IVirt:基于虚拟机自省的运行环境完整性度量机制

林 杰[1),3)] 刘川意[2),3)] 方滨兴[1),3)]

[1)](北京邮电大学计算机学院 北京 100876)

[2)](北京邮电大学软件学院 北京 100876)

[3)](北京邮电大学可信分布式计算与服务教育部重点实验室 北京 100876)

摘 要 完整性度量是检测程序篡改的重要方法,但是在虚拟化环境下传统的检测方法已体现出不足.例如,度量软件与被度量对象处于相同操作系统中易受攻击.该文从安全性和性能两方面出发,提出了一种基于虚拟机自省的完整性度量机制 IVirt(Integrity for Virtualization).该机制从虚拟机外部通过地址转换和内容定位得到所需的虚拟机内存数据,从而对虚拟机内部的程序进行完整性度量,以检验程序是否遭到篡改.该文以典型的虚拟机监视器 Xen 为例实现了 IVirt 原型系统.相比于同类工作,IVirt 一方面将度量软件与被度量对象分离,防止度量软件遭到攻击;另一方面采用地址转换来度量运行时状态,这区别于采用事件拦截机制的度量方法,以降低性能开销.实验结果表明,该方法能够检测出虚拟机运行时的软件篡改,而且在性能上不会引入过高的代价.

关键词 虚拟机自省;完整性度量;虚拟化;虚拟机监视器;运行时间;安全

中图法分类号 TP311 **DOI 号** 10.3724/SP.J.1016.2015.00191

IVirt: Runtime Environment Integrity Measurement Mechanism Based on Virtual Machine Introspection

LIN Jie[1),3)] LIU Chuan-Yi[2),3)] FANG Bin-Xing[1),3)]

[1)](*School of Computer Science, Beijing University of Posts and Telecommunications, Beijing* 100876)

[2)](*School of Software Engineering, Beijing University of Posts and Telecommunications, Beijing* 100876)

[3)](*Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education, Beijing* 100876)

Abstract Integrity Measurement is an important method to detect compromised application, but under the virtualization environment traditional detection approaches have reflected some shortages. For example, the measurement software and measured objects are in the same operating system, so the measurement software is easily attacked. From the perspectives of security and performance, this paper proposes an integrity measurement mechanism based on virtual machine introspection—IVirt (Integrity for Virtualization). This mechanism obtains the needed memory data of virtual machine through address translation and content locating from outside of that virtual machine, thereby measuring the integrity of applications that are in the virtual machine is performed, so as to verify whether the applications are tampered with. The IVirt prototype was implemented in this paper adopting typical virtual machine monitor Xen. Compared with other work of the same kind, IVirt isolates the measurement software from the measured objects, preventing measurement software being attacked. On the other hand, address translation is employed to measure the runtime state, which is different from the method of using events intercepting, in order to reduce the performance overhead. The experimental results show that

收稿日期:2013-09-28;最终修改稿收到日期:2014-08-27.本课题得到国家自然科学基金(61202081)和国家"八六三"高技术研究发展计划项目基金(2012AA012606)资助.**林 杰**,男,1987 年生,博士研究生,主要研究方向为计算机网络、信息安全.E-mail:jie_lin@bupt.edu.cn.**刘川意**(通信作者),男,1982 年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为云计算与云安全、数据安全与数据保护.E-mail:cy-liu04@mails.tsinghua.edu.cn.**方滨兴**,男,1960 年生,博士,教授,中国工程院院士,主要研究领域为网络安全、信息内容安全.

this method has the ability of detecting software modification, and it does not introduce high performance cost.

Keywords virtual machine introspection; integrity measurement; virtualization; virtual machine monitor; runtime; security

1 引 言

云计算提供的基础设施服务是以虚拟机为核心的,服务器上使用的虚拟机比例也在不断增大,虚拟机内的安全问题受到越来越多的关注.

为了确定虚拟机内的软件是否遭到篡改,需要有效监控虚拟机的运行时状态.虚拟机使用者不希望软件在未知的情况下发生变化,所以及时检测出软件是否被篡改,可以降低虚拟机遭到破坏的可能性.研究者提出了许多监控方法,对程序的行为进行监控.通常采用的监控手段是基于主机的完整性度量,该方法直接在被检测系统内部安装监控软件,以检验程序是否被篡改.典型的技术是完整性度量架构(Integrity Measurement Architecture,IMA)[1],操作系统从内核初始化开始,对后续启动的应用程序、内核模块进行度量,将这些程序的 hash 值写入度量列表.由于完整性度量架构不能防止度量列表被篡改,所以引入 TPM(Trusted Platform Module),通过远程验证来检测列表是否有被修改.而在虚拟化环境下,采用 vTPM[2] 可以将可信链扩展到虚拟机内部,使虚拟机可以检测到模拟的 TPM 设备,可以像使用物理 TPM 一样.因此基于 TPM 的完整性度量在虚拟机内部同样可用.但是这种将度量软件和被度量系统放在同一个运行环境的方法所存在的问题是,度量软件暴露了自身,攻击者很容易检测到度量软件的存在.这种度量方法遭到的攻击可能来自两个方面,一个是度量软件本身,另一个是存有度量值的度量列表.所以研究者提出了一些解决的思路,将度量软件和被度量系统分离,如 HIMA[3](Hypervisor-based Integrity Measurement Agent),通过在虚拟机监视器中添加挂钩(hook)来拦截系统调用,从而在程序运行之前能够进行完整性度量,但是这种方法使得虚拟机的性能受到了较大影响.

在虚拟化环境下,对虚拟机内部的软件进行完整性度量主要面临以下挑战:

(1)隔离性.攻击者进入系统后可能对度量软件造成威胁,所以需要将度量软件置于一个与被度量系统隔离的环境中,以降低被威胁的可能性.

(2)虚拟机中地址的转换.完整性度量需要计算进程的 hash 值,在虚拟化环境下,进程的虚拟地址不能从主机操作系统中直接获得,而需要从虚拟地址转换为物理地址,然后才能定位出进程的地址范围.这个地址转换要利用虚拟机监视器的内存管理机制.另外,虚拟机内操作系统的内核结构将被用来确定某些重要变量的偏移量.

(3)度量内容的定位.通常完整性度量是将被度量对象作为文件来进行度量,在被度量对象运行之前,先检查整个文件的 hash 值.而本文的方法则不同,由于所采用的方法是针对虚拟机内存,所以程序必须载入内存后才能实施度量.度量区域要根据操作系统将文件载入内存后所分配的空间来确定,即被度量对象的起始地址和终止地址.

(4)语义鸿沟问题.语义鸿沟问题自利用虚拟化提供服务以来就一直存在[4],也不可避免.从虚拟机外部获取的信息是底层的二进制数据,如何能够准确地识别出所需的底层数据而进行完整性计算是自省机制面临的一个重要挑战.

针对以上问题,本文提出一种基于虚拟机自省的完整性度量方法 IVirt(Integrity for Virtualization),并实现了该原型系统.该方法采用内存映射的方式,将虚拟机操作系统的内核关键部分和进程关键部分映射到特权用户可访问的存储空间,通过复制出相应的物理页,利用地址转换找到所需的内存数据,计算这部分的 hash 值并与原始的 hash 值进行比对,进而得出程序是否有变化的结论.本文的机制不仅将监控软件置于虚拟机之外,也避免了采用系统调用所带来的性能代价.

