doi:10.11959/j.issn.1000-436x.2015210

基于邻接点的 VMM 动态完整性度量方法

吴涛[1,2,3]，杨秋松[1,2]，贺也平[1,2]

(1. 中国科学院 软件研究所 基础软件国家工程研究中心，北京 100190；
2. 中国科学院 软件研究所 计算机科学国家重点实验室，北京 100190；3. 中国科学院大学，北京 100049)

摘 要：对于虚拟机监控器的动态完整性度量，由于其位于特权层，且复杂多变，一直是领域内的研究难点。提出了一种基于邻接点的动态完整性度量方法，利用邻接点作为度量模块的宿主，通过面向内存页的完整性模型和评估算法，实现了动态完整性度量。实验表明，能够准确地检测到完整性受到破坏，且仅对计算密集型任务造成适中的性能损耗。

关键词：虚拟机监控器；完整性；动态度量；邻接点；度量环

中图分类号：TP309 **文献标识码**：A

Method of dynamic integrity measurement for VMM based on adjacency data

WU Tao[1,2,3], YANG Qiu-song[1,2], HE Ye-ping[1,2]

(1. NFS, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China;
2. State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China;
3. University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Due to its high privilege and complicated runtime memory, dynamic integrity measurement for VMM (virtual machine monitor) was always a great difficulty in the current study. An innovative method based on the adjacency data was proposed, which used a neighbor as the host of a measurement module. According to an integrity model in memory page granularity and a new improved measurement algorithm, dynamic integrity measurement for VMM was implemented. Experimental data shows it could detect the integrity broken accurately, only causing a moderate performance loss for computing intensive tasks.

Key words: VMM; integrity; dynamic measurement; adjacency data; ring of measurement

1 引言

云计算为用户带来巨大便利的同时，也引入了一定的风险。不同用户不同安全级别的虚拟机运行于同一物理主机上，一旦 VMM 受到攻击，则该节点上所有的虚拟机都将面临严重威胁。在这种背景下，如何对 VMM 进行动态完整性度量进而及时发现其受到攻击，成为近年来研究的热点。

但是，对于 VMM 的动态完整性度量一直是虚拟化领域的研究难点，其中主要包含 2 个原因。首先，VMM 是一个特权层，而对于特权层的完整性度量，普遍采用的方法是在其底层再增加一个新的度量层。但是，新的度量层势必会引入新的攻击面，且影响 TCB(trusted computing base)[1~3]的体积。换言之，度量层自身的完整性无法得到保障，而其一旦受到攻击，度量结果将全部失效，系统也将面临巨大威胁。其次，VMM 的动态度量不同于静态度量。在 VMM 的运行过程中，其内存状态千变万化，并不能简单地通过校验散列值来判断其完整性是否受到破坏。

为了解决上述问题，本文提出了一种基于邻接

收稿日期：2014-09-10；**修回日期**：2015-04-13
基金项目：国家科技重大专项核高基金资助项目（2012ZX01039-004）；国家自然科学青年基金资助项目（61305054）
Foundation Items: The National Science and Technology Major Project (2012ZX01039-004); The Youth Project of the National Natural Science Foundation of China (61305054)

点的 VMM 动态完整性度量方法 AIM (adjacent integrity measurement)。AIM 首先将度量集群(cluster)内的所有节点组成一个逻辑线性环，该环保证每个节点有且仅有一个度量邻接点。接下来，AIM 开创性地将度量模块置于度量目标的邻接点上，并不断通过消息发送模块将目标节点的状态信息定时同步到位于邻接点的度量模块。这样，即使目标节点的 VMM 受到攻击，也不会影响度量模块的正常工作。另外，AIM 通过使用面向内存页的完整性模型进行检测，不断收集度量目标的状态信息，并采用经过优化改良的完整性评估算法，实时地判断所度量的 VMM 完整性是否受到破坏。

使用邻接点的方法对VMM进行动态完整性度量在国内外尚属首次。经过原型系统实验评估，AIM 并没有对虚拟机产生过高的性能损耗，仅对计算密集型任务造成了适中的影响，但是仍然在合理范围之内。

2 相关工作

完整性是一种属性，是指数据、信息处理过程、计算机设备、软件以及自然人等实体或者上述实体的任意组合在特定的环境下满足预期的先验质量[4]。具体到计算机系统而言，完整性是指系统的行为应该和设计实现时预期的行为保持一致，即操作系统的代码和数据不应该被非法篡改而执行一些与操作系统本来意图相违背的行为[5]。

完整性度量可以分为静态度量和动态度量。静态度量基于 TPM (trusted platform module)芯片，采用信任链传递的方式逐级进行度量[6]，且只能在软件加载时进行度量，一旦度量失败，被度量对象则禁止加载。相比而言，动态度量要复杂得多。所谓动态度量，是指在软件加载到内存之后，仍然通过动态地度量其完整性来达到实时监控的目的。这样，可以及时准确地发现目标对象是否已经受到攻击，并采用对应的应急防护措施[7~9]。

Reiner Sailer 等早在 2004 年就提出了一种基于 TCG (trusted computing group)的 Linux 完整性度量架构[10]。该架构第一次将 TCG 可信度量的概念扩展到内存中的动态可执行区域，并且通过设计新的度量机制和完整性验证协议 ICP(integrity challenge protocol)，最终达到度量动态可执行文件的目的。但是，该方法对可执行文件的度量必须在加载之后，执行之前。严格来说，它只能称作“度量动态目标”，而不能称作“动态度量目标”。而且，它只能度量用户态的可执行文件。换言之，一旦内核自身出现问题，整个度量架构就将全部失效。

2009 年，Kil 等 [11]提出了 ReDAS(remote dynamic attestation system)，它通过对处于运行态的应用的动态属性进行验证，从而提供应用级别的动态度量功能。ReDAS 将运行态的应用完整性划分为 2 种动态属性：结构化完整性和全局数据完整性。二者能够代表一个应用在执行过程中的运行时特点，并由度量模块在状态信息中自动提取。任意动态执行对象的完整性受到破坏，都会反应在相应的属性上。因此，抽象的完整性动态度量演化到对这 2 个属性的具体验证。但是，该方法依然没有提供对系统层软件的完整性度量方案。

Zhang 等[12]将完整性验证协议 ICP 进行增强，并引入到虚拟机领域，用于对迁移目的主机 VMM 进行完整性验证。但是，该方法使用源主机作为挑战者参与验证的过程，其有效性也依赖于源主机 VMM 必须是可信的，且该验证过程也并不属于动态度量的范畴。因此，如何对集群内的所有 VMM 都进行动态的完整性监控，仍然是云计算领域内的一个未解的难点。

