驱动程序加载、用户空间初始化等阶段是没有用户交互的，因此攻击者只能在用户操作阶段对外提供服务时进行攻击。此时关键信息采集器模块中的关键操作捕获器作为驱动程序、数据传输触发器作为守护进程均已开始运行。攻击者可能(1)篡改它们并等待下次系统启动时生效，但这会在下一次启动前的完整性检查时被发现; (2)停用、篡改再启动它们，但再次启动的操作会因触发系统调用而被发现，且篡改后的程序能够被度量; (3)停用再启动它们，并在此期间进行恶意攻击，但攻击者执行恶意程序的操作会因触发系统调用而被发现，恶意程序也能够被度量; (4)仅停用关键操作捕获器，则其与数据传输触发器间的通信管道将中断，数据传输触发器可通过系统调用报告异常; (5)仅停用数据传输触发器，这会使完整性管理中心无法收到周期性的系统调用而发现异常。此外，反病毒软件所使用的防御技术也可用于保护域内模块的安全。

6　小结

本文提出了域外监控和域内外协同监控两种完整性度量方案。利用这些方案, CSP 将能够为租户提供带外的虚拟域完整性监控、度量和验证能力，使租户能够准确判断出所租用虚拟域中操作系统内核、驱动程序、系统配置、可执行程序和脚本文件等是否与预期相同。本文对不触发系统调用的关键文件和不完全开源的 Windows 操作系统进行了深入的研究，因此提出的域外监控方案能够对 Linux 虚拟域进行完全透明的完整性度量而不依赖操作系统的安全机制，同时该方案因无法被虚拟域内的恶意程序旁路而更加安全；提出的域内外协同监控方案通过预先度量和实时度量也能够对 Windows 虚拟域的完整性进行较为全面的分析。实验结果显示两种方案的度量能力是完备的，对系统性能的影响也是可接受的。

本文提出的方案不仅可以用于云计算 IaaS 服务中，还可以用于在私有云或其他涉及敏感操作的信息系统中发现和拦截高危操作。下一步还将继续研究多用户环境下关键操作和用户之间的对应关系，实现基于虚拟机监控的用户行为审计机制等，同时还将通过内存监控等技术进一步保护虚拟域在运行时的完整性。

致　谢　感谢审稿专家耐心细致地审阅并给出宝贵的意见和建议，感谢期刊编辑部的出版工作。本文中设计的方案和实验还得到了北京交通大学计算机与信息技术学院张大伟、王伟等老师的指导和陈勋、王中华等同学的帮助，在此一并表示感谢！

参考文献

[1] J.L. Griffin, T. Jaeger, R. Perez, R. Sailer, L.V. Doorn, and R. Cáceres, “Trusted virtual domains: Toward secure distributed services,” in *Proc. 1st Conference on Hot Topics in System Dependability (HotDep’05)*, pp. 4-4, 2005.

[2] S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, and E. Valdez, “TVDc: Managing security in the trusted virtual datacenter,” *ACM SIGOPS Operating Systems Review*, vol. 42, no. 1, pp. 40-47, 2008.

[3] S. Berger, R. Cáceres, K.A. Goldman, R. Perez, R. Sailer, and L.V. Doorn, “vTPM: Virtualizing the Trusted Platform Module,” in *Proc. of the 15th USENIX Security Symposium (SSYM’06)*, pp. 21-21, 2006.

[4] “Trusted Computing Group,” https://www.trustedcomputinggroup.org/.

[5] R. Sailer, X. Zhang, T. Jaeger, and L.V. Doorn, “Design and Implementation of a TCG-based Integrity Measurement Architecture,” in *Proc. of the 13th USENIX Security Symposium (SSYM’04)*, pp. 16-16, 2004.

[6] T. Jaeger, R. Sailer, and U. Shankar, “PRIMA: Policy-reduced integrity measurement architecture,” in *Proc. of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT’06)*, pp. 19-28, 2006.

[7] G. Cheng, H. Jin, D. Zou, and X. Zhang, “Building Dynamic and Transparent Integrity Measurement and Protection for Virtualized Platform in Cloud Computing,” *Concurrency and Computation: Practice and Experience*, vol. 22, no. 13, pp. 1893-1910, 2010.

[8] A.M. Azab, P. Ning, E.C. Sezer, and X. Zhang. “HIMA: A Hypervisor-Based Integrity Measurement Agent,” in *Proc. of the 25th Computer Security Applications Conference (ACSAC’09)*, pp. 461-470, 2009.

[9] A. Seshadri, M. Luk, N. Qu, and A. Perrig. “SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes,” in *Proc. of 21st the ACM SIGOPS Symposium on Operating Systems Principles (SOSP’07)*, pp. 335-350, 2011.