本文的主要贡献有:

(1)提出一种从虚拟机外部进行运行时完整性度量的方法.传统的完整性度量大多是针对磁盘上的文件,而对运行时的程序并没有考虑.本文提出一种运行时的完整性度量方法,并实现了原型系统.该机制通过度量程序运行时的完整性来判断程序是否遭到篡改,度量的对象包括进程、内核模块以及动态链接库.此外,本文的完整性度量机制将度量软件和

被度量对象分离，保证度量值的准确性. 度量软件与被度量系统处于同一个环境中，容易遭到攻击. 而从虚拟机外部进行度量，利用虚拟化技术的强隔离性可以降低度量软件受攻击的可能性.

(2) 采用组件独立的度量方式. 通常的度量是以一种链式的度量方式，所有程序事先固定，即由先启动的程序来度量后启动的程序，通过这种方式形成物理机启动过程的可信链. 而在虚拟化环境下，本文所提方法则相对独立，度量值并非由可信链上的程序进行度量测得，可以随时对每个组件单独进行度量. 另外，在操作系统启动完成后，通常的度量采用事件触发来度量运行的程序，这就需要在系统函数被调用时进行度量. 而本文的方法则避免了系统调用所带来的性能影响.

(3) 缩小语义鸿沟. 在进行完整性度量之前，本文需要确定被度量对象在内存中的位置. 本文通过研究 x86 体系结构和虚拟机的内存管理机制，实现从虚拟地址到机器地址的转换. 由于不同操作系统对应的内核结构不同，所以本文的方法可以通过适配，以准确获取不同内核结构的数据. 根据获取的内存内容来计算 hash 值，达到度量的目的，从而跨越语义鸿沟障碍.

本文第 2 节对完整性度量的研究以及在虚拟化环境下完整性度量的相关工作进行评述；第 3 节通过分析运行时攻击确定可以度量的完整性内容，并详细说明本文所提方法的设计架构；第 4 节说明 IVirt 的实现；第 5 节对该架构进行实验评估；最后一节进行总结.

2　相关工作

完整性是表明程序是否被篡改的一个重要性质，研究者对完整性的度量和检测方法提出了一些思路. Smith 等人[5]在构建安全协处理器时考虑了代码更新需要进行完整性检验的问题，采用信任棘齿来保证代码层的重写. 每个代码层在一个保护段中，通过硬件锁来允许或拒绝对这些段的写访问. Sailer 等人[1]设计了基于 TCG 的完整性度量架构(Integrity Measurement Architecture，IMA)，它是目前 Linux 内核采用的完整性度量方法，包括度量机制、完整性质疑机制和完整性验证机制，其中度量机制对内核所加载的模块和用户级进程进行度量，在他们被装入内存之前采用 SHA1 算法计算程序的 hash 值. 计算得到的结果保存在度量列表中，该架构并不能防止度量列表被篡改，而是引入可信平台模块(Trusted Platform Module，TPM)使得远程端能够验证度量列表，以防止欺骗攻击. 完整性度量架构在实际部署中也有一些应用研究[6-7]. Jaeger 等人[8]提出了基于信息流完整性的度量方法 PRIMA (Policy-Reduced Integrity Measurement Architecture)，将可信来源的输入信息定义为高完整性，将可能来自不可信来源的输入信息定义为低完整性，低完整性的信息只有通过了过滤接口才能被接受. Baliga 等人[9]提出自动产生内核数据结构完整性规范的方法，采用数据结构不变性的形式，在被监控主机上安装特定的网卡，通过定期抓取内存快照来推理内核数据结构的不变性. Szekeres 等人[10]建立了内存腐化攻击的一般模型，将当前的保护技术作为策略以应对各类攻击，其中涉及到完整性的策略包括代码完整性、代码指针完整性、数据完整性、控制流完整性、数据流完整性.

针对虚拟化运行环境的完整性度量，根据度量软件所处的位置可以将相关研究分成两类：一类是将度量软件直接放在虚拟机中，这种方法可以将原有的非虚拟化条件下的度量方法移植到虚拟机中. Berger 等人[2]对 TPM 设备进行虚拟化，实现了虚拟 TPM(vTPM)，每个虚拟机与唯一的 vTPM 实例关联，所有的 vTPM 实例由虚拟机外部的 vTPM 管理器负责创建. 虚拟机内部只要安装有完整性度量软件便可进行度量操作，利用 vTPM 保证度量列表的完整性. Stelte 等人[11]在每个虚拟机中以内核模块的形式安装感知代理，设计了感知完整性度量架构(Sensory Integrity Measurement Architecture，SIMA)，由感知代理监控虚拟机内的系统事件，在虚拟机监视器中也安装感知代理负责执行特殊任务，进行完整性监控，虚拟感知代理本身的完整性由 vTPM 来保证. 所有的感知代理将收集的信息存储在一个共享的内存区域中，由中心监控软件评估这些信息并控制相应的感知代理. Huh 等人[12]基于软件白名单设计了一个完整性评估框架，框架的核心是配置解析器(Configuration Resolver)，利用虚拟应用出版商和软件生产商提供的信息，使用虚拟机配置验证工具(VMCVT)生成验证报告，以检验虚拟机系统中已安装软件的完整性. 但是从安全性的角度来说，以上研究将度量软件和被度量程序置于同一区域，直接将度量软件暴露出来，易受到攻击.

第二类是将度量软件放置于虚拟机之外，不需要在虚拟机内部安装任何软件. Terra[13]提供“开盒

(open box)"虚拟机和"闭盒(closed box)"虚拟机,开盒虚拟机运行一般目的的操作系统和应用程序,闭盒虚拟机运行有特殊要求的应用程序,让程序运行在专有的封闭平台内,平台拥有者不能修改闭盒虚拟机,利用虚拟机监视器提供的隔离性保证闭盒虚拟机的完整性,但是其无法对虚拟机内部程序的运行时完整性进行验证. Garfinkel 等人[14]实现了一个基于 VMware Workstation 的入侵检测原型系统 Livewire,该系统分为操作系统接口库和策略引擎两部分,策略引擎中的策略模块可以对虚拟机内的用户态进程进行完整性检验,但是该系统没有考虑虚拟机中其他部分运行时的代码完整性,如内核模块. Azab 等人[3]提出基于虚拟机监视器的完整性度量架构 HIMA,使用主动监控对虚拟机内部事件进行拦截,包括系统调用、中断和异常,在虚拟机监视器内放置挂钩,捕获所有所需的信息. 但是这种监控方式所带来的代价是对虚拟机的性能影响较大,只要虚拟机有系统调用,虚拟机监视器就要进行相应的处理.

本文提出了一种利用虚拟机自省技术对虚拟机内部进行完整性度量的机制,虚拟机自省技术为访问虚拟机内存提供了一种研究思路. Garfinkel 等人[14]提出虚拟机自省(VMI)的技术,能够从外部检查虚拟机,以分析运行在虚拟机内部的软件. XenAccess[15]不需要对虚拟机监视器进行修改,而是直接利用虚拟机监视器提供的接口,这种方式需要虚拟机监视器的支持,其利用自省技术对虚拟机中正在运行的程序和所加载的内核模块进行监控,仅仅能够列出它们的名称. 而攻击者在潜入被攻击对象的操作系统后,通常会替换系统中的某些程序,这些被替换程序的名称和原来正常程序的名称是一样的. 所以其监控的功能比较单一,单纯从名称来监控系统并不容易发现系统是否遭攻击.

本文采用 Xen[16]作为虚拟机监视器. Xen 直接运行在物理硬件层之上,每个虚拟机称为一个域(domain). 在机器启动时建立的初始域是 Domain0,拥有对其它虚拟机的创建、管理和销毁的权限. Xen 直接负责对底层物理内存的管理,保证虚拟机之间的隔离性. Xen 对外提供了访问虚拟机的相关接口. Xencontrol 用来对虚拟机的底层信息进行访问. Xenstore 类似一个文件系统,存有所创建虚拟机的基本信息,如内存大小、地址宽度,利用这些信息能有效判断虚拟机的体系结构.