HyperSentry[13]是由 Azab 等在 2010 年提出的一种新度量框架，它通过引入一个与 VMM 处于同运行级的隔离组件对 VMM 进行动态完整性度量。具体而言，该方法使用 IPMI(intelligent platform management interface)通信信道来触发度量机制，并通过 SMM(system management mode)来保护度量代码和关键数据。然而，该方法的主要问题是度量需要特殊的硬件接口支持，对于部分物理节点而言，由于其硬件条件的约束，可能导致该框架无法实施。而且，应该尽量避免对 SMM 的修改，以保证 SMM 自身的安全。

综合已有研究现状[14~18]，VMM 的动态完整性度量中的主要问题可以归纳如下。

1) 对于 VMM 的动态度量方法还比较少，尤其是只采用软件的方式进行度量，必须要解决“软件度量软件”的可信度问题。

2) 由于 VMM 处于高特权层，度量模块的位置选择十分重要。而且，添加度量模块之后，如何保证 VMM 与度量模块的隔离性也是一个研究难点。

3) 复杂多变的内存决定了动态度量模型的重要性。如何从 VMM 的运行状态中提取准确关键的完整性属性作为模型的重要元素，并保证度量过程的效

率，也是动态完整性度量研究中的一个关键瓶颈。

为了解决现有研究中存在的问题，本文提出一种基于邻接点的 VMM 动态完整性度量方法 AIM。AIM 采用纯软件的方式来解决 VMM 动态度量的问题， 并保证度量模块与 VMM 之间的隔离性。同时，设计新的完整性度量模型和评估算法，来保证 AIM 的有效性和可靠性。

3　基于邻接点的完整性度量

3.1　完整性度量架构

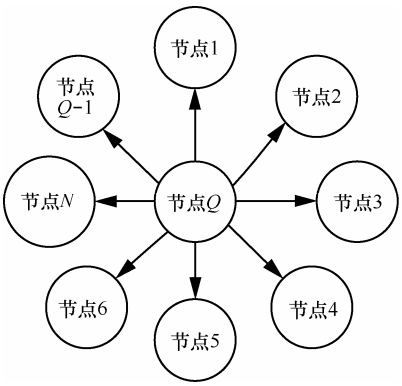
邻接点是一种抽象的逻辑关系，由度量管理节点自主定义，而与实际的网络物理拓扑结构无关。例如，可以在集群内各节点的网络拓扑结构不变的情况下，由度量管理节点根据实际情况动态调整其节点间的邻接关系。

基于邻接点的完整性度量方法的基本思路是将度量模块置于度量节点的邻接点上，而不是在度量节点 VMM 的底层或同层。这样，可以从根本上解决位于特权层的 VMM 一旦失控，造成的无法度量、无法报警的问题。但是，使用邻接点作为度量模块的宿主，从根本上是以邻接点作为完整性度量的可信基础，因此，必须保证邻接点是可信的。而邻接点的可信性则由传统的静态度量技术来保证。而且，度量模块实现于 VMM 内部，其完整性受到邻接点的动态度量，即度量模块的可信性也由邻接点保证。

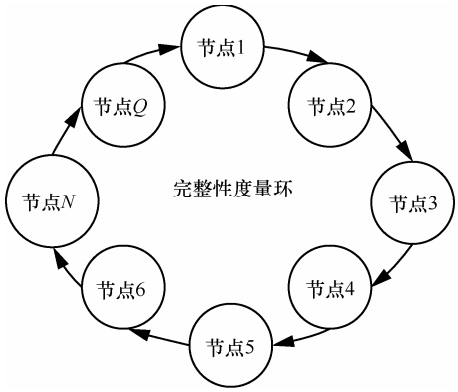
根据上述设计思路，基于邻接点的完整性度量架构可以分为 2 个层次，首先是顶层架构，集群内所有度量节点按照一对一的度量关系组成一个逻辑线性环，并保证每个度量节点有且仅有一个度量邻接点以及被度量邻接点。换言之，每个节点都有双重身份，既是度量邻接点，又是被度量邻接点；其次，具体到度量节点的自身架构而言，其位于 VMM 层次的度量模块可以度量下一个邻接点的 VMM 完整性，同时，该节点也接受上一个邻接点度量模块的完整性监控。因此，度量模块作为完整性度量架构的重要组成部分，其设计实现也是度量方案的关键。

3.1.1　完整性度量环

从整个集群而言，基于邻接点的完整性度量顶层架构可以设计为 2 种类型：一种为星状架构，如图 1(a)所示；另一种为环状架构，如图 1(b)所示。假设集群中共有 Q 个节点，N 表示其中任意某个节点，其中，$1 \leqslant N \leqslant Q$。



(a)　星状架构



(b)　环状架构

图 1　基于邻接点的完整性度量方案顶层架构

所谓星状架构是指在集群中选择一个节点作为专用邻接点，度量其他所有节点。换言之，该节点作为其他所有节点的邻接点，是完整性度量的核心，如图 1 (a)中的节点 Q。而环状架构不存在专用邻接点的概念，每个节点既是其邻接点的度量对象，同时又作为邻接点对其下一个节点进行度量，如图 1(b)中的任意节点。

对于邻接点，本文给出以下形式化定义

$$E(m,n) \Leftrightarrow SC(m,n) \,\&\, \&\, Rel(m,n)$$

其中，m，n 为任意 2 个节点的编号，$E(m,n)$表示 m 为 n 的邻接点。而 SC 函数用于判断 m 和 n 是否属于同一集群，Rel 函数则用于 m 与 n 的关系判断：在星状架构中，当且仅当 m=Q 时，邻接关系成立，即 Q 是所有节点的邻接点；在环状架构中，当且仅当 $n-m$=1 时，邻接关系成立，即 m 是 n 的邻接点，负责对 n 进行动态完整性度量。

通过对比可知，星状架构存在 2 个问题：一是所有的度量性能开销都集中于节点 Q，可能导致其负载过大，不能再对外提供服务。二是节点 Q 作为整个度量架构的核心，如果它的安全性不能保证，

那么整个度量架构就将全部失效。而环状架构则不存在上述问题，且更符合完整性度量架构的设计原则。因此，基于邻接点的完整性度量架构采用环状架构，这就是完整性度量环的由来。

