APM：适用于 IaaS 平台的 agent 保护机制

樊佩茹，赵波，倪明涛，陈治宏
（武汉大学国家网络安全学院空天信息安全与可信计算教育部重点实验室，湖北 武汉 430072）

摘 要：在 IaaS 平台中，虚假数据的存在将对测评结果造成混淆，无法为用户给出公平公正的平台选择依据。针对该问题，提出一种适用于 IaaS 平台的测试代理 agent 保护机制（APM, agent protection mechanism），在不需要额外软硬件支持的条件下保证 agent 的完整性和命令执行的正确性；同时提出一种基于质询的 APM 有效性验证方法，及时发现失效 APM 所在 IaaS 节点以止损。实现了基于 APM 的实验环境，对 APM 的有效性和性能开销进行测试。实验结果表明，该机制可以有效保护 agent 的完整性及其执行命令的正确性，且对 IaaS 平台引入的性能代价较小。

关键词：IaaS 平台；测试；代理；度量
中图分类号：TP391
文献标识码：A
doi: 10.11959/j.issn.1000-436x.2018069

APM: agent protection mechanism applied for IaaS platform

FAN Peiru, ZHAO Bo, NI Mingtao, CHEN Zhihong
Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education,
School of Computer Science/School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China

Abstract: The interference of false or fake test data on IaaS platform will contaminate the evaluation results, confusing users' choices for IaaS services. To solve this problem, an agent protection mechanism (APM) for IaaS platform test environment was proposed. It ensured the integrity and commanded validity of the agent without additional hardware or software. Also an effectiveness verification approach based on requests was presented to detect APM failure problems timely. An experiment environment according to APM was implemented to evaluate the effectiveness and the performance overhead. Experimental results show that the APM is effective in protecting agent integrity and command validation, and its performance overhead is minor.

Key words: IaaS platform, test, agent, measurement

1 引言

IaaS（infrastructure as a service）平台为用户提供高度集中、可动态扩展的计算存储资源，是支撑云平台上层服务 PaaS（platform as a service）和 SaaS（software as a service）的基础。用户通过按需租用降低业务部署与管理成本，但该云服务模式中用户丧失了对存储和运行在云端数据及应用的物理可见性与可控性，需要依赖 IaaS 平台提供商（IPP, IaaS platform provider）部署安全防护措施[1]。然而，为避免遭到针对性攻击，IPP 提供商一般不愿意公开 IaaS 节点的具体配置与安全信息，导致用户对平台缺乏信任，这已成为近年来云计算发展过程中遇到的一大阻碍。

收稿日期：2017-08-10；**修回日期**：2018-02-28
通信作者：赵波，zhaobo@whu.edu.cn
基金项目：国家高技术研究发展计划（“863”计划）基金资助项目（No.2015AA016002）；国家重点基础研究发展计划（“973”计划）基金资助项目（No.2014CB340600）
Foundation Items: The National High Technology Research and Development Program of China (863 Program) (No.2015AA016002), The National Basic Research Program of China (973 Program) (No.2014CB340600)

对 IaaS 平台做测试，衡量其能否达到用户的性能或可信性需求，是增强用户对平台信心的一种有效方法。业界对 IaaS 平台的需求进行了一些研究和探讨，也有服务等级协议 SLA 等用于定义平台质量或性能指标，但由于目前缺乏综合统一的云测试或评估标准[2,3]，在对 IaaS 平台信任或可信性的理解上仍然存有分歧[4,5]。文献[6]在综合考虑可用性、可靠性与安全性等多项属性的基础上，给出一种云平台可信性的定义，认为云平台的各项可信属性均是在其提供服务过程中不同组件动态执行的表现。因此，对 IaaS 平台执行可信性测试任务时，可分解为对该平台各项可信属性的测试。应用基于可信第三方 TTP（trusted third party）的云测试模型[7~10]到 IaaS 平台中时，在 IaaS 平台上收集数据的测试代理程序（agent）由 TTP 与 IPP 联合部署。在整个测试过程中，IaaS 平台为支撑上层 PaaS 或 SaaS 服务运行的用户虚拟机将正常工作，该测试过程对用户虚拟机而言是透明的。针对这种 IaaS 平台测试环境，考虑 agent 运行在 IaaS 节点虚拟化软件栈层的场景，agent 负责从节点上收集可反映测试指标的测试数据并传输到 TTP 做进一步分析。在这个测试过程中，agent 自身的可信性是产生真实有效测试数据的基础，直接关系到 IaaS 平台测评结果的正确性。现有工作主要关注如何从 IaaS 平台获取测试数据和如何基于平台各项指标的测试数据给出综合评估，并已有部分成果[11]，但总体来看，在 IaaS 平台上部署测试代理程序 agent 时仍存在以下问题和挑战。

1) 在 IaaS 节点上执行测试任务的 agent 缺乏有效的保护机制。这就导致 agent 在不可信运行状态下产生的错误或虚假测试数据仍能被 TTP 接受，IaaS 平台最终的测试评估结果将受到干扰，使 TTP 无法对 IaaS 平台存在的问题采取及时的应对措施以止损，也无法为用户提供公平公正的选择依据。

2) 现有虚拟化环境中的软件保护技术无法直接应用到 IaaS 平台的 agent 保护场景中。IaaS 平台支持着上层 PaaS 和 SaaS 的用户服务，在上线运行后其可用性与可靠性是 IPP 非常重视的要素，但当前虚拟化环境中的软件保护技术需要被保护软件所在主机提供额外的硬件或软件支持，这将大幅增加建立 IaaS 平台测试环境的成本，在实际生产环境中并不容易实现。

实际上，考虑到 agent 将运行在不受 TTP 控制的 IaaS 节点上，IPP 可能不得不假设一个强攻击模型，即仅信任 IaaS 平台的一小部分，并希望该平台中被信任的部分越小越好。之前的研究表明，仅仅假设一个抗篡改的 CPU 芯片可信是合理的[12]，因此，在该假设下，本文提出一种适用于 IaaS 平台的测试代理保护机制。结合 IaaS 平台的特点，首先，给出实现 agent 保护时应达到的设计目标，明确其包括的具体内容，以明确本文的技术依据；然后，设计一个 agent 内核态保护模块（APKM，agent protection kernel module），作为在 IaaS 节点上实施 agent 完整性与命令正确性验证组件的宿主，对 agent 进行保护，并制作可信性状态报告作为 TTP 判断 agent 可信性的依据，APKM 自身的完整性由现代 x86 计算机架构可形成的内存锁机制提供保护，不需要 TTP 提供额外硬件或软件组件的支持；最后，设计一种基于质询的 APM 有效性验证方法，可及时发现失效 APM 所在的 IaaS 节点以止损，尽可能减少不可信 agent 产生错误虚假数据混淆 TTP 分析判断的可能，提高 IaaS 平台测试结果的可靠性。

本文的贡献在于提出了适用于现有 IaaS 平台的测试代理 agent 保护技术，在兼顾平台可用性和可靠性需求的基础上，不需要 TTP 与 IaaS 平台提供额外软硬件组件支持即可达到保护 agent 可信性的目标，解决了 IaaS 平台中不可信 agent 产生错误虚假数据干扰测试评估结果的问题。

2　相关工作

业界及相关机构在云平台测试方面开展了部分研究工作，King 等[7]针对云服务开发测试脚本，对云平台的自动化自检测试进行研究。Zech 等[8]将风险分析结果与安全测试相结合，提出一种基于模型驱动的云环境安全测试方法。Khan 等[9]设计部署一个基于可信第三方的完整性验证端 TIV，验证云平台中物理节点的可信性。Pham 等[10]提出了一个软件测试框架 CloudVal，验证云平台虚拟化环境的可信性。谢亚龙等[13]分析了云取证技术所面临的挑战，并提出一种 IaaS 云模型下的取证框架 ICFF。此外，也有一些项目组研究云平台可信属性的测试与认证。ASSERT4SOA 项目[14]针对面向服务的分布式应用程序提供新工具和技术，可评估、认证它们的安全属性。CUMMULUS 项目[15]扩展了 ASSERT4SOA 的工作，为云平台安全认证提供一个