3 IVirt 设计

3.1 完整性分析

运行时的代码完整性攻击威胁主要可以分为两类:一类是将系统本身含有的二进制程序替换;另一类是在程序运行时对其进行替换. 针对现有的程序完整性攻击,本节对每类攻击进行分析,并说明程序在运行时不变的特征.

由于系统中存在经常要运行的程序,如 SSH,所以攻击者通过在这些程序中增加特殊的功能,如开启后门,将原有的程序替换为满足攻击者需要的程序.

现有的研究大多集中于对磁盘上的二进制程序进行完整性验证,即在程序启动之前先计算其 hash 值,符合安全规则后再让其运行. 而运行时的完整性也需要得到保障,因为有些攻击,如通过 ptrace,是针对运行时态的程序,那么启动前验证的程序在运行时就无法保证其完整性.

程序执行时操作系统将可执行的二进制数据加载到代码段中,同时分配给进程相应的数据段、堆、栈等空间. 正常情况下,由于程序的运行,数据段、堆、栈等空间的内容会发生改变,但是其代码段不会被篡改. 而以上两种攻击使程序原来的代码发生了改变,因此在程序加载后,通过检测其代码段可以发现程序是否遭到篡改.

3.2 威胁模型

假设攻击者能够进入虚拟机,而且进入虚拟机后拥有 root 权限,所以攻击者进入虚拟机后能够修改、安装、删除任意的文件,包括应用程序、内核模块、动态链接库等. 修改可以是针对系统中的配置文件,或者针对源代码,以形成满足攻击者特定需要的程序. 而且攻击者会安装自己的应用程序以及内核模块. 删除操作可能是将系统日志文件删除,以掩盖攻击者的行踪.

如果度量软件和被度量系统在同一个环境中,攻击者为了防止攻击被检测到,很可能会对度量软件进行攻击. 本文的机制不在虚拟机中安装任何软件,所以能够抵御这类攻击.

假设攻击只针对虚拟机进行,而针对主机的攻击可以采用现有的检测方式[1],以保证主机上的应用程序没有被篡改,所以本文不对基于主机的检测进行讨论.

虚拟机监视器本身可能存在安全漏洞,攻击者

可能针对该漏洞发起攻击，因此为了防止这类攻击，虚拟机监视器要能够及时进行安全更新. 本文的机制由于与虚拟机监视器保持松耦合，所以虚拟机监视器的安全更新可以很容易地安装到本文所提出的架构中.

3.3　体系结构

根据以上威胁，本文从安全性和性能方面考虑了以下设计原则：

(1) 度量软件处于虚拟机外部. 度量软件置于虚拟机内部可能会受到恶意用户的干扰，降低度量软件的安全性. 本文的方法通过从虚拟机外部获取虚拟机的内存进行度量，对于虚拟机内部来说是透明的，保证了度量软件的安全性.

(2) 度量软件应该对虚拟机造成的影响尽可能小. 不能让虚拟机里的用户感觉到性能有明显降低，采用添加挂钩的方法来拦截系统调用会影响虚拟机的性能，所以本文的方法采用间歇性度量机制对特定的内存进行访问，避免频繁内存访问引起的明显性能代价.

(3) 度量软件应该尽量和虚拟机监视器松耦合. 保持低的耦合性，这样才不会造成对虚拟机监视器的依赖性过高，以利于虚拟机监视器和度量软件各自的维护. 本文的方法尽量避免对虚拟机监视器进行修改，通过接口的方式获得虚拟机内部的信息，以保持好的程序易维护性，防止度量软件过度依赖于虚拟机监视器.

本文提出的完整性度量系统由硬件层、虚拟机监视器层、虚拟化层 3 大部分组成，其设计架构如图 1 所示. 硬件层包括 CPU、内存、网络设备、块设备等. 现代的 CPU 大部分支持虚拟化，如 Intel VT 和 AMD SVM，以满足虚拟化的性能和安全要求. 虚拟机监视器(Virtual Machine Monitor，VMM)层对 CPU、内存、网络、块设备完成虚拟化操作. 在虚拟化层中，由一个负责管理虚拟机的主机(Host)和多个虚拟机(Guest)组成. 由于本文在具体实现中虚拟机监视器采用 Xen[16]，所以主机中有 Xen 提供的用户接口 Xencontrol 和 Xenstore. 拥有特权的用户通过 Xencontrol 和 Xenstore 可以访问得到虚拟机的信息.

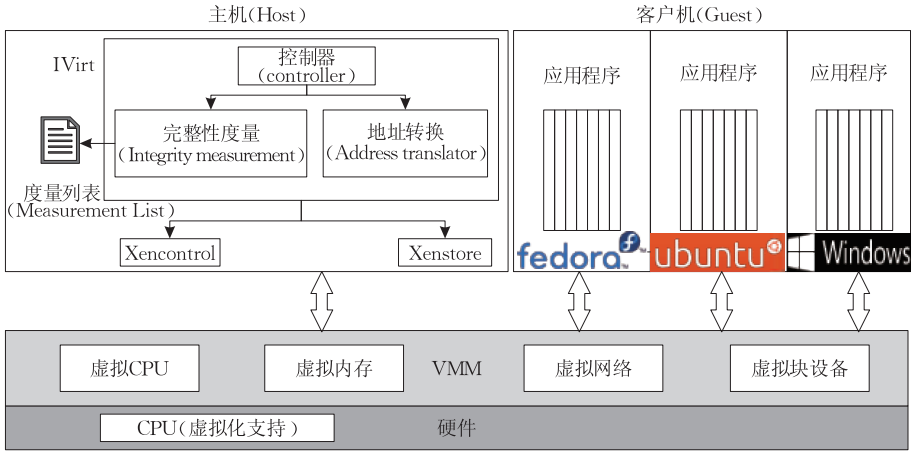


图 1　IVirt 完整性度量架构

度量软件 IVirt 位于虚拟化层的主机中，完成度量工作，包括控制端(Controller)、地址转换部分(Address translator)和完整性度量部分(Integrity measurement). 控制端负责控制完整性度量的整个过程，接收用户输入的一些参数，包括指定要度量的虚拟机名称、被度量对象、虚拟机的操作系统类型等，调用地址转换部分访问特定的虚拟机内存，将内存数据传递给完整性度量部分. 地址转换部分需要得到虚拟机中 CPU 的状态值、虚拟机地址的长度，同时还要能够访问虚拟机的地址映射接口，所以度量软件会与 Xencontrol 和 Xenstore 进行对接. 在取得信息之后根据具体的虚拟机 CPU 架构，进行地址转换工作，由此可以获得虚拟机的物理地址. 根据得到的虚拟机内存内容，由完整性度量部分来计算程序、模块以及动态库的 hash 值，并将这些值保存在一个度量列表(Measurement List)中. 度量软件根据需要可以定时度量也可以实时度量，新得到的度量值与原来保存在度量列表中的值进行比较，从而可以发现进程是否发生改变.

3.4　地址转换

在进行地址转换之前，需要对虚拟机监视器的内存管理进行说明. 本文采用 Xen 作为虚拟机监视器，其内存管理使用 3 种地址空间，分别是机器地址、物理地址和虚拟地址. 机器地址，或者也称为硬件地址，只有 Xen 监视器可以访问，物理地址由虚拟机操作系统进行管理，虚拟地址是应用程序所能

访问的地址.采用这种地址结构,不连续的机器地址就可以转换为看起来连续的物理地址.

虚拟机所能看到的是物理地址,并不能看到底层的机器地址.虚拟机监视器负责把虚拟地址转换为物理地址,同时维护由物理地址到机器地址转换的表.