但是，完整性度量环的有效性必须立足于以下假设，即在度量环中任意 2 个邻接节点的 VMM 都不可能同时受到攻击，失去控制权。可以依此推导，假设在集群中攻击任意节点并且成功的可能性相等，为 P。P_1 代表在完整性度量环中的任意一个节点受到攻击进而失控的可能性，P_2 代表在完整性度量环中的所有 2 个相邻的节点受到攻击同时失控的可能性。则 P_1，P_2 可以表示为

$$P_1 = PC_Q^1 = QP \tag{1}$$

$$P_2 = PPC_Q^1 T = QP^2 T \tag{2}$$

其中，Q 代表集群中节点的个数，T 为时间因子，即相邻的 2 个节点同时受到攻击的时间可能性。但是需要注意的是，对于 P_2 而言，并不是在集群中任选 2 个节点，而是任选 2 个相邻的节点，因此在式(2)中使用 C_Q^1，而不是 C_Q^2。在商业云计算环境中，由于采用的都是成熟的虚拟化产品，导致 P 的值往往都十分小。假设 P 的值为 10^{-3}，则 P_2 值的量级受 Q 和 T 的共同影响，而在忽略 T 的情况下就仅仅为 10^{-5} 或 10^{-6}。而且，正如上文所述，在实际的商业云中，P 的值还可能远远小于 10^{-3}，T 的值往往更小。因此，P_2 的值也就微乎其微。换言之，在完整性度量环中任意 2 个相邻的节点同时受到攻击，并且同时失控的可能性可以认为是零，即不可能出现这样的情况。在此基础上，才能保证 AIM 的有效性。

另外，集群中的每个节点在启动过程中通过静态度量的方式校验 VMM 的完整性。只有其完整性未受到破坏，才被允许加入完整性度量环，并分配一个唯一的编号。而在度量环中，所谓的邻接点是一种逻辑的邻接，而且邻接关系并不是确定的。只有当集群内的所有节点完成初始化后，度量环才构造完毕。原则上，邻接关系与节点编号 ID 相关，即编号为 N 的节点被编号为 $N-1$ 的节点度量，并对编号为 $N+1$ 的节点进行度量。最后，编号为 Q 的节点对编号 1 的节点度量，从而完成环的构造。

完整性度量环是 AIM 的顶层架构，它的形成是采用邻接点进行度量的必然结果。环状结构将各个度量任务平均分配到集群内的各个节点，降低了单个节点度量的最大性能开销。然而，环状架构也可能引入新的问题。如果一个集群内存在的物理节点过多，完整性度量环的长度过长，则可能引起信任传播的积累效应，最终导致可信性判定错误。在这种情况下，可以通过设定阈值，限定完整性度量环的最大长度，即将集群内部划分成多个完整性度量环的方法解决该问题。

3.1.2 邻接点度量架构

从顶层架构而言，基于邻接点的完整性度量架构表现为完整性度量环。但是，节点的内部架构设计依然十分重要，具体表现为度量模块的的设计实现形式。而且，从设计原则上而言，度量模块应位于 VMM 的相同级别，并且也作为 VMM 完整性度量对象的一部分，动态地接受其邻接点度量模块的完整性验证。通过这种方式，度量模块自身的安全性就可以得到保障。

图 2 所示为邻接点完整性度量的设计架构。其中，虚线箭头和尖括号表示的邻接点度量关系，实线箭头和圆括号表示与度量模块相关的数据流。

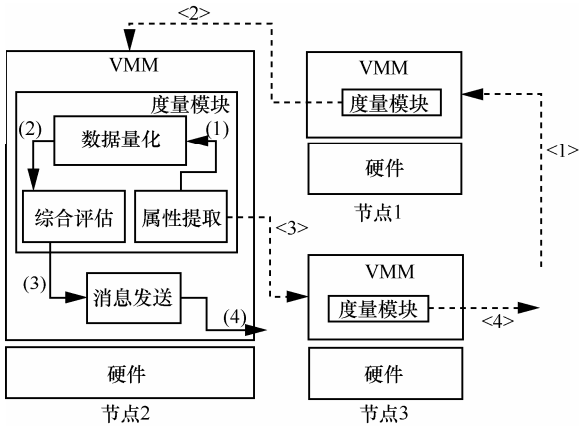


图 2 邻接点度量架构

对于节点 2 而言，其接受节点 1 的度量，并对节点 3 进行度量。同时，考虑到一旦发生 VMM 完整性受到破坏，需要发出报警信息。而且，报警信息是 VMM 动态完整性度量的最终结果，其重要性不言而喻。因此，消息发送模块被重新设计实现于 VMM 层。该模块从度量模块接收完整性评估信息，并将该信息转发到集群度量管理节点，以对该安全事件进行及时的自动或人工应急响应。

度量模块可以划分为属性提取、数据量化以及综合评估 3 个单元。从数据流来看，属性提取单元通过预定义的度量模型提取被度量 VMM 的属性信息，并以一定时间间隔动态地将这些属性收集到度量模块。需要说明的是，图 2 中<3>表示度量关系，并不是指数据流向。状态信息流入度量模块后，其传递过程在度量模块内部可以分为 2 步，一是属性提取单元将收集的信息发送到数据量化单元，二是数据量化单元再将数据处理后发送到综合评估单元。这时，如果评估结果超出设定阈值，则认为所度量的 VMM 完整性可能已经受到破坏。最后，由综合评估单元代表整个度量模块将完整性评估信息发送到消息发送模块，并由该模块将该信息发送到其他节点。

如上所述，基于邻接点的 VMM 动态完整性度量方法 AIM 的度量模块包含 3 个功能单元，各自主要功能描述如下。

1) 属性提取单元。该单元是度量模块所有度量原始数据的唯一入口。这主要是指度量提取单元根据预定义的基于邻接点的完整性度量模型对其度量的 VMM 的关键属性信息进行提取。之后，属性提取单元将提取的结果转发到数据量化单元。需要注意的是，属性提取单元对于目标 VMM 的动态属性提取是以一定时间间隔持续进行的，而该时间间隔的设置必须合理。如果过大，可能导致度量结果不及时；反之，则会导致性能开销急剧上升。

2) 数据量化单元。该单元负责将属性提取单元传递来的原始数据量化，并将量化结果传递到下一个功能单元。所谓量化是指根据度量算法，将该时间点的动态度量属性数据转化为一个 0~100 之间的值。量化过程与动态属性自身特性密切相关，也就是说，每个动态属性的量化算法并不相同。另外，从功能实现来看，完整性度量的主要工作是由数据量化单元来完成的，因此，该单元是度量模块的核心。

3) 综合评估单元。该单元通过综合评估算法，将数据量化单元产生的多个属性量化结果进行综合，并对结果进行评估，得出结论。而且，还需要为每个评估结果设定阈值，当评估结果超出阈值时，才最终判定目标 VMM 的完整性受到了破坏。这时，综合评估单元还负责将评估结果发送到消息发送模块。

在集群中，每个节点都必须采用邻接点度量架构，并通过度量模块对其下一个节点的 VMM 进行度量。所有节点通过这种邻接点度量关系最终形成一个完整性度量环。而 AIM 的架构正是由顶层的完整性度量环和底层的邻接点度量架构共同组成。

3.1.3 度量管理

在集群中必须提供一个节点作为度量管理节点，用于存储整个度量架构产生的安全关键数据。而且，管理节点不提供用户服务，对外不可见。因此，管理节点的安全性相对较高。表 1 为度量管理。