包含可信第三方的新框架，以在云用户、云平台提供商、云服务提供商和认证机构的合作下确保云平台中各项安全属性认证结果的有效性。CloudSec 项目[16]提出一个包含云平台提供商安全需求的校验表，并根据该校验表对云平台各项安全属性进行审计。但上述云平台可信性认证或测试方案中，重点关注了如何获取反映云平台可信性状态的测试数据和对云平台计算出合理的可信性评估结果，这些研究为进一步增强用户对云平台提供了理论指导和现实基础，但对如何保护 agent 的可信性讨论较少。

在基于 TTP 的 IaaS 平台测试环境中，agent 本质上是由 TTP 与 IPP 联合部署在 IaaS 节点虚拟化软件栈层（hypervisor）的测试软件。目前，针对该类软件的保护方法可分为 2 类。1) 硬件辅助实现软件保护。文献[17,18]基于可信硬件 TPM 保护软件的可信性。文献[19]基于 TPCM 设计了针对云平台的可信证据收集方案。TapCon[20]提出一种基于可信第三方的云服务可信性验证方法，利用 SGX 技术保证软件的可信执行。Royan[21]提出一种分布式沙箱，使运行在 SGX 中的应用程序不会遭受来自云平台的攻击。将这类硬件辅助实现的软件保护技术应用于 agent 可信性保护时，需要 IPP 为接受测试的 IaaS 节点提供额外的可信硬件，并需要 TTP 对 agent 进行相应修改以适应该硬件，这会同时增加建立 IaaS 平台测试环境时 IPP 与 TTP 的部署成本，在实际生产环境中并不容易实现。2) 基于可信虚拟化软件栈提供保护。文献[22,23]通过修改 hypervisor 软件代码设计程序隔离方案，保证 agent 代码的完整性。HyperSafe[24]针对 hypervisor 控制流完整性提出全生命周期的自保护方案，但该方案需要修改 hypervisor 代码并重启主机，会对云平台服务造成经济损失。HyperCheck[25]和 HyperSentry[26]通过验证完整性实时保证 hypervisor 不被篡改，防范 rootkit 等攻击，但该类方案利用了 SMM 机制，而 SMM 机制自身存在局限性，需在操作时冻结所有 CPU 核，会为 IaaS 平台带来巨大的性能损耗。可信启动[27]能够保证 hypervisor 启动时的安全性，但无法解决 hypervisor 运行时的安全问题。文献[28]为安全敏感型应用提供了一个安全执行环境 APPSec，根据应用程序的意图保护用户的私有数据和人机交互数据，但它需要一个特定隔离专用操作系统的支持。CQSTR[29]针对 IaaS 平台上的虚拟机集群提出一种可运行服务的云容器，以利于 TTP 判断运行在该容器中服务的安全性。对第 2 类技术，考虑到当前虚拟化软件栈随着功能的增强，其代码量越来越大，要对硬件资源进行管理和分配，也有与虚拟机进行频繁交互，拥有着巨大的攻击面，且虚拟化软件栈自身存着在许多已知或未知的漏洞，在被攻击者利用时，该类保护技术就会失效。因此，当构建基于 TTP 的 IaaS 平台测试环境时，现有技术无法直接应用到 agent 的可信性保护中。

3 Agent 保护机制

基于 TTP 的 IaaS 平台测试环境中，运行在 IaaS 节点虚拟化软件栈层的 agent 面临着极大的安全威胁。攻击者可以借助受控虚拟机发起逃逸攻击，获取物理节点虚拟化软件栈的控制权限，进而篡改测试代理程序 agent 并伪造测试数据以欺骗 TTP，隐瞒该节点已遭到破坏的事实。因此，本文假设一种强攻击模型，攻击者可以利用 IaaS 平台虚拟化软件栈的漏洞或虚拟机的错误配置尝试写入物理节点内存的任意位置，但为了成功发起一次攻击，攻击者不得不注入恶意代码或错误利用已有代码，如 ROP 或 JOP 攻击[30]。注意本文威胁模型不考虑侧信道攻击，如缓存侧信道、时间侧信道或内存总线侧信道等，因为当前该类攻击在 IaaS 平台的复杂计算环境下实现极其困难[31]；也不考虑对 IaaS 节点拥有物理权限而采取的攻击，如 BIOS 木马[32]或冷启动攻击[33]等，因为在实际运营时，IPP 不会分配这种具备所有权限的管理员，并会采取严格的访问控制策略保护平台物理设备[34]。考虑到攻击发生在基于 TTP 的 IaaS 平台测试环境中，攻击的隐蔽性是攻击者考虑的第一要素，攻击者最希望达成的目标是，控制 IaaS 节点，破坏 agent 并篡改测试数据，欺骗 TTP 为一个已经遭到破坏的平台给出正常的测试评估结果，而在整个过程中，TTP 与 IPP 均无法察觉到该攻击。因此，本文不考虑攻击者采用拒绝服务等容易被检测到的手段进行破坏的情况。

在基于 TTP 的 IaaS 平台测试中，考虑到以上攻击模型时，所提 agent 可信性保护机制 APM 应满足以下 4 个设计目标。

1) 支持 agent 的完整性保护。支持在 agent 启动时和运行过程中的完整性验证，在发现 agent 完整性被破坏时及时通知 TTP 与 IPP 采取应对措施。

2) 确保 agent 执行命令的正确性。在 agent 运行过程中能验证其执行命令的正确性，只有通过正确性验证的命令才会被执行。

3) 支持 agent 的更新与扩展。APM 应在不依赖其他任何特殊软硬件支持的情况下达到保护 agent 可信性的目的，同时最小化对节点虚拟化软件栈的修改，并最小化对平台运行的影响。对 IaaS 平台做测试时，一般采用多个指标进行评估，且出于对软件可维护性的考虑，部署到节点上的 agent 可能要进行不定时更新，因此，需要保证 APM 支持 agent 的更新与扩展。

4) 支持 APM 的有效性验证。TTP 与 IPP 应能在任意时刻发起质询并验证 APM 的有效性以及时发现 APM 被破坏的情形，并采取应对措施。

3.1　体系结构

将基于 TTP 的云测试模型应用到 IaaS 平台测试时，APM 通过在 IaaS 节点内核空间增加一个 APKM 为其提供保护，具体 APM 的结构如图 1 所示。

图 1 中实线箭头表示传统基于 TTP 的 IaaS 平台测试执行流程①~③。Agent 从测试对象集合收集测试数据并发送到 TTP，TTP 对测试数据进行综合分析给出评估结果，其中，测试对象是 IaaS 节点上影响平台指标的资源。由于本文重点不在于研究如何从被测对象上获取反映指标的测试数据，所以这里不对测试对象集合及测试指标做详细讨论。

图 1 中点线箭头表示引入本文 APM 后新增的执行流程①~④。利用 APKM 对 agent 实施保护，APKM 自身的完整性由 x86 计算机架构可形成的内存锁机制提供保护，内存锁机制的有效性依赖于 IaaS 节点控制寄存器 CR0 中写保护位 WP 状态的正确性，然后以 WP 的置位正确性为起点，将信任关系扩展到 APKM，再到 agent，从而保证从节点上获取测试数据的可信性。由于部署在节点上负责具体测试任务的 agent 会根据被测指标的不同而变化，