完整性度量需要定位出程序在内存中的位置,确切地说是程序在虚拟机物理内存页的偏移位置,所以要先将虚拟机的物理页复制到特权用户可访问的空间,再进行定位,而这个过程需要完成从虚拟地址到物理地址的转换,IVirt 的地址转换部分实现机制如图 2 所示.IVirt 读取内核符号表,将内核符号对应的虚拟地址转换为物理地址.具体来说,在 64 位 x86 的体系结构中,如果 CPU 的页式存储功能是打开的,那么虚拟地址由 5 部分组成:PML4、页目录指针、页目录、页表、表内偏移.IVirt 获取虚拟机 CR3 寄存器的内容,该寄存器存放着页式层次结构 PML4 的物理基地址,通过逐级转换找到物理地址,完成虚拟地址到物理地址的转换,得到物理页框号(MFN).在主机管理域中,由虚拟机监视器的底层访问控制接口 libxc 负责对机器地址上的内容进行访问.根据物理页框号将机器地址上的内容映射到主机能够访问的内存空间,从而得到虚拟机内存的内容.在得到所需要的内容后,由度量计算部分负责对相应的内容进行完整性度量.

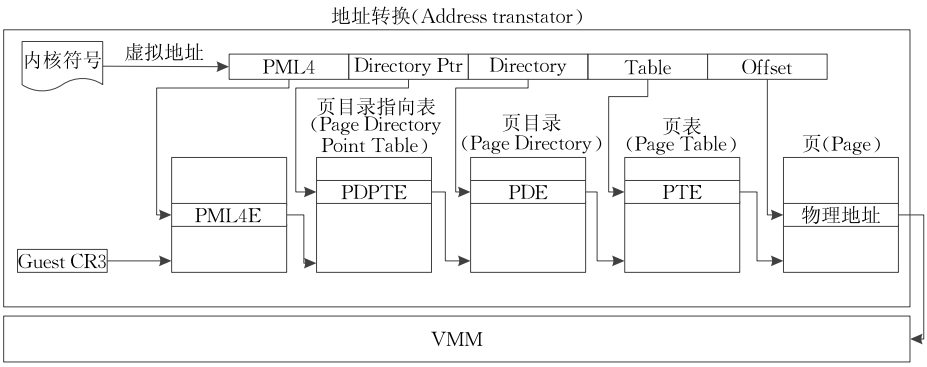


图 2 地址转换机制

3.5 完整性度量

度量得到的 hash 值被保存到度量列表中,同一个程序在不同的时刻出现不同的度量值,那么该程序就被检测出遭到修改.要对一个系统进行完整性度量,应该尽可能地度量多种程序.本文所提的机制具有广泛的适用性,即只要程序有被加载到内存当中,即可对其进行完整性度量.本文根据运行时加载到内存中的程序类型,主要考虑了能够从虚拟机外部检测到的被度量对象,包括进程、内核模块、动态库.其他不在本文中列出的可度量对象也可以采用同样的方法进行度量.

操作系统为方便管理进程和内核模块,会采用链式结构将系统中正在运行的进程和已加载的模块连接起来.因此通过这种结构可以获取到被度量对象所在的物理内存区域,再结合前一节所述的地址转换机制,便可以得到被度量对象的硬件内存的内容.下面分别对进程、模块和动态链接库具体说明如何根据其内存的存储区域进行完整性度量,本文采用 sha1 来计算 hash 值.

进程结构如图 3 所示.每个进程由进程控制块管理相关信息,所有进程通过双向链表连接起来.通过遍历该链表即可得到系统中运行的进程,进程控

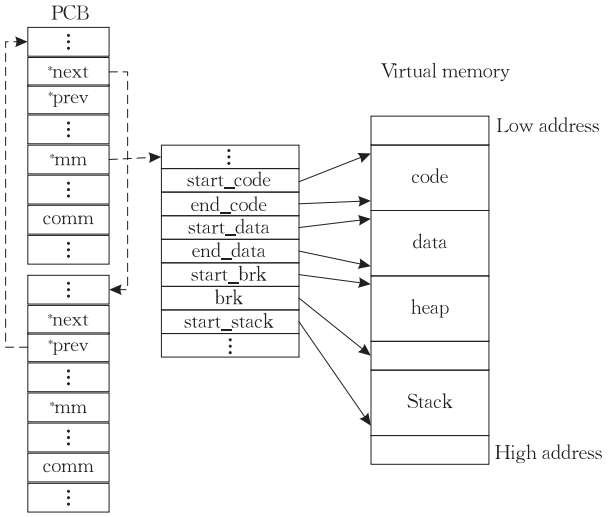


图 3 进程结构

制块含有运行进程的名字,根据名字字段到该进程控制块起始地址的偏移,可知道进程的名称.进程控制块中有指向内存管理结构的指针,内存管理结构是操作系统管理进程虚拟地址空间分配的结构体.进程的虚拟地址空间包括代码段、数据段、堆、栈、参数、环境变量等.进程在运行时代码段是不会发生变化的,如果本次运行时进程的代码段与上次运行时不同,说明进程对应的程序发生了变化.因此,进程的完

整性度量通过计算进程代码段的 hash 值来检验.

模块结构如图 4 所示. 模块在系统启动时进行加载，有些模块是在系统启动之后通过用户空间程序进行加载的，不管是哪种方式，其内存空间的分配方式是相同的. 模块是一种对象文件，不能直接运行，因为它要使用内核导出的符号，是一种可重定向格式. 模块被读入内存后，由内核进行模块的初始化操作. 模块在内存中有代码区域和数据区域，而数据区域有只读数据和可读写数据. 通过计算不可变的区域，即代码区域和只读数据区域得到 hash 值，能够较好地表示出模块的唯一性. 每个模块都有一个双向链表，链接到前一个模块和后一个模块，还包括一个名字字段，表示模块的名字. 通过遍历双向链表可得到系统所加载的模块，再根据名字字段到该模块起始地址的偏移，便访问到某个特定的模块内容.

动态库是程序运行时才被装载入内存的，可以通过动态链接或动态加载来使用，它的存放位置位于使用它的进程的虚拟内存区域，结构如图 5 所示，图 5 显示了进程虚拟内存区域的布局. 程序运行过程中需要调用动态库的函数时，操作系统会搜索动

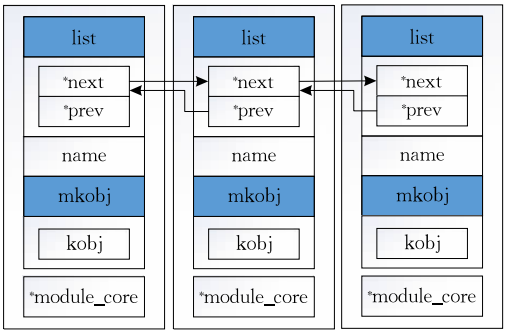


图 4　模块结构

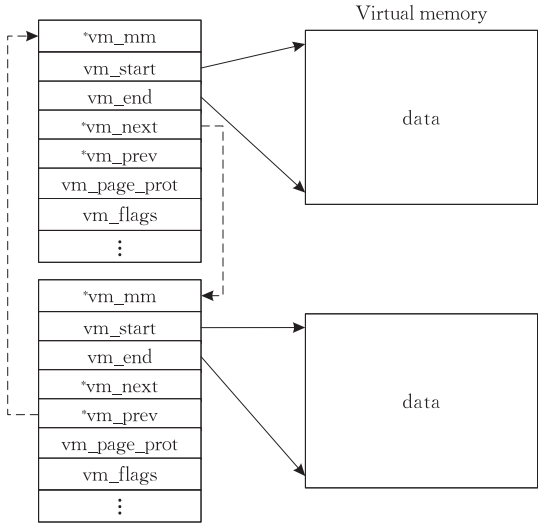


图 5　虚拟内存区域结构

态库在磁盘上的存放位置，将其加载到虚拟内存区域. 每个虚拟内存区域可分为 4 类：可读、可写、可执行、可共享. 动态库代码段的存放位置会被标记为可执行区域，通过计算这部分的 hash 值来度量动态库.