表 1　　度量管理

ID	IP 地址	静态度量	动态度量	邻接 ID	安全标签
1	10.3.3.1	1	95	*Q*	Top
2	10.3.3.2	1	80	1	Middle
3	10.3.3.3	1	90	2	Low
...	10.3.3....	1	…	…	…
Q	10.3.3.*Q*	1	85	*Q*−1	Top

根据度量架构的整体设计，度量管理表的数据项可以划分为 6 列，分别是 ID、IP 地址、静态度量、动态度量、邻接 ID 以及安全标签。

首先，ID 为节点的唯一标识，在每个集群中是不可重复的。但是 ID 并不是固定的，一旦该节点断电重启，则 ID 就会发生变化，这是为预防重放攻击而设计。而 IP 地址即为该节点 Domain 0 的网络地址。

其次，静态度量和动态度量分别表示静态度量值和动态度量值。静态度量的类型为布尔值，1 表示该节点静态度量成功，并已加入到集群中；0 则表示完整性校验不成功，初始化失败。该值默认为 0。动态度量的取值范围为 0~100。动态度量的结果为采用完整性评估算法取得的一个综合结果。同时，还必须对该表项设定报警阈值，而阈值由对应节点的综合评估单元根据最近评估经验值计算得来。动态度量值是动态变化的。

最后，邻接 ID 表示当前节点的邻接点 ID，即度量当前节点 VMM 完整性的邻接点 ID。而安全标签用于控制基于邻接点进行完整性度量的安全性，即约束邻接点的安全级别必须大于等于当前节点的安全级别。在 AIM 中，提供 3 个安全级别，分别为 Top、Middle 和 Low。

度量管理表将集群、节点及对应的邻接点以时

间、空间 2 个维度进行组织，可以极大地提高系统管理员的管理效率，增强集群的架构安全性。

3.2 完整性度量模型

基于邻接点的完整性度量模型是度量模块对 VMM 状态信息进行提取的决定性因素，它直接控制属性提取单元在何时以何种方式同步哪些 VMM 的动态属性信息。

AIM 拟采取的度量模型以代码分析为基础，并延伸作用到内存页，可以从根本上保证 VMM 的完整性状态能够反映到提取的属性信息中。一般情况下，系统代码的生命周期包括：首先由系统程序员使用高级程序语言开发系统，其存在形式为可直接阅读编辑的代码。经过编译链接之后，该段代码生成对应的二进制可执行文件。在系统启动过程中，可执行文件被加载到内存中，以内存页的形式存在。当系统需要执行这段代码时，其作为控制流的一部分被调度进 CPU，经过指令译码、解码等过程，最后执行。

基于邻接点的度量模型首先对 VMM 的代码进行分析，并按照代码的不同属性并将其划分成不同的可信区。而且，当可执行文件加载到内存之后，这些可信区也把各自的属性信息带入到内存中。因此，系统的内存代码段也被划分成不同的可信区。对应地，数据段被划分成不同的可控区。需要明确的是，基于邻接点的度量模型动态度量的对象目前只能是内存页框，并不保证度量控制流的完整性。也就是说，如果代码在控制流阶段例如在 CPU 缓存中被篡改，度量模块则不能检测。因此，AIM 的度量模型更侧重于对动态内存页的度量。

3.2.1 定义

基于邻接点的度量模型将 VMM 的代码划分为 4 种类型，分别为静态可信区 ST(static trusted area)、静态不可信区 SU(static untrusted area)、动态可控区 DC(dynamic controllable area)以及动态不可控区 DU(dynamic uncontrollable area)。同时，给出以下完整性相关的定义。

定义 1 对于任意区域的划分，其必须是连续的，且代码或内存的属性必须是确定的，其只能属于唯一一个区，即

$$\sum_{i=1}^{M} ST_i \cap \sum_{j=1}^{N} SU_j \cap \sum_{k=1}^{R} DC_k \cap \sum_{l=1}^{S} DU_l = \boldsymbol{\Phi}$$

其中，M，N，R，S 分别表示对应区域的个数，i，j，k，l 分别表示对应的索引号。并且，对于 $\forall ST_i \in \{\mathrm{ST}\}$，以及 $\forall SU_j \in \{\mathrm{SU}\}$，$\forall DC_k \in \{\mathrm{DC}\}$，$\forall DU_l \in \{\mathrm{DU}\}$ 都必须保证 ST_i、SU_j、DC_k、DU_l 是连续的。

定义 2 静态可信区 ST。对于 $\forall ST_i \in \{\mathrm{ST}\}$，在任意时间都必须保证

$$\forall_{ST_i}(Hash(ST_i) = ST_{i0}), i = 1,2,\cdots,M$$

其中，ST_{i0} 为 ST_i 的初始校验值。

也就是说，静态可信区的代码在系统执行阶段不会发生改变。除此之外，还必须对静态可信区的跳转指令(包括函数返回等所有非顺序执行序列)的跳转目标进行约束

$$\forall_{ST_i}(\forall_{\mathrm{jump}}(Target_{\mathrm{jump}}) = ST_j), i, j = 1,2,\cdots,M$$

对于任意静态可信区，要么不存在任何跳转指令，要么存在且仅存在那些跳转目标仍然是静态可信区的跳转指令。另外，静态可信区还必须保证其所有读入操作的数据来源是动态可控区，即

$$\forall_{ST_i}(\forall_{\mathrm{read}}(Source_{\mathrm{read}}) \in \{\mathrm{DC}\}),\ i = 1,2,\cdots,M$$

定义 3 静态不可信区 SU。对于 $\forall SU_j \in \{\mathrm{SU}\}$，在任意时间都必须保证

$$\forall_{SU_j}(Hash(SU_j) = SU_{j0}),\ j = 1,2,\cdots,N$$

同样，SU_{i0} 为 SU_i 的初始校验值。

静态不可信区的代码在系统执行阶段也不会发生改变，这是由其静态属性决定的。而静态不可信区与可信区的主要不同体现在对跳转指令的约束条件不同。静态不可信区的跳转约束条件为

$$\forall_{SU_j}(\exists_{\mathrm{jump}}((Target_{\mathrm{jump}}) \notin \{\mathrm{ST}\})),\ j = 1,2,\cdots,N$$

对于任意静态不可信区，其一定存在这样的跳转指令，即跳转目标不属于任何静态可信区的指令。它们的目标可能仍然是静态不可信区，也有可能是动态可控区或不可控区。另外，静态不可信区一定存在读入操作的数据来源是动态不可控区，即

$$\forall_{SU_j}(\exists_{\mathrm{read}}(Source_{\mathrm{read}}) \in \{\mathrm{DU}\}),\ j = 1,2,\cdots,N$$

