第37卷第10期　　电　子　与　信　息　学　报　　Vol.37 No.10
2015年10月　　Journal of Electronics & Information Technology　　Oct. 2015

基于访问控制的Hypervisor非控制数据完整性保护

陈志锋* 　李清宝　张　平　曾光裕
(解放军信息工程大学　郑州　450001)
(数学工程与先进计算国家重点实验室　郑州　450001)

摘　要：在虚拟化技术广泛应用的同时虚拟层的安全问题引起了国内外研究人员的密切关注。现有的虚拟机管理器(Hypervisor)完整性保护方法主要针对代码和控制数据的完整性保护，无法抵御非控制数据攻击；采用周期性监控无法提供实时的非控制数据完整性保护。针对现有方法的不足，该文提出了基于UCON的Hypervisor非控制数据完整性保护模型UCON_{hi}。该模型在非控制数据完整性保护需求的基础上简化了UCON模型，继承了UCON模型的连续性和易变性实现非控制数据的实时访问控制。根据攻击样例分析攻击者和攻击对象确定主客体减少安全策略，提高了决策效率；并基于ECA描述UCON_{hi}安全策略，能够有效地决策非控制数据访问合法性。在Xen系统中设计并实现了相应的原型系统Xen-UCON_{hi}，通过实验评测Xen-UCON_{hi}的有效性和性能。结果表明，Xen-UCON_{hi}能够有效阻止针对虚拟机管理器的攻击，且性能开销在10%以内。

关键词：虚拟机管理器；非控制数据；使用控制；完整性保护；事件条件行为
中图分类号：TP316; TP309　　**文献标识码**：A　　**文章编号**：1009-5896(2015)10-2508-09
DOI: 10.11999/JEIT150130

Access Control Based Hypervisor Non-control Data Integrity Protection

Chen Zhi-feng　Li Qing-bao　Zhang Ping　Zeng Guang-yu
(*PLA Information Engineering University*, *Zhengzhou* 450001, *China*)
(*State Key Laboratory of Mathematical Engineering and Advanced Computing*, *Zhengzhou* 450001, *China*)

Abstract: With the widely spread of virtualization technology, the security problems of virtual layer have attracted the close attention of domestic and foreign researchers at the same time. Existing virtual machine monitor (or Hypervisor) integrity protection methods mainly focus on code and control data integrity protection, and can not resist the non-control data attacks; using periodic monitoring can not provide real-time non-control data integrity protection. According to the deficiencies of the existing methods, Hypervisor non-control data integrity protection model UCON_{hi} is proposed based on Usage CONtral (UCON). The model simplifies the UCON model according to the needs of the non-control data integrity protection, inheriting the continuity and mutability of UCON model to realize real-time access control of non-control data. The attacker and the attacked object are analyzed to determine the subjects and objects and reduce the security policies according to the attacking samples, and UCON_{hi} security policies are described based on ECA, which can effectively decide the legality of non-control data access. A prototype system Xen-UCON_{hi} is designed and implemented based on Xen system, and the effectiveness and performance overhead of Xen-UCON_{hi} are evaluated by comprehensive experiments. The results show that Xen-UCON_{hi} can effectively prevent attacks against Hypervisor with less than 10% performance overhead.

Key words: Virtual Machine Monitor (Hypervisor); Non-control data; Usage control; Integrity protection; Event condition action

1　引言

近年来，虚拟化技术在计算机系统中得到了广泛的使用，其底层性、隔离性，尤其是安全性越来越受到研究人员的关注[1–6]。针对操作系统内核的安全问题，文献[1]首次提出了虚拟机自省(Virtual Machine Introspection, VMI)的概念并将其应用于系统监控和恶意软件检测。Gibraltar[3], vDetector[4]和KCoFI[5]等利用虚拟化保护客户操作系统内核的完整性或者提供操作系统内核行为的可靠监控。同时，基于虚拟化的系统调试和分析工具如K-Tracer[2]等被用于诊断系统的异常行为和内核恶意软件的检测。

收稿日期：2015-01-27；改回日期：2015-06-23；网络出版：2015-07-27
*通信作者：陈志锋　xiaohouzi06@163.com
基金项目：核高基国家科技重大专项(2013JH00103)和国家863计划项目(2009AA01Z434)
Foundation Items: The National Science and Technology Major Project of China (2013JH00103); The National 863 Program of China (2009AA01Z434)