4　实　现

本文实现的 IVirt 采用 x86 架构的 CPU，而且 CPU 需要有虚拟化支持，目前大多数处理器都能满足这种要求. 主机操作系统采用 Ubuntu 12.04.1，使用 gcc 编译器进行开发. 虚拟机监视器使用 Xen 4.1.4. Xen 提供两种虚拟化模式：半虚拟化和硬件辅助虚拟化. 半虚拟化需要虚拟机内核的支持，硬件辅助虚拟化则不需要修改虚拟机的内核. 为了能够运行多种虚拟机操作系统，使度量更具有通用性，IVirt 采用硬件辅助虚拟化模式.

实现 IVirt 的关键技术难点主要有两个：一个是要从主机(Host)中得到虚拟机(Guest)所需内容的内存页. 虚拟机操作系统的内核符号表含有内核符号变量的虚拟地址，但是直接利用虚拟地址无法访问到所需的虚拟机内存空间. IVirt 通过将虚拟地址转换为物理页框号，如图 2 所示，再利用 Xencontrol 得到虚拟机内存页.

第 2 个是准确定位到被度量对象的位置，这里存在语义鸿沟问题. 得到的内存页数据都是底层的二进制形式，根据虚拟机操作系统的内核数据结构，将这些底层的二进制数据对应到高层的语义中. 数据结构中各成员变量的值利用该成员变量的偏移量进行查找. 由于数据结构中存在着指针，指针指向的位置也是虚拟地址，如前所述，直接利用虚拟地址无法得到所需的内存，所以虚拟地址都需要转换为物理页框号. 因此每一次通过虚拟地址对虚拟机内存的访问都要进行地址转换，最终找到所要度量的代码段位置. 得到代码段的数据后，IVirt 采用 sha1 算法来计算 Hash 值.

5　实验及评价

本实验将对所提出的完整性度量方法进行功能评估和性能评估. 功能评估是为了测试所提方法是否能够从虚拟机外部检测出虚拟机内部程序遭到篡改，性能评估是为了评价所提方法对磁盘 I/O、操作系统产生的性能代价，包括主机的性能和虚拟机的性能. 性能评估采用对比的方式进行，将本文所提方

法与无完整性度量的情况进行对比,以此说明本文所提方法引入的性能代价.实验组为虚拟机自省方式的完整性度量,对照组为无度量方式.实验在服务器上进行,硬件环境配置如表 1 所示,软件环境默认配置如表 2 所示.

表 1 硬件配置

配置项	参数	配置项	参数
CPU	Intel® Xeon X5650 24 核	三级缓存	12 288 KB
频率	2666.854 MHz	内存	33 545 332 KB
一级缓存	32 KB	硬盘	1 TB
二级缓存	256 KB		

表 2 软件配置

主机配置	虚拟机配置
操作系统 Ubuntu 12.04.1	操作系统 Ubuntu 12.04.1
虚拟机监视器(VMM)Xen 4.1.4	CPU1 核
	内存 1 GB
	磁盘 20 GB

5.1 功能检测

功能检测是为了评估本文所提方法是否能够发现被度量对象遭到篡改.这里功能检测将对进程的度量和模块的度量进行说明,而其他被度量对象,如动态库,其度量的实现原理是类似的,采用同样的方法也能够达到一样的效果.

在进程度量的实验中,通过对系统中经常存在的 SSH 程序发起攻击来说明本文方法的有效性.SSH 是远程连接的程序,攻击者通过攻击该程序使系统留下后门,方便攻击者再次登录.实验结果如图 6 所示,图 6(a)显示攻击之前采用本文的机制对系统中的所有程序进行度量,计算出所有进程的 hash 值.图 6(b)显示攻击之后的情形,攻击者对 SSH 程序攻击成功后,该程序已被替换成攻击者修改过的程序,其 hash 值也会发生改变.

在模块度量的实验中,采用对测试模块篡改的方式进行测试,因为任何拥有特权的用户都可以插入模块.实验采用测试模块是因为需要模拟对模块进行篡改攻击,而不影响系统中其他模块的运作.先插入测试模块,利用本文所提方法对系统中的所有模块进行度量.在对测试模块进行篡改后,可以检测到其 hash 值发生了变化.实验结果如图 7 所示.

图 7(a)是篡改之前模块的 hash 值,图 7(b)是篡改之后的 hash 值.从图 7 中可以看出,模块篡改前和篡改后其名称语义并没有发生变化,而测试模块的 hash 值在篡改后不同了.这说明本文所提方法确实能够从虚拟机外部检测出虚拟机内部模块的更改.这种篡改攻击的方式能够通过完整性度量检测出来.

```
[ 1996] sudo
sha1 is 8ba27dd81fe7389ea940f971f3216baab2fad90a

[ 1997] su
sha1 is feb72e8e05f50f88bddbc06f134415af806451b3

[ 2005] bash
sha1 is 6cf3ccfaccfd8db959a1b1a4650608d83661baaa

[ 2016] gnome-screensav
sha1 is 4dffbeeaed808389dcb0c20898f1ba90d3e9039c

[ 2027] update-notifier
sha1 is 2614a0f9cd979ef537441691084bfa097f3fa792

[ 2049] system-service-
sha1 is 15ec55efd3bf22b17a8dea89be75f0fde4ab7bbf

[ 4325] deja-dup-monito
sha1 is a0af2d78d841e751db1f17086a3aebd526ab7feb

[13350] sshd
sha1 is 21a33f53d98dab0751be58b547ea25fbec0249c1
```

(a) 攻击之前实验结果

```
[ 1996] sudo
sha1 is 8ba27dd81fe7389ea940f971f3216baab2fad90a

[ 1997] su
sha1 is feb72e8e05f50f88bddbc06f134415af806451b3

[ 2005] bash
sha1 is 6cf3ccfaccfd8db959a1b1a4650608d83661baaa

[ 2016] gnome-screensav
sha1 is 4dffbeeaed808389dcb0c20898f1ba90d3e9039c

[ 2027] update-notifier
sha1 is 2614a0f9cd979ef537441691084bfa097f3fa792

[ 2049] system-service-
sha1 is 15ec55efd3bf22b17a8dea89be75f0fde4ab7bbf

[ 4325] deja-dup-monito
sha1 is a0af2d78d841e751db1f17086a3aebd526ab7feb

[20271] sshd
sha1 is d20def9419157cf407d52e2657faf1b5650b9a00
```

(b) 攻击之后实验结果

图 6 进程度量检测结果

```
moduletest
sha1 is 9718baab9f2aa1ff82adb085869113a8b8629f15

bnep
sha1 is 1793eca1441e1a02e142cbd559962153ea2a6c4e

bluetooth
sha1 is 904bfd42c09f55cd8c4d775a8a94a90cbc5edb82

ppdev
sha1 is d66d72984b45cf5bd178ca59b93059899b1f8193

psmouse
sha1 is 65c8a4cd4ea52523ff5386da927e145c7b5d179d

parport_pc
sha1 is e765686311e7456a52b6773f00dcfa2428ceffc0

i2c_piix4
sha1 is 4dab5b6702cbe42d2d06129580e3560edd71a9ec
```