静态区的划分针对于代码段，这是由于系统代码段在系统执行期间不会改变。如果 VMM 静态区的完整性被篡改，那么 VMM 将面临巨大威胁。

定义 4　动态可控区 DC。动态可控区是对系统内存数据段的划分。对于数据段而言，在 VMM 执行过程中，其内容一定是发生变化的，例如创建一个新的虚拟机，则会在内存中对应产生一个新的 Domain 结构体。因此，动态可控区并不能通过校验其完整性来判断它是否可信，必须定义新的标准和约束来判断动态可控区的可信性。

为此，对于 $\forall DC_k \in \{\mathrm{DC}\}$ 给出以下约束

$$\forall_{DC_K}(\forall_{\mathrm{change}}((Source_{\mathrm{change}}) \in \{\mathrm{ST}\})),\ k = 1,2,\cdots,R$$

对于任意动态可控区，它的数据状态变化必须能够追溯引起该内存变化的源，且必须保证该源属于静态可信区。换言之，静态可信区且只有静态可信区引入的数据变化所属的区才称为动态可控区。虽然动态可控区数据状态复杂多变，但是总体而言它依然属于可控的变化。

定义 5　动态不可控区 DU。对于动态不可控区 DU 的定义并没有严格规范的定义。原则上，对于数据段而言，除动态可控区之外的所有内存区，都属于动态不可控区。即

$$\{\mathrm{DU}\} = A - (\{\mathrm{ST}\} \cup \{\mathrm{SU}\} \cup \{\mathrm{DC}\})$$

其中，A 表示所有系统内存区的全集。由于动态不可控区复杂难控，无法判断其变化是否合法，因此在本度量模型上暂不考虑将其加入。

对静态可信区 ST，静态不可信区 SU，动态可控区 DC 以及动态不可控区 DU 定义完毕之后，还需要定义在何种条件下才能判定 VMM 的完整性仍然完整，未受破坏。

定义 6　原则上，VMM 完整性度量的结果有 4 个属性区的信息共同决定，定义其为 *PERF*。理论上，*PERF* 应为一个布尔值。要么为 1，代表完整性未受破坏；要么为 0，代表完整性受到破坏。由此，*PERF* 的理论表达式应为

$$PERF = IntCheck(\mathrm{ST}) \,\&\&\, IntCheck(\mathrm{SU}) \,\&\&\, ChangeMeasure(\mathrm{DC}) \,\&\&\, (Measure(\mathrm{DU}) \| 1)$$

其中，*IntCheck* 表示一个综合校验函数，用于判定 ST 和 SU 的完整性和跳转是否合法，该函数返回布尔值。*ChangeMeasure* 是对 DC 改变进行溯源的函数，其判断 DC 的每次修改是否合法。通过特殊的逻辑关系，DU 在这个表达式中被设置不起任何作用。

但是，VMM 的完整性度量并不是一个非真即假的结果。它往往需要经过综合的度量评价，表示一种可能性，即 *PERF* 的值并不仅仅为 0 或 1，也可能为 0.3 或 0.8。为了方便计算，在 AIM 中，设置 *PERF* 的取值范围为 0~100，并设定可能与不可能的阈值范围。这样，定义 6 中 *PERF* 的计算过程也需要对应发生改变，即不能仅仅通过简单的逻辑与、逻辑或来计算 *PERF* 的最终结果，而需要一个综合度量评价的过程，在本文中 *PERF* 的计算由完整性评估算法实现。

3.2.2　模型说明

根据上述定义，基于邻接点的完整性模型在度量过程中主要有 4 种类型的操作任务，详细的任务描述如下。

1) 静态完整性校验，即对所有静态区，包括可信区和不可信区进行完整性校验。AIM 采用基于 Merkle tree 的静态区度量树的方法进行完整性度量，如图 3 所示。由于所有可信区和非可信区的大小并不相同，在内存区到度量树叶子节点的映射采用 SHA1 算法计算。

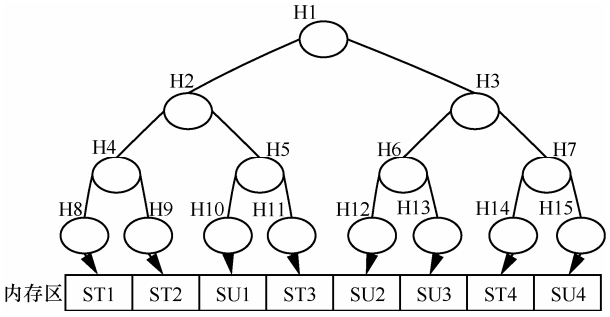


图 3　基于 Merkle tree 的静态区度量树

2) 跳转指令合法性判断，该操作只适用于静态区，必须保证可信区和不可信区的跳转符合对应的约束条件。该判断通过追踪跳转以及内存属性区的标记实现。

3) 读操作合法性判断，即对该内存区代码执行流中的读操作进行合法性判断，该操作同样只适用于静态区，以保证可信区和不可信区的读操作满足对应的约束。该判断同样通过追踪数据来源以及内存属性区的标记实现。

4) 写操作合法性判断，严格来说，该操作是判断写操作来源的合法性，只适用于动态可控区，用于保证对于动态可控区的改变只能由静态可信区发起。该操作通过记录最近的指令序列实现。

基于邻接点的完整性度量模型通过抽取静态

代码、动态内存的不同属性将其划分成不同级别的可信、可控区，同时限定各个区的动态约束条件，实现对 VMM 运行状态的动态完整性度量。

3.3 完整性综合评估

完整性模型主要定义 VMM 中需要度量的属性信息，其具体功能由度量架构中属性提取单元实现。而完整性综合评估包含 2 个过程，分别是评估过程和综合过程，它们负责将提取的信息进行评估、量化、综合，最终生成符合度量管理表中动态度量表项要求的数据。评估过程和综合过程的功能实现分别对应度量架构中的数据量化单元和综合评估单元。

3.3.1 评估过程

所谓评估过程，即为数据量化过程，是指将属性提取单元提取到的属性信息根据完整性评估算法，将各个属性进行量化，并生成对应的值。评估过程是一个动态持续的过程，每隔一定时间以数据量化单元接收到新的状态消息为触发，开始新一轮的评估。另外，在评估结束之后，数据量化单元还负责将评估结果发送到综合评估单元。

算法 1 为完整性评估算法的伪代码描述。首先，需要对消息进行切割得到与属性区对应的元数据。接下来，对本次状态同步来的所有元数据分别进行处理，这是一个循环的过程，其中 totalSize 理论上应等于 M，N，R 的和。在循环体中，不同的属性区类型有不同的处理方式。因此，必须先从元数据中提取其类型和状态信息。其中，I 和 II 分别代表静态可信区和静态不可信区，III 代表动态可控区。