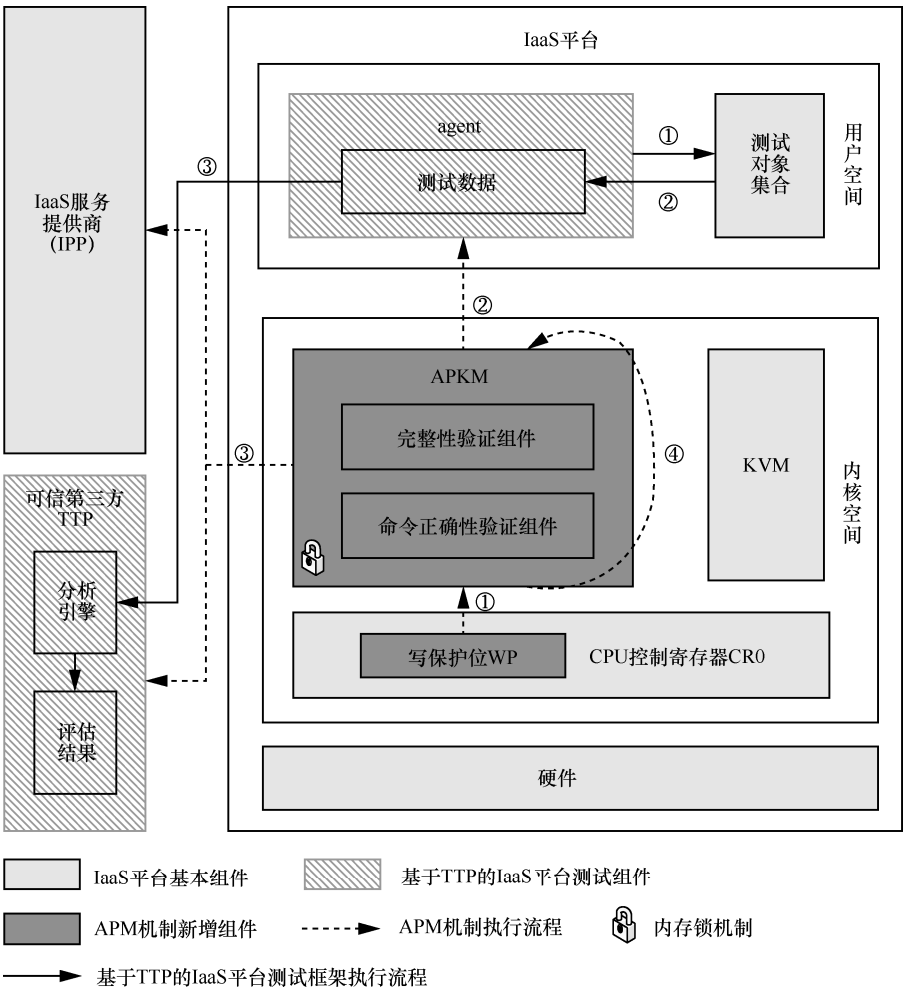


图 1　APM 结构示意

保存在 APKM 中与 agent 相关的预存数据也需要随之进行更新，这就要求存在一种 APKM 数据的动态更新机制④，使其支持 agent 的更新与扩展。图 1 只展示了一个 IaaS 节点上的 APM 结构，实际在构建 IaaS 平台测试环境时，TTP 与 IPP 每个 IaaS 节点上都将部署 APM。APKM 验证 agent 的完整性和执行命令的正确性后，将该验证结果记录到 agent 可信性状态报告中；TTP 与 IPP 也将周期性地或在任意时刻发起对节点控制寄存器 CR0 中 WP 置位状态的质询，获得 WP 状态质询结果。该质询结果和 agent 可信性状态报告都将作为 TTP 判断测试数据可信性的依据。

接下来，分别对 APM 架构中涉及的关键方法进行做进一步描述。

3.2 APKM 完整性保护方法

内核态模块 APKM 的完整性依赖于现代 Intel x86 架构可形成的内存锁机制[35]提供保护，关键在于实现 APKM 的正确初始化，本文假设 TTP 与 IPP 在 IaaS 节点上联合加载 APKM 时，其初始环境可信，一切攻击都发生在 APKM 加载完成后的任意时刻。APKM 被载入 IaaS 节点内核内存空间后，首先标记 APKM 代码与静态数据所在的内存页面为只读的写保护属性，标记 APKM 内存页的页表项所在页表页为只读的写保护属性，然后打开 IaaS 节点控制寄存器 CR0 中的写保护位 WP（将 WP 位置 1），此时，完成了 APKM 的初始化工作，基于内存锁机制的 APKM 保护技术开始生效。在此后的任意运行时刻，只要 WP 位保持打开状态，任何对 APKM 代码与静态数据的写操作都将遭到拒绝，这就保护了 APKM 的完整性不会被破坏。

APKM 负责度量和校验 agent 启动时与运行时的完整性，它必然拥有预先记录的 agent 在可信环境中启动与运行时的完整性度量结果作为基准值。但根据 IaaS 平台测试指标的不同，负责具体测试任务的 agent 时常面临升级或变更情形，这就要求同时对 APKM 中 agent 的基准值进行更新；此外，随着时间的推移，TTP 也可能存在升级 APKM 的需求。但受到内存锁机制保护的 APKM 将拒绝任何对其受保护内存页面的写操作，无论这个写操作是正常请求还是恶意篡改，也无论发起该操作的是普通用户或是系统管理员，这就为 APKM 的正常更新造成困难。本文采用原子更新操作解决该问题，当修改 APKM 中的代码与数据时，先短暂将 WP 位置 0，待完成写操作后，将 WP 重新置 1，同时确保这整个更新操作是原子性的（不可被中断），以避免攻击者通过对该更新操作发起攻击使 WP 位永久置 0，导致 APM 失效。由于 APKM 代码与静态数据所在的内存页被设置为只读的写保护属性，任何对该页面的写操作都将引发页错误中断，因此，将 APKM 的原子更新操作写在页错误中断函数内，同时将该函数所在内存页也设置写保护属性。

根据上述解决思路，提出 APKM 初始化方法，实现部署 APM 的第一步，一旦完成 APKM 的初始化后，所有对 APKM 的操作都将被纳入到内存锁机制的保护范畴，攻击者无法通过攻击 APKM 破坏 APM 机制，因而解决了 APKM 的完整性保护问题。为更好地描述 APKM 完整性保护方法，使用符号表达其中涉及的操作，如表 1、算法 1 和算法 2 所示。

表 1 符号释义 1

符号	描述
M(APKM)	APKM 被载入到 IaaS 节点内核后所在的内存页面
PT(APKM)	APKM 所在内存页面对应的页表项所在的页表页面
M(Update)	APKM 更新关键数据时其执行函数所在的内存页面
LOCK[M]	设置 M 页面为只读的写保护属性
WP=1	IaaS 节点控制寄存器 WP 写保护位被打开（被置位）

算法 1 APKM 的初始化

1) 加载 APKM 模块到系统内核空间
2) LOCK[PT(APKM)]
3) LOCK[M(APKM)]
4) LOCK[M(Update)]
5) 置 WP 位为 1
6) 启动 APKM 模块使其正常工作

算法 2 APKM 的原子更新

1) 发送更新 APKM 模块中的数据的命令
2) 触发页错误处理函数
3) 调用更新函数 M(Update)
4) 置 WP 位为 0
5) 将数据写入 APKM 模块
6) 置 WP 位为 1，重启 APKM 模块

算法 2 的步骤 4)~步骤 6)是对 APKM 的写保护操作，该过程被写入到修改 APKM 内存页时会触发的页错误中断函数中去，并被设置为不可中断，以此防止任何针对该更新操作的攻击。