(a) 攻击之前实验结果

```
moduletest
sha1 is ec70fe94539d0cd73565e214ff286c11d76de105

bnep
sha1 is 1793eca1441e1a02e142cbd559962153ea2a6c4e

bluetooth
sha1 is 904bfd42c09f55cd8c4d775a8a94a90cbc5edb82

ppdev
sha1 is d66d72984b45cf5bd178ca59b93059899b1f8193

psmouse
sha1 is 65c8a4cd4ea52523ff5386da927e145c7b5d179d

parport_pc
sha1 is e765686311e7456a52b6773f00dcfa2428ceffc0

i2c_piix4
sha1 is 4dab5b6702cbe42d2d06129580e3560edd71a9ec
```

(b) 攻击之后实验结果

图 7 模块度量检测结果

5.2 性能检测

本节通过性能检测实验,分别对虚拟机的性能和主机的性能进行测试.在虚拟机的性能测试中,对比不进行完整性度量和进行完整性度量的性能,来

说明本文的方法并不会带来很高的代价. 实验分为底层级测试、操作系统级测试和应用程序级测试，分别采用 Nbench[17]、Hbench[18] 和 TPCC-UVa[19] 测试工具. 而在主机的性能测试中，测量完成完整性度量所消耗的时间. 为说明本文方法引入的代价较小，性能实验还对采用系统调用拦截机制的完整性度量方法进行对比.

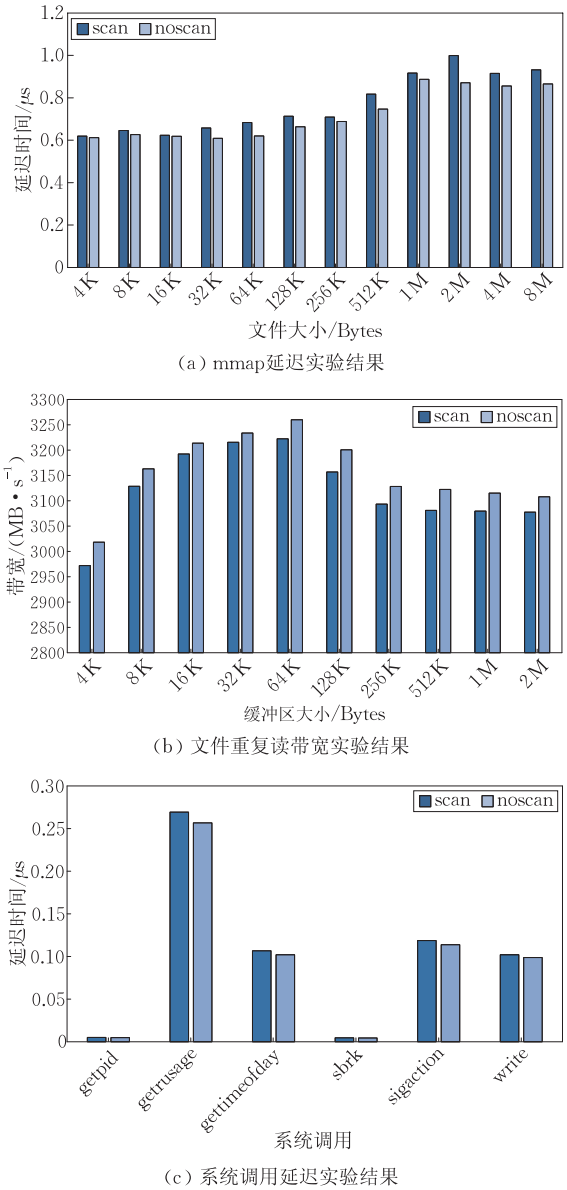
Nbench 是测试系统的 CPU、FPU 和内存管理的测试工具，基于 BYTEmark[17] 测试程序的第 2 版，测试算法没有改变，主要是能够更好地适应 64 位机器的测试. 实验的软件配置为表 2，实验中通过设置不同的扫描时间间隔来测试完整性度量对虚拟机内底层性能的影响，测试中选择针对进程的完整性度量进行. 结果如表 3 所示.

表 3　Nbench 测试结果

测试	1 s 时间间隔 (Iterations/s)	2 s 时间间隔 (Iterations/s)	无完整性度量 (Iterations/s)
NUMERIC SORT	1230.4	1261.8	1312.6
STRING SORT	713.72	741	773.64
BITFIELD	5.31E+08	5.51E+08	5.72E+08
FP EMULATION	274	284.56	296.76
FOURIER	33797	34947	36287
ASSIGNMENT	40.215	41.653	43.325
IDEA	8163.8	8422.6	8764
HUFFMAN	2805.3	2903.8	3011.6
NEURAL NET	73.52	76.32	79.288
LU DECOMPOSITION	1945.6	2024.4	2098.6

表 3 中的结果共有 10 项测试，每项测试表示每秒能够完成的测试个数，进行多次测试取平均值，数值越大表明处理速度越快. 在这 10 项测试当中，无完整性度量的性能略高于 2 s 时间间隔的度量，而 2 s 时间间隔度量的性能略高于 1 s 时间间隔度量的性能. 这是由于无完整性度量时，物理 CPU 资源供虚拟机使用，并没有其他程序占用该资源. 当特权域中有程序要占用 CPU 资源时，会优先将资源分配给特权域中的程序使用，虚拟机中的程序需要暂时的等待，等特权域中程序运行完后虚拟机中的程序再继续运行，所以在虚拟机外部没有程序运行时的性能会稍高一些. 而且度量程序的间隔时间越短，该程序的 CPU 占用时间就会越长，测试程序的性能影响就会相应增高. 但是这种性能开销的相差并不会太大，从总体上可以看出，进行完整性度量与不进行完整性度量每秒完成的测试个数保持在同一数量级上. 在 1 s 时间间隔的度量中，STRING SORT 测试与无完整性度量相比引入的性能消耗是最高的，处理速度降低了约 7.75%，而在 2 s 时间间隔的度量中，STRING SORT 测试的处理速度降低了约 4.22%. 所以度量间隔时间越长，引入的代价越小.

Hbench[18] 是一个操作系统级的测试工具，能够测试内存的延迟和带宽. 实验通过对常用的文件映射系统函数 mmap 和文件读取系统函数 read 进行测试，来评估完整性度量给虚拟机内系统函数带来的性能影响. 为了进行对比，实验由每隔两秒进行一次完整性度量和无完整性度量组成，其结果如图 8 所示.



(a) mmap延迟实验结果

(b) 文件重复读带宽实验结果

(c) 系统调用延迟实验结果

图 8　延迟和带宽实验结果

在图 8(a)中，横坐标表示映射的磁盘文件大小，纵坐标表示从 mmap 操作开始到 mmap 操作结束所用的延迟时间. 函数 mmap 将磁盘文件数据直接映射到用户进程的地址空间，使文件的读取无需在内核空间和用户空间切换. 总体上来说，映射的文件越大，mmap 操作的延迟时间也就越大，但是并不

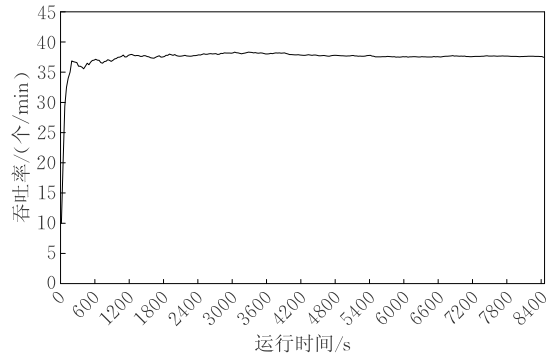
是严格单调,而是会有局部的上下波动,在 1 MB 文件大小之后延迟时间趋于平稳.在完整性度量下,1 MB文件大小之后的延迟时间都在 0.9 μs 以上,无完整性度量下的延迟时间都在 0.8 μs 以上.由于 2 s 间隔的完整性度量需要暂停虚拟机,计算出所有进程的 hash 值,所以其延迟时间比无完整性度量的延迟时间稍大.从 4 MB 到 8 MB 文件的测试中,以完整性度量情况下的最大延迟时间为例,是在 2 MB 文件大小处,比无完整性度量的情况只增加了 0.128416 μs 的延迟时间,所以总体来说采用完整性度量对 mmap 的性能影响是较小的.