虚拟机系统的核心组成是 Hypervisor，或者称为虚拟机管理器(Virtual Machine Monitor, VMM)，通过虚拟化真实系统的硬件为客户虚拟机提供隔离的运行环境。由于 Hypervisor 比传统的操作系统拥有更精简的代码量，能够更好地减少软件漏洞和缺陷，因此通常被认为是安全的并用来提高运行在它之上的虚拟机的安全性。然而，即使相对于传统的操作系统而言，Hypervisor 的代码量已十分精简，但是其代码仍显得很庞大。例如，Xen[7] 4.1.2 版本含有大约 350k 行代码。最近针对 Hypervisor 漏洞进行攻击的案例[8−10]也表明了 Hypervisor 并不是绝对的安全，VM 逃亡攻击[8]以及出现的 Hypervisor Rootkits[9,10]大大增加了 Hypervisor 的安全威胁。美国国家脆弱性数据库(National Vulnerability Database, NVD)的相关研究[11]表明在最近的 3 年，Xen 系统中已发现 131 个安全漏洞，VMware ESX 有 83 个安全漏洞。因此，研究有效增强 Hypervisor 安全性的方法势在必行。

现有的增强 Hypervisor 安全性的方法主要有 3 类，一类是形式化证明方法[12−14]。该类方法基于定理证明构建 Hypervisor 形式化模型证明模型的安全属性，但该类方法受限于 Hypervisor 自身的代码量。第 2 类方法是借鉴微内核的思想减小可信计算基(Trusted Compute Base, TCB)以最小化 Hypervisor 的攻击面[15−17]。该类方法针对特定的应用或者目标构建 Hypervisor，使得 Hypervisor 的功能受限，适用范围小。第 3 类方法是通过确保运行时的 Hypervisor 完整性来提高 Hypervisor 的安全性，该类方法是目前比较流行和实用的方法[18−22]。为了确保一个应用程序运行时的安全性，可以在其底层运行一个监控软件监控它的执行。由于 Hypervisor 已经运行在软件栈的最底层，在其下构建监控软件的方法并不适用。有人曾使用嵌套的 Hypervisor 运行在需要保护的 Hypervisor 下，但这并没有解决一个本质问题：无法保证运行在 Hypervisor 下的 Hypervisor 的安全。

第 3 类方法较第 1、第 2 类方法实用有效，一方面未牺牲 Hypervisor 的任何能力，另一方面部署容易，提供了较强的安全防护能力，但其主要考虑了 Hypervisor 代码、控制数据的完整性，对于非控制数据的完整性基本未涉猎。控制数据指的是在程序运行过程中加载到程序计数器中的数据，主要包括返回地址和函数指针。通过篡改劫持控制数据，攻击者可以修改目标程序的控制流并执行攻击者想要执行的任意代码或者功能。尽管控制数据攻击被认为是最严重的威胁，但是已有研究[23]指出非控制数据攻击能够造成和控制数据攻击等价的威胁，并且很多现实世界中的应用很容易遭受这种攻击。攻击者通过破坏目标程序的非控制数据，如配置数据，用户身份数据和决策数据等，破坏应用程序的完整性。除了应用层，文献[24]首次提出了针对 Hypervisor 非控制数据的攻击，如通过修改 Xen 内存中的非控制数据实现客户虚拟机权限提升[8]、信息泄露[25]等，对 Hypervisor 的安全造成了严重威胁。目前针对 Hypervisor 非控制数据保护的研究甚少。HyperVerify[21]是第 1 个讨论 Hypervisor 非控制数据的系统，该系统通过可信 VM 监控 Hypervisor 的状态，然后对这些状态进行语义重构，从而得到 Hypervisor 的上下文信息，并通过判断这些信息的完整性来决策 Hypervisor 非控制数据的完整性。但该系统监控的 Hypervisor 非控制数据存在一定的片面性，而且其采用的周期性监控方法无法提供 Hypervisor 非控制数据的实时保护。