算法 1 完整性评估算法

```
1)  metadata=slice(message)
2)  while index<totalSize do
3)    type=getType(metadata)
4)    state=extract(metadata)
5)    if (type==I or type==II)
6)      value=measure(state.int, state.read)
7)      value=trace(value, type, state.jump)
8)      return value
9)    else {type==III}
10)     source=trace(type, state.write)
11)     value=judge(source);
12)     return value
13)   end if
14)  end while
```

对于 I 和 II 而言，必须要度量其完整性和读操作的数据来源。其中，完整性的度量采用基于 Merkle Tree 的静态区度量树的方法进行。而且，对于二者的校验是有优先级的，完整性一旦校验失败，后续其他操作则都可以省略。因为这时就已经可以认为 VMM 不可信，并将返回值设置为 0。如果二者都校验成功，则还需要对跳转指令进行判定，并根据属性区类型给出判定结果。最后，将判定结果与返回值的当前值结合作为当前属性区的最终评估值。

对于 III 而言，由于其自身的动态属性，因此不需要对其进行完整性校验。但是，根据完整性模型定义的约束条件，必须对其写操作追溯来源，以保证来自于静态可信区。最后，最终评估值由判定函数根据追溯结果给出。

完整性评估算法是以属性区为单位给出的评估值。因此，还需要综合评估单元对所有值进行综合，以计算一个对 VMM 当前时刻进行完整性度量的最终值。

3.3.2 综合过程

综合过程是度量模块数据流的最后一个处理步骤，它的输入是多个属性区评估值，输出则是一个综合评估值。综合过程可以看作定义 6 根据实际情况进行的变形，并由以下公式计算而得。

$$D_F = \frac{\alpha}{M}\sum_{i=1}^{M} VST_i + \frac{\beta}{N}\sum_{j=1}^{N} VSU_j + \frac{\lambda}{R}\sum_{k=1}^{R} VDC_k$$

其中，D_F 表示最终的综合评估值，而 VST_i，VSU_j，VDC_k 分别表示 ST_i，SU_j，DC_k 对应属性区的评估值。α，β，λ 则分别表示 ST，SU，DC 3 种属性区对应的综合因子。对于综合因子而言，其设置既要体现 3 种属性区的重要性，又要保证最终的 D_F 值介于 0 与 100 之间。基于经验值，AIM 将 $\alpha:\beta:\lambda$ 设定为 5:3:2。

综合过程由综合评估单元完成之后，再由该单元将最终的评估值发送给本节点的消息传递模块，最后由消息传递模块将该值同步到度量管理表。但是，综合评估值作为完整性度量的结果，并不能直观地反应出度量成功与否。因此，在度量管理节点还应为该度量结果设置阈值。当评估值高于该阈值时，表示完整性度量成功；当评估值低于或等于该阈值时，表示完整性度量失败。

4　安全分析与措施

本文采用基于邻接点的 VMM 动态完整性度量方法 AIM 来监控虚拟机监控器的安全状态，进而可以及时采取安全措施保护虚拟机上的用户数据。但是，引入邻接点及度量管理体系之后，可能会引入一些其他潜在的安全威胁，包括 TOCTTOU(time of check to time of use)攻击，以及重放攻击(replay attack)。因此，还必须采取对应的安全措施对上述攻击进行防护。

4.1　TOCTTOU 攻击防护

如图 4 所示，在度量架构中，邻接点会不断将当前时刻的度量结果同步到管理节点。如果该值在阈值范围内，则不必采取任何响应措施。如果该值超出阈值，则会立即采取对应的响应措施。但是，在某些情况下，可能需要寻找一个可信的 VMM，例如虚拟机迁移中的目标节点[19~21]。如果在查询度量管理表时，确定某 VMM 为可信的，可以作为迁移的目标节点。然而，在迁移开始后，该 VMM 的完整性却突然受到破坏，那么该迁移就会引入巨大的安全隐患。这是 TOCTTOU 攻击的一种表现形式。

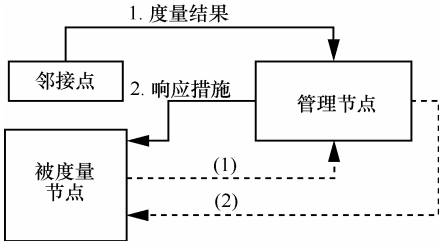


图 4　TOCTTOU 攻击防护

为了对这种攻击进行有效防护，在涉及到度量管理表的虚拟机操作中，AIM 允许被度量节点可以继续多次查询度量管理表中的数据，如图 4 中的虚线部分，以避免 TOCTTOU 这种查询与使用时数据不一致的情况出现。

4.2　重放攻击防护

一般的重放攻击是指攻击者向目的主机发送一个已经使用的分组，达到欺骗目的主机的目的。它主要应用于身份认证的过程，目的是获得一个伪造的身份，但又不被认证管理系统发现。

在本文所述的度量架构中，也可能存在这样的状况。在初始化的过程中，所有节点会发送其 ID、IP 地址以及静态度量结果到度量管理表。但是，如果有攻击者将这些数据窃取，继而利用这些数据在下次初始化时冒充原节点。那么，度量体系就将全部失效。

在这种情况下，AIM 约定在基于邻接点的度量架构中，ID 并不是静态的，而是动态的，即每次初始化时，ID 都是不同的，但必须保证 ID 在集群中是唯一的。同时，对于度量管理节点而言，还必须保证属于同一节点的诸多 ID 能够代表相同的节点信息。

5　实验及结果分析

本文实现了一个基于邻接点的 VMM 完整性度量原型，用于对 AIM 的有效性机验证以及性能测试的实验环境。同时，为了支持全虚拟化虚拟机，硬件环境的 CPU 必须支持 Intel VT 技术，这对于主流的 CPU 已经十分普遍。另外，每个节点都必须具备 TPM(trusted platform module)模块，以存储完整型散列值。

如图 5 所示，实验环境的集群包含 5 个物理节点，其中一个节点作为度量管理节点，一个节点作为存储节点，其他 3 个节点为一般的用户服务节点。其中，服务节点型号为惠普 EliteDesk 800 G1，CPU 为四核 i5 处理器，3.2 GHz 主频，并配备 4 GB 内存，500 GB 硬盘，以及支持 TPM 1.2 的硬件模块。对于软件环境，实验的 VMM 以 Xen 4.1.4[22,23]为原型进行修改，Domain 0 为 CentOS 6.4，内核版本为 2.6.32。同时，实验使用 2 种类型的全虚拟化虚拟机，分别是 Ubuntu 12.04 和 Fedora 18。它们各自配备 2 GB 内存，并分别部署不同的应用作为测试对比。最后，还需要 2 台硬件环境完全相同的主机，并安装官方标准的 Xen 4.1.4 作为性能测试的基准。另外，还需要说明的是，由于实验只是为了测试方法的有效性和性能开销，所以在测试中并没有对所有的系统代码和数据进行属性分区，只是对 Xen 常见攻击[24]涉及到的部分代码进行了划分，作为测试的代表。