文件重复读带宽是度量具有缓冲区功能的文件读取操作所能达到的带宽.应用程序读取文件时会调用文件读函数 read(),该函数会先将文件数据读入内核缓冲区,待缓冲区填满后再将文件数据读入变量.在图 8(b)的实验结果中,横坐标表示设置的文件系统缓冲区大小,纵坐标表示带宽.设置文件大小为固定值 2 MB,缓冲区的大小从 4 MB 递增到2 MB.从图中可以看出,随着缓冲区大小的增加,带宽先逐渐增大,之后再下降,最终会趋于稳定值.两种方法都是在缓冲区大小设置为 64 MB 时达到最大值,说明缓冲区设置为 64 MB 能发挥文件读取的最大性能,采用 2 s 间隔的完整性度量并不会影响最大文件重复读带宽的特性.在 64 MB 的缓冲区条件下,2 s 间隔的完整性度量比无完整性度量下降的带宽约为 1.15%,下降幅度最大的情况是在 4 MB 的缓冲区,但是幅度也仅为 1.53%,可见,2 s 间隔的完整性度量对文件重复读的带宽影响是较小的.

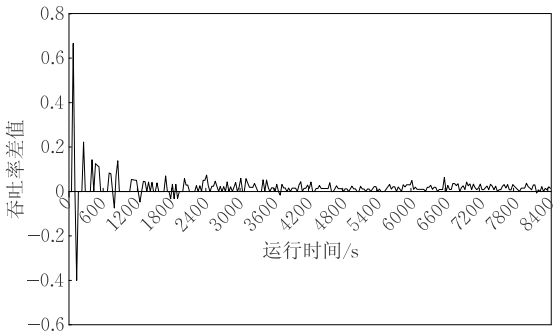
实验还测试了其它一些系统调用函数的延迟时间,结果如图 8(c)所示.函数 getpid 获取调用进程的 id 号,getrusage 获取调用进程或调用线程的资源使用情况,gettimeofday 获取系统时间,sbrk 增加程序的数据空间,sigaction 改变进程收到某个信号后所执行的行为,write 将缓冲区数据写入文件.在这 6 个系统调用当中,getrusage 测试引入的时间最高,2 s 间隔的完整性度量比无完整性度量多花费了 0.0127 μs.所以可以看出,完整性度量引入的时间延迟代价是较小的.

如图 9 所示的是采用 TPCC-UVa[19] 对本文的架构中虚拟机内部性能进行测试,该基准测试程序模拟一个在线交易处理环境,能够较好地反映出系统在应用程序级别上的负载情况.实验中设置数据仓库个数为 3,每个数据仓库的终端数为 10,度量时间为 120 min.

如图 9(a)所示是虚拟机对性能负载新订单的处理能力,该实验对虚拟机中进程的完整性每隔两秒度量一次.图 9 中曲线表示随着虚拟机的运行,系统每分钟能处理的订单数量.曲线的前面部分是订单处理的上升期,在模拟刚开始时,订单数较少,还未达到系统能够处理订单的稳定值,所以曲线不断上升.随着订单数不断增多,系统每分钟能够处理的订单数趋于稳定状态,这说明订单处理达到了系统所能承载的能力.为了与无完整性度量时的性能负载进行对比,图 9(b)显示了不进行完整性度量与每隔两秒完整性度量的吞吐率差值图.曲线在前面部分的波动较大是因为处于订单处理的上升期,不进行完整性度量和进行完整性度量的系统都在模拟的初始阶段.在这个阶段,模拟的远程终端进程会不断启动,两种情况下相同的时间范围内系统启动的进程数不同,所处理的订单数是不同的.在无完整性度量的情况下,不需要对虚拟机进行暂停操作和度量这些进程,而在有完整性度量的情况下,虚拟机会被暂停,度量进程也会有相应的时间消耗,导致这两种情况在上升期的时间范围内吞吐率出现较大的波动.而当两个系统都处于稳定期后,完整性度量与不进行完整性度量的吞吐率差值在 ±0.1 之间波动,所以这种性能代价并不高.



(a) 虚拟机吞吐率



(b) 吞吐率差值

图 9 吞吐率实验结果

对主机性能影响的测试如图 10 所示，分别是对进程和模块进行一次度量产生的时间消耗. 实验中设置虚拟机的内存不断增大，分别对不同内存的虚拟机测量从度量开始到度量完成所消耗的时间. 进程度量的时间消耗在 0.08 s～0.09 s 之间范围内波动，模块度量的时间消耗在 0.05 s～0.06 s 之间范围内波动，所以度量的时间消耗较小. 而系统中的进程数量一般会比模块的数量多，所以进程度量的时间消耗会略高于模块度量的时间消耗. 此外，这两个结果还表明随着虚拟机内存的增大，度量的消耗时间并没有明显的呈递增或递减趋势，所以内存大小的变化对于本文的度量方法影响较小.

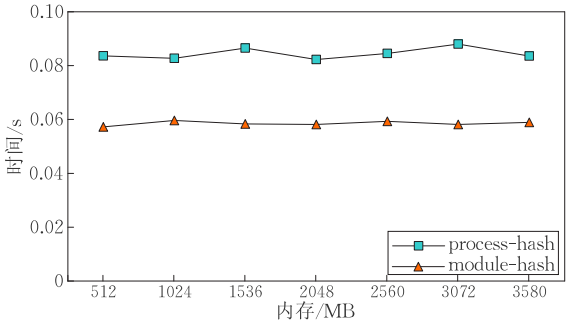


图 10　度量时间消耗

除了以上对引入完整性度量和无完整性度量的比较外，本文还将 IVirt 与采用系统调用拦截方式的完整性度量进行实验对比分析，来说明本文的方法引入的性能代价较小. HIMA[3] 通过拦截系统调用，在进程启动前进行完整性度量. 为方便与其对比，实验采用 HIMA 中使用的性能测试工具 UnixBench，测试虚拟机的性能，实验结果如图 11 所示.

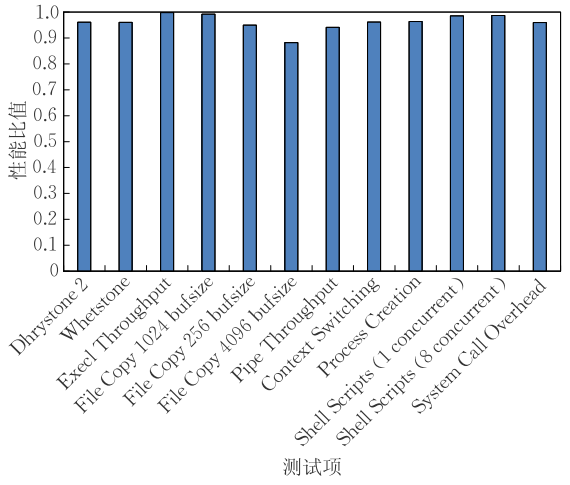


图 11　UnixBench 测试结果

图 11 中横坐标表示测试项，纵坐标表示采用 IVirt 度量测试结果与无完整性度量测试结果的比值. 在 HIMA 的测试结果[3] 中，引入负载较大的是 Execl Throughput、Process Creation、Shell Scripts (8 concurrent). 而从图 11 中可以看出，IVirt 引入的负载并不十分明显. HIMA 通过缓存内存页的 hash 值来降低计算负载. 测试 Execl Throughput 的有缓存和无缓存分别引入的负载是 31% 和 75%. 测试 Process Creation 的有缓存引入的负载是 21%，无缓存时引入的负载和有缓存时的相差不大. 测试 Shell Scripts (8 concurrent) 的有缓存和无缓存分别引入的负载是 43% 和 99%. 而 Ivirt 在 Execl Throughput 测试下引入负载是 0.22%，Process Creation 测试引入的负载是 3.62%，Shell Scripts (8 concurrent) 测试引入的负载是 1.30%，均低于 HIMA 引入的负载值. 由于 HIMA 在发生系统调用时就触发 hash 值的计算，而 IVirt 是在程序加载到内存中后才计算 hash 值，避免了系统调用引入的代价. 因此，IVirt 引入的性能代价比系统调用拦截引入的性能代价较小.