3.3 Agent 完整性与命令正确性验证方法

TTP 与 IPP 在 IaaS 节点上联合部署 APM 时，只要完成了 APKM 的正确初始化且保证 WP 位的置 1 状态，APKM 的代码与静态数据就无法被随意修改，它自身的完整性得到了保护，因此，以 APKM 为信任基础，对 agent 实施保护是可行的。TCG 组织[5]在可信规范中通过实体的完整性判断其可信性，但采用完整性度量仅能保证 agent 的完整性不被篡改，无法阻止 agent 执行错误命令，为解决这个问题，本文还采用执行命令的正确性验证方法校验 agent 行为的正确性，只有正确的命令将被允许执行，错误的命令将被拒绝。由于 agent 一般由 TTP 与 IPP 协商设计或选择，很容易获得它们在可信环境中启动与运行时的完整性度量值和命令正确执行时的上下文数据，这些将被事先计算获得并作为静态数据保存在 APKM 中，作为之后验证时调用的基准值。

对 agent 的完整性验证采用 APKM 主动发起周期性度量和响应 TTP 与 IPP 的完整性度量请求 2 种方式，将计算获得的完整性度量值与事先保存的完整性基准值作比对，相同则表示 agent 未受到破坏，不同则表示 agent 已遭到篡改，记录该异常信息到 agent 的可信性状态报告中，并及时通知 TTP 与 IPP 采取进一步应对措施；对 agent 的命令正确性验证采用检查 agent 执行命令时上下文数据（包括寄存器值、调用该命令的函数地址和该命令调用的函数地址等）的正确与否来完成，首先，提取 agent 在可信环境执行某一命令时的上下文数据；然后，当 agent 在 IaaS 节点上执行该命令时，读取运行时 IaaS 节点的该类数据，并与事先保存的基准值作比对，相同则表示该命令正确，允许执行，不同则表示该次执行将被拒绝，这就确保了 agent 无法执行除预定义正确命令之外的非法行为，保证了它运行时的正确性。

APKM 验证了 agent 的完整性和命令正确性后，将验证结果记录到 agent 可信性状态报告中并汇报给 TTP 与 IPP，使二者可以及时了解 agent 的可信性状态。agent 可信性报告中包含 APKM 对 agent 的验证信息，包括验证类型（完整性或命令正确性验证）、验证发起者（APKM 或相应 TTP 与 IPP 的验证请求）、验证起止时刻、验证对象信息、验证结果等。

根据以上解决思路，提出 agent 完整性和命令正确性验证方法，防止遭到破坏的 agent 生成虚假测试数据混淆 TTP 对 IaaS 节点的测试结果，攻击者也无法通过操纵 agent 代码执行非法命令来为自己谋得好处，因而实现了基于 APKM 对 agent 的保护。为更好地描述 agent 完整性和命令正确性验证方法，使用符号表达其中涉及的操作，如表 2、算法 3 和算法 4 所示。

表 2　　符号释义 2

符号	描述
agent	agent 可执行程序的关键代码
StaticM-base	在可信环境中度量 agent 启动时的完整性基准值
DynamicM-base	在可信环境中度量 agent 运行时的完整性基准值
Env-base	在可信环境中 agent 正确执行所有命令时的环境数据基准值集合
Tlog	agent 可信性状态报告

算法 3　Agent 的完整性验证

1) 从 TTP 上下载 agent 程序
2) 计算 agent 静态度量值 StaticM(agent)
3) if (StaticM(agent)==StaticM-base)
　　agent 完好，跳转步骤 4)
　else
　　agent 被破坏，跳转步骤 9)
4) 安装 agent 到节点
5) 启动 agent 并运行
6) 经过时间周期 T 或收到来自 TTP/IIP 的度量请求后，对 agent 进行动态度量
7) 计算 agent 运行时度量值 DynamicM(agent)
8) if (DynamicM(agent)=Dynamic-base)
　　agent 完好，跳转步骤 5)
　else
　　agent 被破坏，跳转步骤 9)
9) 记录 agent 异常状态到 Tlog 中，警示 TTP 和 IIP

算法 4　Agent 的命令正确性验证

1) agent 处于正常运行状态
2) 发送执行命令到 agent
3) 提取该时刻 agent 环境数据 Env(command)
4) if (Env(command) ==Env-base)
　　命令正常，跳转步骤 1)
　else
　　命令异常，跳转步骤 6)
5) 正常执行该命令后，跳转步骤 1)

6) 记录该异常命令到 Tlog，警示 TTP 和 IIP

本文采用 agent 完整性与命令正确性验证方法保护 agent，实际上，当对 agent 施加保护时，还可以考虑采用其他软件静态或动态度量技术[36~39]等，TTP 与 IPP 可根据具体应用场景在 APM 中添加更适合实际情况的 agent 可信性保护技术。

3.4 APM 有效性验证方法

APM 的有效性和保护 APKM 完整性的内存锁机制的正确性直接相关，只有当 IaaS 节点控制寄存器 CR0 中的 WP 置 1 时，APM 才能正确生效。在基于 TTP 的 IaaS 平台测试环境中，部署 APM 后，APKM 可以确保攻击者无法在 TTP 与 IPP 毫无察觉的情况下破坏 agent，也无法利用 agent 对 IaaS 节点造成进一步破坏，但若攻击者利用 IaaS 节点虚拟化软件栈中的其他漏洞提取到 ring0，将 WP 置 0 后破坏 APKM，APM 也无能为力。实际上，在这种攻击场景中，一旦 APM 失效，即可认为该 IaaS 节点已不再可信，因为系统中如果已经存在有非 TTP 与 IPP 的用户发送特权指令改变了 CR0 中 WP 的置位状态，那么可以推断出该系统中一定已经存在有攻击者获得了执行该敏感特权指令的权限。

根据以上描述，本文提出一种基于质询的 APM 有效性验证方法，由 TTP 与 IPP 对 APM 所在 IaaS 节点发起周期性和随机时刻的质询，以获得该节点上 CR0 中 WP 的置位状态；TTP 也分析从上个质询时刻起到当前质询时刻之间时段内 TTP 与 IPP 所发送指令对 WP 置位状态的影响，将质询结果与分析结果进行对比，若一致，则认为 APM 当前正常生效；否则，认为该节点上 WP 已发生异常反转，已存在有获得了系统控制特权的未知用户，因此，有理由认为该节点已经受损，该异常情况将被及时汇报给 TTP 与 IPP，以采取进一步应对措施。为更好地描述 APM 有效性验证方法，使用符号表达该技术中涉及的操作，如表 3 和算法 5 所示。

表 3　　符号释义 3

符号	描述
Time(current)	收到 WP 置位状态质询请求的当前时刻
Time(last)	收到 WP 置位状态质询请求的上一个时刻
WPbit	WP 置位状态的质询结果
WPanalysis	TTP 对 WP 置位状态的分析结果

算法 5　APM 的有效性验证

1) APM 机制正常运行时，在时刻 Time(current) 发起一次 WP 位状态查询

2) 读取 WPbit 的当前状态

3) TTP 分析从上次运行时刻 Time(last) 到当前时刻 Time(current)的所有命令，给出正确的 WP 置位状态分析结果 WPanalysis

4) if (WPbit=WPanalysis)

APM 运行完好，跳转步骤 1)

else

系统中存在有未知用户执行特权指令，该节点有很大概率已经被破坏，跳转步骤 5)

5) 记录该异常命令到 Tlog，警示 TTP 和 IIP

通过 TTP 和 IPP 发起的基于质询的 APM 有效性验证方法，能及时发现失效 APM 所在节点并止损，尽量减少不可信 agent 产生错误虚假数据混淆 TTP 分析判断的可能性，可有效提高 IaaS 平台测试结果的准确性。

4 安全性分析

Bell-LaPadula 模型[40]（BLP 模型）是一种计算机安全模型，形式化定义了系统、系统状态及状态间的转换规则，可以证明一个安全系统，经过一系列规则转换后，仍是安全的。本文利用 BLP 模型对 APM 机制进行建模，分析其安全性。APM 机制中涉及的元素有：主体、客体、访问属性、权限、请求、决定和系统状态，具体含义描述如下。