在测试过程中，首先，必须验证基于邻接点的 VMM 动态完整性度量方法的有效性，证明方法是可用的。同时，设置不同的时间间隔，对比检测到 VMM 受到攻击的时间。其次，对比基于邻接点的 VMM 动态完整性度量方法在不同的测试环境下引入的性能开销。实验提供 2 种测试环境，即在 VMM

上创建虚拟机，分别提供 Web 服务，编译内核。而且，这 2 个环境需要分别在 Ubuntu 和 Fedora 虚拟机上运行测试。最后，2 种测试环境的性能需要与标准的 Xen 进行对比，因此，必须在 Xen 上运行相同的测试基准，以对比邻接点度量产生的性能开销。

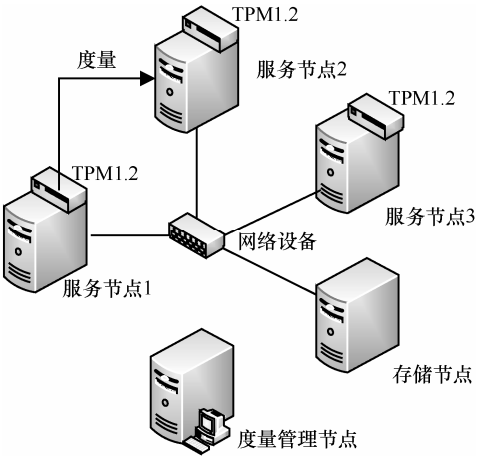


图 5 实验系统架构

5.1 有效性验证

由图 6 和图 7 所示的实验数据可知，基于邻接点的 VMM 动态完整性度量方法是有效的，其通过提取模型中定义的内存关键属性，能够检测到 VMM 的完整性受到了破坏，且有效性验证成功耗时均小于度量周期。

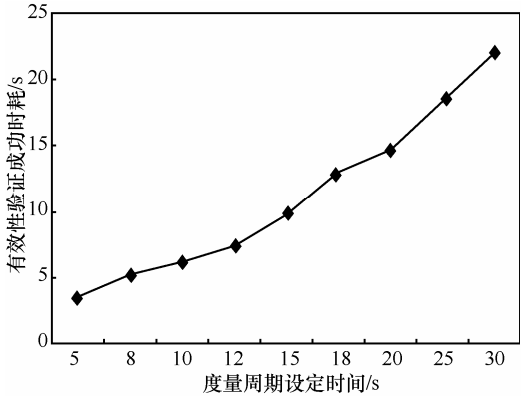


图 6 有效性验证成功耗时与度量周期的关系

但是，有效性验证成功耗时与完整性度量的周期设置有一定关系，有效性验证成功耗时会随着度量周期逐步放大，但是二者之比在 30 s 之内浮动不大，处于 62%~74%之间。通过对比可知，时间间隔在 12 s 时，该值达到最小为 62%。也就是说，这时候的度量周期设置最为合理。

在有效性验证中，只是针对性地对完整性度量模型所涉及到的内存关键属性进行了篡改。因此，实验有其局限性。如果存在某种攻击针对模型未定义的内存关键属性，则方法可能失效。后续研究将重点考虑如何精确地提取并尽可能覆盖所有的运行时内存关键属性。

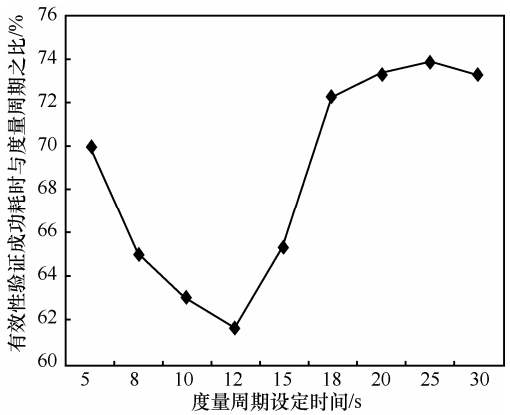


图 7 有效性验证成功耗时与度量周期之比

5.2 性能分析

图 8 所示为与标准 Xen 的性能开销对比，横坐标表示虚拟机分别在提供 Web 服务和编译内核的不同负载下进行完整性度量的性能对比，基准值为 100%。当 Web 服务器的并发数为 8 000 时，Xen 和 AIM 的原型系统网络性能差距并不大。但是，对计算密集型任务内核编译而言，却造成了适中的性能损耗，但是仍在可接受范围之内。

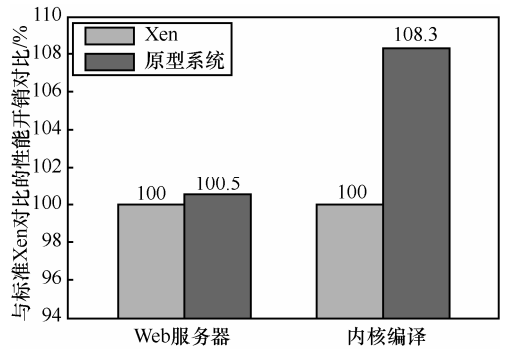


图 8 与标准 Xen 对比的性能开销
(时间间隔为 12 s，且使用 Ubuntu 虚拟机)

图 9 和图 10 则分别对比了在原型系统中使用 Fedora 和 Ubuntu2 种不同的虚拟机进行 Web 服务器和内核编译的性能开销测试结果。在图 9 中，Web 服务器的并发数分别设置为 1 000 到 8 000，可见网络性能也随之稍有下降。而在图 10 中，内核编译任务数由 1 到 5 递增时，对应的编译时间也呈线性的增长。实验结果表明，度量模块在原型系统中对