[10] B. Zou, and H. Zhang. “Integrity Protection and Attestation of Security Critical Executions on Virtualized Platform in Cloud Computing Environment,” in *Proc. of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, (GreenCom-iThings- CPSCom’13)*, pp. 2071-2075, 2013.

[11] N.A. Quynh, and Y. Takefuji. “A novel approach for a file-system integrity monitor tool of Xen virtual machine,” in *Proc. of the 2nd ACM Sumposium on Information, Computer and Communications Security (ASIACCS’07)*, pp. 194-202, 2007.

[12] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu. “Process Out-Grafting: An Efficient "Out-of-VM" Approach for Fine-Grained Process Execution Monitoring,” in *Proc. of the 18th ACM Conference on Computer and Communications Security (CCS’11)*, pp. 363-374, 2011.

[13] T. Hwang, Y. Shin, K. Son, and H. Park. “Design of a Hypervisor-based Rootkit Detection Method for Virtualized Systems in Cloud Computing Environments,” in *Proc. of the 2013 AASRI Winter International Conference on Engineering and Technology*, pp. 27-32, 2013.

[14] X. Jiang, X. Wang, and D. Xu. “Stealthy Malware Detection Through VMM-Based "Out-of-the-Box" Semantic View Reconstruction,” in *Proc. of the 14th ACM Conference on Computer and Communications Security (CCS’07)*, pp. 128-138, 2007.

[15] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, “Ether: Malware analysis via hardware virtualization extensions,” in *Proc. of the 15th ACM Conference on Computer and Communications Security (CCS’08)*, pp. 51-62, 2008.

[16] B. Xing, Z. Han, X. Chang, and J. Liu, “OB-IMA: out-of-the-box integrity measurement approach for guest virtual machines,” *Concurrency and Computation: Practice and Experience*, vol. 27, no. 5, pp. 1092-1109, 2015.

[17] “Kernel-based Virtual Machine,” http://www.linux- kvm.org/.

[18] B.D. Payne, M.D.P. de Carbone, and W. Lee. “Secure and Flexible Monitoring of Virtual Machines,” in *Proc. of the 23rd Computer Security Applications Conference (ACSAC’07)*, pp. 385-397, 2007.

[19] X. Wang, R. Karri. “NumChecker: detecting kernel control-flow modifying rootkits by using hardware performance counters,” in Proc. of the 50th Annual Design Automation Conference (DAC’13), pp. 79-79, 2013.

[20] “Trusted Platform Module (TPM) Specifications,” Trusted Computing Group, https://www.trustedcomputinggroup.org/developers/trusted_platform_module/specifications, 2011.

[21] S. Bratus, N.D. Cunha, E. Sparks, and S.W. Smith, “TOCTOU, Traps, and Trusted Computing”. *Lecture Notes in Computer Science*, vol. 4968, pp. 14-32, 2008.

[22] J. Pfoh, C. Schneider, and C. Eckert. “Nitro: Hardware-based System Call Tracing for Virtual Machines,” *Lecture Notes in Computer Science*, vol. 7038, pp. 96-112, 2011.

[23] “libMicro,” https://java.net/projects/libmicro/pages/Home/.

[24] M. Russinovich, D.A. Solomon, and Alex lonescu, “Windows Internals, Sixth Edition, Part 1,” *Microsoft Press*, 2012.

[25] M. Russinovich, D.A. Solomon, and Alex lonescu, “Windows Internals, Sixth Edition, Part 2,” *Microsoft Press*, 2012.

[26] “Process Monitor,” https://technet.microsoft.com/en-us/sysinternals/processmonitor.aspx.

邢彬 于 2009 年在北京交通大学信息安全专业获得硕士学位。现在北京交通大学信息安全专业攻读博士学位。研究领域为信息安全。研究兴趣包括: 云计算、虚拟化、可信计算。
Email: xingbin@bjtu.edu.cn

韩臻 于 1992 年在中国工程物理研究院计算数学专业获得博士学位。现任北京交通大学教授。研究领域为信息安全、计算机应用。研究兴趣包括: 信息安全体系结构、可信计算、云计算安全。
Email: zhan@bjtu.edu.cn

常晓林 于 2005 年在香港科技大学大学计算机科学技术专业获得博士学位。现任北京交通大学副教授。研究领域为信息安全、计算机网络。研究兴趣包括: 云计算和云安全、网络性能优化。
Email: xlchang@bjtu.edu.cn

刘吉强 于 1999 年在北京师范大学基础数学专业获得博士学位。现任北京交通大学教授。研究领域为信息安全。研究兴趣包括: 可信计算、应用密码学、安全协议、隐私保护、模型论。
Email: jqliu@bjtu.edu.cn