1) 主体集 $S=\{s_1,s_2\}$，其中，s_1 表示 TTP 或 IIP，s_2 表示潜在攻击者。

2) 客体集 $O=\{o_1,o_2\}$，其中，o_1 表示 APKM，o_2 表示用于执行具体测试任务的应用软件。

3) 访问属性集 $A=\{r,w,e\}$，其中，r 表示可读，w 表示可写，e 表示可执行。

4) 权限集 $F=\{f_1,f_2\}$，其中，f_1 是主体权限函数，$f_1(s)$ 越大，s 的访问权限越高，f_2 是客体权限函数，$f_2(o)$ 越大，o 越不容易被访问。

5) 请求元素集 $RA=\{c,d\}$，

由 $c(change, create)$，$d(delete)$ 组成，表示主体对客体可以执行的操作请求。

6) 决定集 $D=\{\text{yes,no,error}\}$，表示 APM 机制对主体对客体所发起的一个操作请求的判断结果。

7) 系统状态集 $V=\{v\,|\,v=(b,f)\}=P(S\times O\times A\times F)$，由所有状态 $v=(b,f)$ 组成，其中，$b\subset(S\times O\times A), f\in F$。

8) 状态转移关系表示为 $W\subset R\times D\times V\times V$，一

个系统 $\sum(R,D,W,v_0)$ 由状态转移关系及初始状态 v_0 决定，其中，W 表示主体对客体发起的一个操作请求被 APM 机制接受后系统安全状态的变化，v_0 表示 APM 机制中系统的初始状态，即 APKM 进行正确初始化后对 M(APKM), PT(APKM)和 M(Update)加锁，且 WP 位置 1 的状态。

根据 BLP 模型，得知 APM 机制满足以下定义。

定义 1　由状态转移关系 W 及初始状态 v_0 决定的系统 $\sum(R,D,W,v_0)$ 是指集合 $\sum(R,D,W,v_0)=\{(r,d,v)\mid\forall t\in T,(r_t,d_t,v_t,v_{t-1})\in W\}$。

定义 2　$R\times D\times V\times V$ 中的元素 (r_t,d_t,v_t,v_{t-1}) 称为系统 $\sum(R,D,W,v_0)$ 的一个动作，该动作的含义在于在系统状态 v_{t-1} 下输入请求 r_t，系统就输出决定 d_t，并将状态切换到 v_t。

定义 3　称 $(s,o,x)\in S\times O\times A$ 满足相对于 F 的安全性条件，当且仅当 $x=r$ 或 $x=w$ 且 $f_1(s)\geqslant f_2(o)$ 时，称 (s,o,x) 满足 $sc\mid F$，否则，称 (s,o,x) 不满足 $sc\mid F$。

定义 4　当状态 $v=(b,F)$ 为安全状态时，等价于所有的 $(s,o,x)\in b$ 都满足 $sc\mid F$。

根据以上定义，可证明 APM 机制中存在以下 3 个定理。

定理 1　APKM 的完整性保护。当且仅当下列条件成立时，内核态模块 APKM 的完整性得到保护，任何主体对 APKM 的操作请求都将被拒绝。条件 1) 系统正确初始化为 v_0 状态；条件 2) WP=1。

证明　由条件 1)可知，M(APKM), PT(APKM), M(Update)已加锁，系统 $\sum(R,D,W,v_0)$ 在该初始状态下是安全的。

由条件 2)可知，当 WP 置 1 时，根据 APM 机制，有 $f_1(s)\leqslant f_2(o_1)$，此时，根据定义 3，$\exists\forall(s,o_1,x)\in b$ 都不满足 $sc\mid F$，任何主体对客体 o_1 的操作请求都不满足相对于 F 的安全性条件，决策结果将是 no，因此，在以上条件下，APKM 的完整性不会被破坏，定理 1 成立，证毕。

定理 2　Agent 的完整性和命令执行正确性保护。当且仅当定理 1 成立时，agent 的完整性和命令正确性得到保护，任何主体对 agent 的操作请求都将被拒绝。

证明　在定理 1 成立时，系统 $\sum(R,D,W,v_0)$ 是安全状态，WP 置 1，由 APM 机制知 $f_1(s)\leqslant f_2(o_1)$，此时根据定义 3，$\exists\forall(s,o_1,x)\in b$ 都不满足 $sc\mid F$，任何主体对客体 o_2 的操作请求都不满足相对于 F 的安全条件，决策结果将是 no，因此，基于以上条件，agent 的完整性和命令执行正确性不会被破坏，定理 2 成立，证毕。

定理 3　APM 的有效性验证。当且仅当下列条件成立时，APM 机制是有效的。条件 1) 系统正确初始化为 v_0 状态；条件 2) WPbit=WPanalysis。

证明　由条件 1)可知，M(APKM), PT(APKM), M(Update)已加锁，系统 $\sum(R,D,W,v_0)$ 在该初始状态下是安全的。

记当前质询时刻为 t，上一个质询时刻为 $t-1$，WPbit 是 s_1 对 IaaS 节点控制寄存器 CR0 中 WP 实际置位状态的质询结果，表示为 $v_t=(b_t,F)$；WPanalysis 是 TTP 分析 s_1 操作指令后得到的 WP 安全置位状态，表示为 $\sum(R,D,v_{t^*},v_{t-1})$。当条件 2)成立时，有 $v_{t^*}=v_{t-1}$，根据定义 4 知，状态 v_t 为安全状态，所有的 $(s,o_1,x)\in b$ 都满足 $sc\mid F$，决策结果是 yes，因此，在以上条件下，APM 机制是正确生效的，即定理 3 成立，证毕。

经过以上安全性分析，证明本文提出的 APM 机制可有效保证基于 TTP 的 IaaS 平台中 agent 的完整性和命令执行的正确性，解决其完整性被破坏及代码被攻击者滥用执行非法操作的问题，达成了第 3 节提出的设计目标 1)和目标 2)；APM 机制还支持在 agent 升级或更换时基准值的动态更新，支持对多种测试软件的保护，达成了设计目标 3)；此外，提出的 APM 有效性验证方法可以及时发现 APM 失效的问题并通知 TTP 与 IPP，防止 TTP 在评估 IaaS 平台性能时受到伪造测试数据的混淆，达成了设计目标 4)。

5　实现与评价

5.1　实验环境

基于 KVM 技术模拟搭建单节点和多节点 2 类 IaaS 平台对本文方法的有效性和性能进行评价，具体实验场景中涉及的软硬件配置信息如表 4 所示。

采用 Linux 系统中流行的分析软件 Tripwire 作为 agent 对 IaaS 平台中部分文件进行完整性测试，生成的测试报告作为对应数据完整性指标的测试数据。实现一个自定义的 APKM 模块，用于验证 Tripwire 的完整性和命令执行的正确性，采用 SHA-1 算法计算 Tripwire 的散列值作为其完整性度量值，通过检查 Tripwire 运行时内存中的关键数据

表 4 软硬件配置信息

序号	类型	配置信息
S1	单节点 C_1	计算机 C_1：处理器为 Intel(R) Pentium G3250 @ 3.20 GHz，内存为 4 GB，操作系统为 OSX 10.9.5
S2	多节点 C_1, C_2, C_3	计算机 C_1：处理器为 Intel(R) Pentium G3250 @ 3.20 GHz，内存为 4 GB，操作系统为 OSX 10.9.5
		计算机 C_2：处理器为 Intel(R) Xeon(R) E5- 2620v3 @ 2.40 GHz，内存为 64 GB，操作系统为 CentOS release 6.8
		计算机 C_3：处理器为 Intel(R) Xeon(R) E5- 2603 v4 @ 1.70 GHz，内存为 64 GB，操作系统为 Ubuntu 16.04.2 LTS