于 2 种不同的虚拟机影响接近一致，但是 Ubuntu 下性能稍微领先。

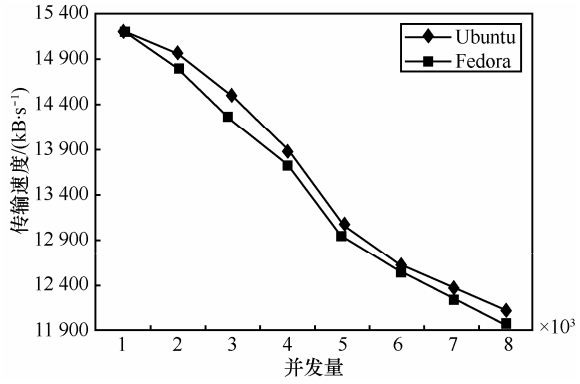


图 9　Ubuntu 和 Fedora 虚拟机作为 Web 服务器的性能开销对比

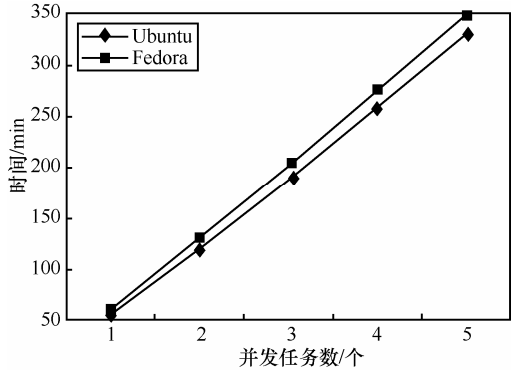


图 10　Ubuntu 和 Fedora 虚拟机进行内核编译的性能开销对比

通过实验，可以证明 AIM 的有效性，且其只对计算密集型任务造成了适中的性能影响。

6　结束语

VMM 的动态完整性度量一直以来是云计算领域的研究难点，其主要问题在于 VMM 自身处于特权层导致度量模块难以置放，以及复杂多变的内存导致难以验证。基于邻接点的 VMM 动态完整性度量方法 AIM 创新地将度量模块置于度量节点的邻接点上，并重新设计改进的更符合动态内存的度量模型，从根本上解决了目前存在的 2 个关键问题。实验结果显示，该方法能够及时地检测到 VMM 的完整性受到破坏，且仅对计算密集型任务造成了适中的性能影响，而对其他类型的任务只引起了很小的性能开销。

但是，VMM 的动态完整性度量一直是领域内的研究难点，很难找到高效准确的通用方法来解决该问题。下一步工作将继续对内存关键属性的提取进行分析和研究，并着重于提高动态完整性度量方法的实际可部署能力。

参考文献：

[1] MCCUNE J, PARNO B, PERRIG A, *et al*. Minimal TCB code execution [A]. Proc of IEEE Symposium on Security and Privacy[C]. 2007. 267-272.

[2] MCCUNE J, PARNO B, PERRIG A, *et al*. An execution infrastructure for TCB minimization[A]. Proc of Eurosys[C]. 2008.

[3] MCCUNE J, LI Y, QU N, *et al*. TrustVisor: efficient TCB reduction and attestation[A]. Proc of IEEE Symposium on Security and Privacy[C]. 2010. 143-158.

[4] SANDHU R S. On five definitions of data integrity[A]. Proc of the 7th IFIP WG 11.3 Working Conference on Database Security[C]. 1993. 257-267.

[5] Department of Defense, USA. Trusted Computer System Evaluation Criteria, TCSEC[S]. 1985.

[6] Trusted Computing Group. TPM Main Specification Level 2, Revision 116 [EB/OL]. http://www.trustedcomputinggroup.org/resources/tpm_main_specification.

[7] HOFMANN O, KIM S, DUNN A, *et al*. Inktag: secure applications on an untrusted operating system[A]. Proc of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2013[C]. 2013. 253-264.

[8] WANG Z, WU C, GRACE M, *et al*. Isolating commodity hosted hypervisors with hyperlock[A]. Proc of Eurosys[C]. 2010. 127-140.

[9] CRISWELL J, DAUTENHAHN N, ADVE V. Virtual ghost: protecting applications from hostile operating systems[A]. Proc of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014[C]. 2014. 81-96.

[10] SAILER R, ZHANG X, JAEGER T, VAN DOORN L. Design and implementation of a TCG-based integrity measurement architecture[A]. Proc of the 13th USENIX Security Symposium[C]. 2004. 16.

[11] KIL C, SEZER E, AZAB A, NING P, ZHANG X. Remote attestation to dynamic system properties: towards providing complete system integrity evidence[A]. Proc of the 39th International Conference on Dependable Systems and Networks[C]. 2009.

[12] ZHANG F, CHEN H B. Security-preserving live migration of virtual machines in the cloud[J]. Journal of Network and Systems Management, 2013, 21(4):562-587.

[13] AZAB A, NING P, WANG Z, *et al*. HyperSentry: enabling stealthy in-context measurement of hypervisor integrity[A]. Proc of the 17th Conference on Computer and Communications Security[C]. 2010. 38-49.

[14] AZAB A, NING P, SEZER E, *et al*. A hypervisor-based integrity measurement agent[A]. Proc of the Annual Computer Security Applications Conference[C]. 2009. 461-470.

[15] DAVI L, SADEGHI A, WINANDY M. Dynamic. integrity measurement and attestation: towards defense against return-oriented programming attacks[A]. Proc of the 2009 ACM Workshop on Scalable Trusted Computing[C]. 2009. 49-54.

[16] LIU Z, LEE J, ZENG J, *et al*. CPU transparent protection of OS kernel and hypervisor integrity with programmable DRAM[A]. Proc of The 40th International Symposium on Computer Architecture[C]. 2013. 392-403.

[17] SAILER R, ZHANG X, JAEGER T, *et al*. Design and implementation of a TCG-based integrity measurement architecture[A]. Proc of the 13th Usenix Security Symposium[C]. 2004.

[18] WANG Z, JIANG X X. Hypersafe: a lightweight approach to provide lifetime hypervisor control-flow integrity[A]. Proc of IEEE Symposium on Security and Privacy[C]. 2010.

[19] CLARK C, FRASER K, HAND S, *et al*. Live migration of virtual machines[A]. Proc of the 2nd Symposium on Networked Systems Design and Implementation[C]. 2005.

[20] JO C, GUSTAFSSON E, SON J, *et al*. Efficient live migration of virtual machines using shared storage[A]. Proc of the 9th Annual International Conference on Virtual Execution Environments[C]. 2013. 41-50.

[21] SONG X, SHI J C, LIU R, *et al*. Parallelizing live migration of virtual machines[A]. Proc of the 9th Annual International Conference on Virtual Execution Environments[C]. 2013. 85-96.

[22] TAKEMURA C, CRAWFORD L. The Book of Xen: A Practical Guide for the System Administrator[M]. No Starch Press, 2009.

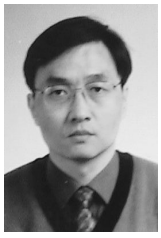
[23] Xen Project.[EB/OL] http://www.xenproject.org.

[24] WANG Z, JIANG X X, CUI W D, *et al*. Countering kernel rootkits with lightweight hook protection[A]. Proc of the 16th ACM Conference on Computer and Communications Security[C]. 2009. 545-554.

作者简介：

吴涛（1985-），男，河北保定人，中国科学院软件研究所博士生、工程师，主要研究方向为系统安全、数据安全、安全操作系统等。

杨秋松（1977-），男，河北泊头人，博士，中国科学院软件研究所正高级工程师、博士生导师，主要研究方向为软件工程、系统安全、可信计算等。



贺也平（1962-），男，甘肃兰州人，博士，中国科学院软件研究所研究员、博士生导师，主要研究方向为密码协议、安全操作系统、可信计算等。