6　结　论

在云计算中，虚拟机内的程序存在被篡改的可能. 本文分析了虚拟机运行时的完整性问题，提出了基于虚拟机自省的完整性度量方法 IVirt，通过将虚拟机内存的内容映射到主机上，实现对虚拟机内部的完整性度量. 该方法将度量软件与被度量系统分开，实现了安全隔离，避免暴露监控软件. 通过实验表明，在功能上本文的方法能够检测出被度量对象是否遭篡改，实现对代码完整性检测，在性能上本文的方法在虚拟机中的底层级测试、系统级测试和应用程序级测试以及主机测试所引入的代价都较小. 与采用系统调用拦截的完整性度量方法对比，本文方法的性能代价也较小.

参　考　文　献

[1] Sailer R, Zhang X, Jaeger T, van Doorn L. Design and implementation of a TCG-based integrity measurement architecture//Proceedings of the 13th USENIX Security Symposium. San Diego, USA, 2004: 223-238

[2] Berger S, Cáceres R, Goldman K A, et al. vTPM: Virtualizing the trusted platform module//Proceedings of the 15th USENIX Security Symposium. Vancouver, Canada, 2006: 305-320

[3] Azab A M, Ning P, Sezer E C, Zhang X. HIMA: A hypervisor-based integrity measurement agent//Proceedings of the 2009 Annual Computer Security Applications Conference. Honolulu, USA, 2009: 461-470

[4] Chen P M, Noble B D. When virtual is better than real//Proceedings of the 8th Workshop on Hot Topics in Operating Systems. Munich, Germany, 2001: 133-138

[5] Smith S W, Weingart S. Building a high-performance, programmable secure coprocessor. Computer Networks, 1999, 31(8): 831-860

[6] Li S J, He Y P. A privacy-preserving integrity measurement architecture//Proceedings of the 2010 3rd International Symposium on Electronic Commerce and Security. Guangzhou, China, 2010: 242-246

[7] Liu T, Agrawal P. A trusted integrity measurement architecture for securing enterprise network//Proceedings of the 2011 International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11. Changsha, China, 2011: 726-731

[8] Jaeger T, Sailer R, Shankar U. PRIMA: Policy-reduced integrity measurement architecture//Proceedings of the 11th ACM Symposium on Access Control Models and Technologies. Lake Tahoe, USA, 2006: 19-28

[9] Baliga A, Ganapathy V, Iftode L. Detecting kernel-level rootkits using data structure invariants. IEEE Transactions on Dependable and Secure Computing, 2011, 8(5): 670-684

[10] Szekeres L, Payer M, Wei T, Song D. SoK: Eternal war in memory//Proceedings of the 2013 IEEE Symposium on Security and Privacy. Berkeley, USA, 2013: 48-62

[11] Stelte B, Koch R, Ullmann M. Towards integrity measurement in virtualized environments—A hypervisor based sensory integrity measurement architecture (SIMA)//Proceedings of the 2010 IEEE International Conference on Technologies for Homeland Security. Waltham, USA, 2010: 106-112

[12] Huh J H, Montanari M, Dagit D, et al. An empirical study on the software integrity of virtual appliances: Are you really getting what you paid for?//Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. Hangzhou, China, 2013: 231-242

[13] Garfinkel T, Pfaff B, Chow J, et al. Terra: A virtual machine-based platform for trusted computing//Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York, USA, 2003: 193-206

[14] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection//Proceedings of the Network and Distributed System Security Symposium. San Diego, USA, 2003: 1-16

[15] Payne B D, Carbone M D P de A, Lee W. Secure and flexible monitoring of virtual machines//Proceedings of the 23rd Annual Computer Security Applications Conference. Miami Beach, USA, 2007: 385-397

[16] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization//Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York, USA, 2003: 164-177

[17] Grehan R, Thompson T, Franklin C Jr, Stewart G A. Introducing the new BYTE benchmarks. Byte, 1988, 13(6): 239-266

[18] Brown A B, Seltzer M I. Operating system benchmarking in the wake of Lmbench: A case study of the performance of NetBSD on the Intel x86 architecture//Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. Seattle, USA, 1997: 214-224

[19] Llanos D R. TPCC-UVa: An open-source TPC-C implementation for global performance measurement of computer systems. SIGMOD Record, 2006, 35(4): 6-15

LIN Jie, born in 1987, Ph. D. candidate. His current research interests include computer network, information security.

LIU Chuan-Yi, born in 1982, Ph. D., associate professor. His research interests include cloud computing and cloud security, data security and data protection.

FANG Bin-Xing, born in 1960, Ph. D., professor, member of Chinese Academy of Engineering. His current research interests include network security, information content security.

Background

The issues this paper focus on belong to the runtime integrity measurement for virtual machine in the area of information security. Virtual machine security in cloud computing gets more and more attention. Currently most methods of integrity measurement are installing measurement software in the virtual machine. The mechanism used is the event trigger, i. e. before the binary file is loaded, the measurement software is triggered to measure it. A problem in this method is that the measurement software and the measured system are in the same domain. Attacker is able to

detect the existence of measurement software, so the measurement software is vulnerable to attack. Researchers also proposed some methods out of VM. These methods modified the virtual machine monitor to add hooks to intercept the events. When certain event occurs, the corresponding binary is measured to identify whether it is tampered with. But logging a lot of events will introduce some performance overhead. Especially when system calls are frequently invoked in the virtual machine, events intercept mechanism will log a large number of records, which will affect the performance of virtual machine. Monitoring virtual machine also encounters semantic gap problem. The data gained from outside of virtual machine is binary. In order to bridge this gap, researchers make use of predefined data structure to infer the high level meaning.

This paper proposes an integrity measurement mechanism based on virtual machine introspection, called IVirt (Integrity for Virtualization), and a prototype was implemented to measure the integrity of applications in the virtual machine from outside of the virtual machine. IVirt can discover the modification of programs in the virtual machine, including processes, modules, and dynamic link libraries at runtime. There is no need to install any software in the virtual machine. Because of the isolation between IVirt and measured system, IVirt will not be compromised by attacker in measured system. IVirt reads kernel symbols and conducts address translation to access virtual machine memory. According to the appropriate kernel structure, IVirt infers the code location in virtual machine. It measures independent components in the memory. The measurement time can be periodical or random. The experiment results show that IVrit has the ability of finding the programs tampered with, and it has little effects on the performance.

This research is supported by the National Natural Science Foundation of China under Grant No. 61202081, and the National High-Tech R&D Program (863 Program) of China under Grant No. 2012AA012606. The project aims to study the virtual machine security in cloud computing environment.