验证其命令执行的正确性。为简化实验环境，在实验过程中关闭系统中的地址随机化 ASLR 功能进行分析。

5.2 实验和结果分析

本节从功能和性能这 2 个方面对 IaaS 平台上的 agent 保护机制 APM 进行测试，功能测试用于验证 APM 的有效性；性能测试用于评估 APM 引入的开销。

5.2.1 有效性实验及分析

设计 2 个对 Tripwire 软件的攻击实验验证 agent 保护机制的有效性，模拟实现 IaaS 节点控制寄存器 CR0 中 WP 位的不同攻击场景并给出 APM 有效性质询方法的表现。

1) Tripwire 篡改

攻击者通过篡改 Tripwire 的代码达到伪造度量报告的目的。本文利用软件 010Editor 修改 Tripwire 的二进制文件，将偏移地址 0x00262628-0026263C 处的警告数据“No Errors”修改为错误提示数据“22 Errors”，使 Tripwire 无法将检测结果正确记录到度量报告中，以欺骗 TTP 与 IPP。

计算正确 Tripwire 二进制文件的散列值为 orgin_hash = dd89b440818b374fb4d9d944f5adc3979e5dafc1，篡改后 Tripwire 二进制文件的散列值为 new_hash = 8e3633b36ed5072998e3912917a8a7210589f00c，二者存在明显差异，可以看出，对该程序进行手动修改实现模拟攻击后，APKM 可以通过完整性验证方法检测到该攻击。

2) Tripwire 命令异常

攻击者通过修改 Tripwire 运行时函数返回地址中的数据达到扭曲程序正常执行流程的目的。Tripwire 执行文件完整性度量命令 check 时观察发现，被调用 cIntegrityCheck 类函数 CompareFCOs 返回地址中保存的数据一直以十六进制显示为 0x3800，因此，通过修改该数据模拟攻击。

修改 Tripwire 编译文件的源代码，使之在执行 Tripwire-check 命令时能打印出 CompareFCOs 返回地址中所保存的数据。编写脚本循环执行以上命令时，修改该数据模拟攻击，可以看出攻击前后该地址的输出数据有明显改变。因此，将该地址数据作为命令执行正确性检测的环境数据之一时，可以检测到该软件命令异常执行的问题。这里对该地址数据的修改只是一个示例，实际应用时可增加环境数据的种类。

3) APM 有效性质询

APM 机制中，APKM 的完整性受到内存锁机制的保护，已有学者验证了该方案的有效性[20]，因此，本文没有讨论 APKM 遭受攻击的情形。这里考虑 APM 部署完毕后，IPP 不再向 IaaS 节点发送任何可能改变 WP 置位状态操作指令的场景。这时，只要 WP 位的状态是 1，就认为 APM 机制是正确生效的；一旦检测到 WP 位被置 0，就认为该节点上已存在有非 IPP 用户执行了 ring0 级特权指令，内存锁机制已失效，APM 机制已不再可信，需及时向 TTP 与 IPP 示警。

实验通过对一个标志位状态的变更模拟攻击者对 IaaS 节点控制寄存器 CR0 的 WP 位进行的操作，研究 APM 失效检测机制中不同质询周期 T_{inquire} 对不同攻击周期 T_{attack} 及不同攻击延续时长 T_{adelay} 的检测率，记一次攻击为标志位初始状态为 1（表示 APM 机制正确生效），经过 T_{attack} 时间后攻击者发起延续时长为 T_{adelay} 的攻击，此时，标志位被置 0（表示 APM 机制处于失效状态），攻击结束后，该标志位被再次置 1。在该攻击发生时间内，TTP 以 T_{inquire} 为周期查询该标志位的置位状态，以检测是否发生了攻击。攻击检测率是成功检测到的攻击次数对攻击总次数的百分比值。具体 APM 失效检测结果如表 5 所示，时间的单位是 s。

分析可知，基于周期性质询的 APM 失效检测方法的有效性与 T_{attack}、T_{adelay} 及 T_{inquire} 密切相关。以上检测中存在发生了攻击但未被检测到的情况，这是由于攻击延续时长较短，整个攻击流程发生在 2 次质询之间，因此，无法发现该攻击。基于随机

表 5　　APM 失效检测结果

攻击间隔时间 T_{attack}/s	攻击延续时间 T_{adelay}/s	攻击时间的增长步长	攻击次数	总时长/s	$T_{inquire}$ 质询周期	质询总次数	攻击总次数	成功检测到的攻击次数	检测率
5	200	1	200	21 097.912 109	2	9 943	200	200	100%
5	200	1	200	21 100.072 266	4	5 225	200	200	100%
5	200	1	200	21 100.044 922	6	3 483	200	193	96.5%
5	200	2	100	10 500.020 508	2	5 149	100	100	100%
5	200	2	100	10 500.018 555	4	2 850	100	99	99%
5	200	2	100	10 500.019 531	6	1 717	100	98	98%

时刻的质询可在一定程度上提高该方法的检测率，它可用于对周期性质询结果的补充。

5.2.2　性能实验及分析

本节通过计算开启 APM 机制后执行 Tripwire 命令增加的时延分析 APM 引入的性能开销。完整性度量时延是指 APKM 对 agent 进行完整性度量时引入的时间开销，命令执行时延是指 APKM 检验 Tripwire 函数返回地址中数据的正确性时引入的时间开销。

1) 完整性度量时延

取 5 个 Tripwire 命令包括策略更新（twadmin）、数据库初始化（init）、完整性检查（check）、版本查询（version）和主程序测试（test）进行实验，为提高时间测量结果的精确性和科学性，每个命令重复执行 1 000 次，取 1 000 次执行结果的平均值，单位为 μs。计算开启 APM 机制前后的完整性度量时延比值，定义该比值为 $1-\frac{\text{原始执行时间}}{\text{开启APM后命令执行时间}}$，具体实验数据记录在表 6 中，时间单位为 μs。

将表 6 中 3 个节点对 5 个测试命令的时延开销数值表示为图 2。

从表 6 和图 2 中可以看出，开启 APM 后命令执行时间比原始执行时间有所增加，其完整性度量时延最大增幅为 1.613 5‰（在 C_1 上执行命令 test），最小增幅为 0.344%（在 C_3 上执行命令 twadmin）。分析 APM 机制可知，在命令执行过程中增加时延的主要操作在于计算 Tripwire 二进制文件的完整性度量值及与 APKM 中预保存基准值的对比。考虑到开启 APM 机制前后命令执行时间的差值很小，在本文实验环境中最大相差时间在 100 μs 以下，因此，对用户实际使用造成的影响是很微小的。

表 6　　完整性度量时延

场景	命令	原始执行时间/μs	开启 APM 执行时间/μs	时延开销/μs	时延比值
节点 C_1	twadmin	993 633.602	993 672.604	39.002	0.39
	init	993 243.960	993 281.249	37.289	0.38
	check	998 251.158	998 320.632	69.474	0.70
	version	5 910.368	5 991.211	80.843	134.94
	test	5 807.799	5 903.042	95.243	161.35
节点 C_2	twadmin	106 269.497	106 294.458	24.961	2.35
	init	106 548.015	106 572.666	24.651	2.31
	check	995 196.121	995 226.251	30.130	0.30
	version	33 205.232	33 243.813	38.581	11.61
	test	6 436.648	6 474.78	38.132	58.89
节点 C_3	twadmin	989 539.064	989 373.686	34.622	0.35
	init	990 192.956	990 239.783	46.827	0.47
	check	8 989 757.590	8 989 788.516	30.926	0.034 4
	version	39 730.975	39 748.145	17.170	4.32
	test	2 896.657	2 925.743	29.086	99.41

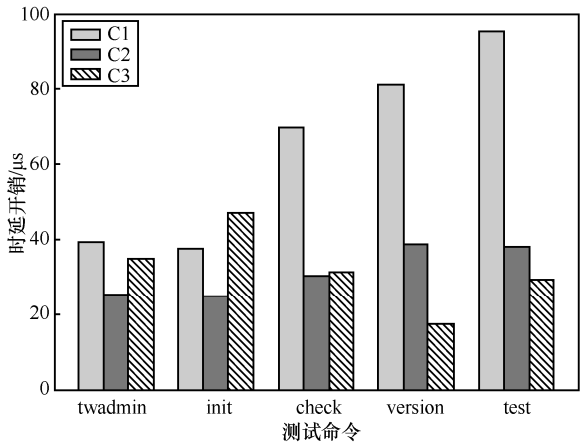


图 2 完整性度量时延开销

2) 命令执行时延

取 Tripwire 执行文件完整性度量 check 命令时调用的 cIntegrityCheck 类中函数 Execute、CompareFCOs、ProcessDir 和 ProcessChangedFCO 进行实验，分别将之记为 $func_1$、$func_2$、$func_3$ 和 $func_4$，观察发现，$func_1$ 与 $func_2$ 返回地址中保存有固定数据，而 $func_3$ 与 $func_4$ 返回地址中保存有相等数据，根据该规律，讨论执行一个命令时检查不同数目函数引入的时延开销。为提高时间测量结果的精确性和科学性，每个命令重复执行 1 000 次，取 1 000 次执行结果的平均值，单位为 μs。具体实验数据记录在表 7 中，并用图 3 表示该数据。

从表 7 和图 3 可以看出，开启 APM 后执行命令时间有所增加。分析可知，增加耗时的主要操作为 APKM 读取 Tripwire 运行时内存数据及与其预存正确数值的比对工作。虽然开启 APM 后对命令的执行引入了一定时延，且该时延随 APM 所检查 Tripwire 命令中函数数目的增加而增加，但该开销在 μs 级，而命令执行时间在 s 级，考虑到对 IaaS 平台测试环境中 agent 安全性的增加，该时间开销在可接受范围内。在实际部署 agent 时，需精心选择命令中需接受检查的函数，以在 agent 的安全性与性能之间取得平衡。

表 7 命令执行时延

场景	被检查函数数量	原始执行时间/μs	引入 APM 后的时延开销/μs
节点 C_1	1	999 441	8.541
	2	999 027	9.295
	3	998 277	9.688
	5	998 297	10.333
	8	999 583	12.865
	12	998 499	15.504
	17	999 888	17.571
节点 C_2	1	998 798	9.544
	2	999 679	12.243
	3	999 731	14.607
	5	998 415	19.906
	8	999 899	27.349
	12	999 403	36.758
	17	999 022	50.027
节点 C_3	1	8981 673	15.070
	2	9 021 959	15.860
	3	9 018 784	18.709
	5	8 981 750	19.950
	8	9010 309	26.639
	12	9 017 070	33.090
	17	8 962 640	40.549

6 结束语

基于 TTP 的 IaaS 平台测试主要依赖代理程序 agent 收集测试数据，但当前对 agent 保护技术的研究较少，现有虚拟化环境中的软件保护技术也无法直接应用到该场景中。针对这些问题，本文提出一种适用于 IaaS 平台的测试代理 agent 保护机制 APM，结合平台特点，首先提出实现 APM 应达到的设计目标，给出本文的技术依据；然后解决在现有条件下 TTP 与 IPP 不提供额外硬件或软件组件支持时保护 agent 需解决的问题，设计一个内核态保护模块 APKM，作为保护 IaaS 节点 agent 工作组件的宿主，实现对 agent 完整性与命令执行正确性的验证和保护，并生成 agent 可信性状态报告作为 TTP 判断 agent 可信状态的依据；最后为解决 APM 失效问题，设计一种基于质询的 APM 有效性验证方法，及时发现失效 APM 所在的 IaaS 节点以止损，尽可能减少不可信 agent 产生错误虚假数据混淆 TTP 分析判断的可能，提高 IaaS 平台测试结果的准确性。通过实验研究与分析，APM 可以有效保证 agent 的完整性和命令执行的正确性，时间开销保持在 μs 数量级，对系统的负担影响很小，与 APM 对 agent 的安全性提升而言，这个代价是可接受的。

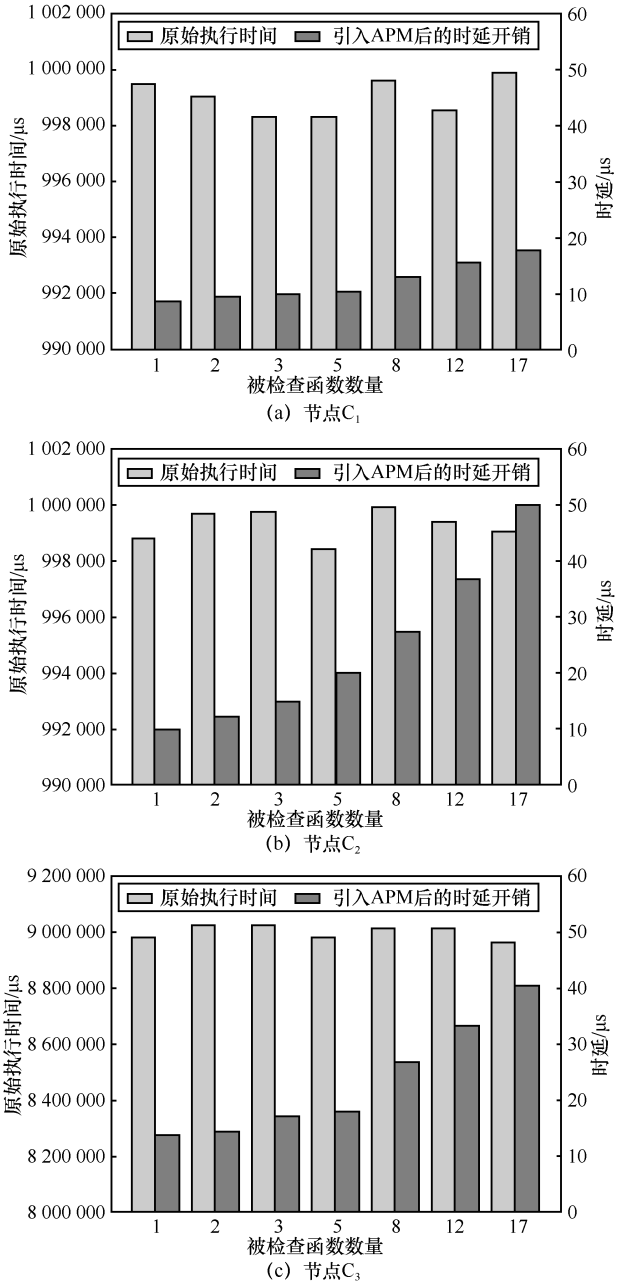


图 3　命令执行时延开销

参考文献：

[1] RIDDLE A R, CHUNG S M. A survey on the security of hypervisors in cloud computing[C]//International Conference on Distributed Computing Systems Workshops. 2015:100-104.

[2] SHAHZAD F. State-of-the-art survey on cloud computing security challenges, approaches and solutions[J]. Procedia Computer Science, 2014, 37:357-362.

[3] SARAVANAKUMAR C, ARUN C. Survey on interoperability, security, trust, privacy standardization of cloud computing[C]//International Conference on Contemporary Computing and Informatics. 2015:977-982.

[4] Common Criteria Project Sponsoring Organizations. Common criteria for information technology security evaluation: Version 2.1[S]. 2004.

[5] Trusted Computing Platform Alliance. Main specification: Version 1.1[S]. 2002.

[6] 赵波, 戴忠华, 向骕, 等. 一种云平台可信性分析模型建立方法[J]. 软件学报, 2016, 27(6):1349-1365.
ZHAO B, DAI Z H, XIANG S, et al. Model constructing method for analyzing the trusty of cloud[J]. Journal of Software, 2016, 27(6): 1349-1365.

[7] KING T M, GANTI A S. Migrating autonomic self-testing to the cloud[C]//2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW). 2010: 438-443.

[8] ZECH P. Risk-based security testing in cloud computing environments[C]//IEEE International Conference on Software Testing. IEEE Computer Society, 2011:411-414.

[9] KHAN I, REHMAN H, ZAHID A. Design and deployment of a trusted eucalyptus cloud[C]// 2011 IEEE International Conference on Cloud Computing (CLOUD). 2011: 380-387.

[10] PHAM C, CHEN D, KALBARCZYK Z, et al. CloudVal: a framework for validation of virtualization environment in cloud Infrastructure[C]// International Conference on Dependable Systems & Networks. 2011:189-196.

[11] SHAIKH R, SASIKUMAR M. Trust model for measuring security strength of cloud computing service[J]. Procedia Computer Science, 2015, 45:380-389.

[12] CARBONE M, CUI W, LU L, et al. Mapping kernel objects to enable systematic integrity checking[C]// ACM Conference on Computer and Communications Security. 2009:555-565.

[13] 谢亚龙, 丁丽萍, 林渝淇, 等. ICFF: 一种 IaaS 模式下的云取证框架[J]. 通信学报, 2013, 34(5):200-206.
XIE Y L, DING L P, LIN Y Q, et al. ICFF: a cloud forensics framework under the IaaS model[J]. Journal of Communications, 2013, 34(5): 200-206.

[14] PAZZAGLIA J C, LOTZ V, CERDA V C, et al. Advanced security service certificate for SOA: certified services go digital[M]. Vieweg Teubner, 2011.

[15] ARJONA M, HARHANI R, MUNOZ A. An engineering process to address security challenges in cloud computing[C]//ASE Bigdata/ Social Com/Cybersecurity Conference. 2014:1-12.

[16] JAATUN M G, MELAND P H, BERNSMED K, et al. A briefing on cloud security challenges and opportunities[R]. Cloud Security Whitepaper, 2013.

[17] MCCUNE J M, LI Y, QU N, et al. TrustVisor: efficient TCB reduction and attestation[C]//Security and Privacy. 2010:143-158.

[18] MUNOZ A, MAFIA A. Software and hardware certification techniques in a combined certification model[C]//11th International Conference on Security and Cryptography (SECRYPT). 2014: 1-6.

[19] WU L, ZHAN J, ZHAO Y, et al. A trusted evidence collection method based on the trusted third party for cloud platform[J]. International Journal of Distributed Sensor Networks, 2015, 501: 984964.

[20] ZHAI Y, CAO Q, CHASE J, et al. TapCon: practical third-party attestation for the cloud[C]//9th Workshop on Hot Topics in Cloud Computing (HotCloud 17), 2017: 1-7.

[21] HUNT T, ZHU Z, XU Y, et al. Ryoan: a distributed sandbox for untrusted computation on secret data[C]//Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2016:533-549.

[22] RILEY R, JIANG X, XU D. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing[C]//International Symposium on Recent Advances in Intrusion Detection, RAID 2008. 2008: 1-20.

[23] HUA J, SAKURAI K. Barrier: a lightweight hypervisor for protecting kernel integrity via memory isolation[C]//ACM Symposium on Applied Computing. 2012: 1470-1477.

[24] WANG Z, JIANG X. HyperSafe: a lightweight approach to provide lifetime hypervisor control-flow integrity[C]//Security and Privacy. 2010:380-395.

[25] ZHANG F, WANG J, SUN K, et al. HyperCheck: a hardware-assisted integrity monitor[J]. IEEE Transactions on Dependable and Secure Computing, 2014, 11(4): 332-344.

[26] AZAB A M, NING P, WANG Z, et al. HyperSentry: enabling stealthy in-context measurement of hypervisor integrity[C]//ACM Conference on Computer and Communications Security. 2010:38-49.

[27] LIN K J, WANG C Y. Using TPM to improve boot security at BIOS layer[C]//IEEE International Conference on Consumer Electronics. 2012:376-377.

[28] REN J, QI Y, DAI Y, et al. AppSec: a safe execution environment for security sensitive applications[C]//ACM Sigplan/Sigops International Conference on Virtual Execution Environments. 2015:187-199.

[29] ZHAI Y, YIN L, CHASE J, et al. CQSTR: securing cross-tenant applications with cloud containers[C]//ACM Symposium on Cloud Computing. 2016:223-236.

[30] HUANG Z, ZHENG T, SHI Y, et al. A dynamic detection method against ROP and JOP[C]//International Conference on Systems and Informatics. 2012:1072-1077.

[31] BRAR N S, DHINDSA K S. Study of virtual side channel attack in cloud computing a review[J]. International Journal of Engineering Development and Research, 2015, 3(3):1-6.

[32] MCCUNE J M, PARNO B J, PERRIG A, et al. Flicker: An execution infrastructure for TCB minimization[C]//ACM European Conference on Computer Systems. 2008:315-328.

[33] BAUER J, GRUHN M, FREILING F C. Lest we forget: cold-boot attacks on scrambled DDR3 memory[J]. Digital Investigation, 2016, 16:S65-S74.

[34] 刘川意, 林杰, 唐博. 面向云计算模式运行环境可信性动态验证机制[J]. 软件学报, 2014, 25(3):662-674.
LIU C Y, LIN J, TANG B. Dynamic trustworthiness verification mechanism for trusted cloud execution Environment[J]. Journal of Software, 2014, 25(3): 662-674.

[35] WANG Z, JIANG X. HyperSafe: a lightweight approach to provide lifetime hypervisor control-flow integrity[C]//IEEE Symposium on Security and Privacy. 2010:380-395.

[36] 刘贵堂, 周正, 周鲁苹. 软件行为的一种静态可信度量模型[J]. 海军航空工程学院学报, 2012, 27(4): 459-463.
LIU G T, ZHOU Z, ZHOU L P. A static trustworthy measurement model for software behaviors[J]. Journal of Naval Aeronautical and Astronautical University, 2012, 27(4): 459-463.

[37] SHI W C, ZHOU H W, Y J H, et al. DCFI-Checker: checking kernel dynamic control flow integrity with performance monitoring counter[J]. China Communications, 2014, 11(9):31-46.

[38] PENG G, PAN X, ZHANG H, et al. Dynamic trustiness authentication framework based on software's behavior integrity[C]//The International Conference for Young Computer Scientists. 2008: 2283-2288.

[39] 吴涛, 杨秋松, 贺也平. 基于邻接点的 VMM 动态完整性度量方法[J]. 通信学报, 2015, 36(9):169-180.
WU T, YANG Q S, HE Y. Method of dynamic integrity measurement for VMM based on adjacency data[J]. Journal of Communications, 2015, 36(9): 169-180.

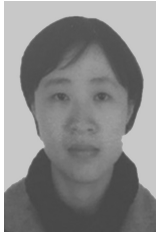
[40] TIAN-GE S I, ZHANG Y X, DAI Y Q. L-BLP security model in local area network[J]. Acta Electronica Sinica, 2007, 35(5): 1005-1008.

[作者简介]

樊佩茹（1990-），女，山西忻州人，武汉大学博士生，主要研究方向为虚拟化与云安全。

赵波（1972-），男，山东青岛人，武汉大学教授、博士生导师，主要研究方向为可信计算、虚拟化安全、嵌入式系统安全等。

倪明涛（1977-），男，湖北天门人，武汉大学博士生，主要研究方向为可信计算、物联网安全等。



陈治宏（1984-），女，重庆人，武汉大学博士生，主要研究方向为虚拟化与云存储安全。