使用控制(Usage CONtrol, UCON)模型[26]的显著特点是使用决策的连续性与属性的易变性。连续性意味着使用控制决策不仅在访问前可以做出，在访问过程中也可以做出。易变性意味着主体、客体属性作为访问执行的结果而发生改变。这两个特性保证了 UCON 模型能够提供对数据的实时的、细粒度的、可扩展的访问控制[27]。因此，针对现有方法在 Hypervisor 非控制数据完整性保护上存在的不足，本文提出了一种基于 UCON 模型的 Hypervisor 非控制数据完整性保护方法。首先，我们根据需求简化了 UCON 模型，提出了一种适用于 Hypervisor 的访问控制模型 $UCON_{hi}$。然后，详细阐述了模型的原理和主客体的基本组成，并讨论了基于事件和行为的 $UCON_{hi}$ 策略描述方法，通过实例说明了 $UCON_{hi}$ 策略的有效性。在此基础上，设计了 $UCON_{hi}$ 的框架并实现了原型系统。最后，通过实验验证了基于 $UCON_{hi}$ 的 Hypervisor 能够有效抵抗现有的针对 Hypervisor 非控制数据的攻击，并且性能开销仅为原系统的 10%左右。

2 基于 UCON 模型的 Hypervisor 非控制数据完整性保护

2.1 $UCON_{hi}$ 模型

Hypervisor 实时完整性保护的基本需求是控制系统运行过程中非信任主体对敏感 Hypervisor 客体的访问。因此，授权需要根据主体和目标客体的实时属性做出决策。进一步地，随着安全需求等级的变化，对于主体访问请求的控制应该满足灵活性和细粒度以反应易变性：即授权可随属性的变化而动态调节。

在传统的访问控制模型中，用户标识和组员身份等是授权考虑的主要元素。然而，对于 Hypervisor 架构，VM 中活跃进程的状态、I/O 设备、Hypervisor 数据等是授权决策考虑的重要因素。例如，普通进程不能够访问 Hypervisor 的代码和数据，但是特权程序在与 Hypervisor 通信之后可改变 Hypervisor 的某些状态。

UCON 模型中授权、权利和义务是使用决策的 3 大决定因素。授权是基于主体属性和客体属性做出的；义务是主体在访问前或者访问中必须完成的动作。从上述对 Hypervisor 实时完整性保护需求可知，这一过程实现主体对客体的访问控制，并不要求主体在访问前或访问中执行动作，也不需要限制主体和客体的环境。

考虑到这些需求，本文提出了一种使用控制简化模型 $\mathrm{UCON_{hi}}$，忽略了 UCON 模型中的条件和义务，基本定义如下。

定义 1 $\mathrm{UCON_{hi}}$ 是一个六元式(S,O, ATT(S), ATT(O), R, A)。其中，S 是主体，指能够访问 Hypervisor 的对象。guestOS 中的进程、可加载内核模块(Loaded Kernel Module, LKM)等利用 API 或者漏洞，外部 I/O 设备通过 DMA 方式，Hypervisor Rootkit 利用特权等均能够访问 Hypervisor，所以主体包括 OS 进程，LKM, I/O 和 Hypervisor Rootkit。

O 是客体，指提供给主体访问的内容，有内存(Hypervisor 代码、数据、堆栈)、寄存器、Hypervisor 映像文件等。前文已指出，本文重点关注的客体是 Hypervisor 非控制数据。文献[24]归纳总结了 3 类 Hypervisor 非控制数据，分别为资源利用相关数据、安全策略相关数据和特权相关数据。在分析现有的恶意攻击样例中我们发现配置数据(如 VMCS)和决策数据对 Hypervisor 的安全性也至关重要。因此，该模型中的客体包括这 5 类对象，即关注这 5 类 Hypervisor 非控制数据。

ATT(S)是主体属性，包括主体名称、可执行代码哈希值等。

ATT(O)是客体属性，包括 Hypervisor 非控制数据的地址、类型、状态、取值等。

R 是权利，指一般的读写访问行为。

A 是授权，指在满足谓词关系为真时允许授权，在谓词关系为假时拒绝或者收回授权。

根据定义 1，授权决策是由主体属性、客体属性和请求的权利决定的，也就是说谓词是由主客体属性及请求权利组成的。例如，一个进程(主体)请求写某一个 Hypervisor 非控制数据的权利，其策略为进程的属性是被认证过的且非控制数据的类型不能够是限制集合中的一种，那么系统就对该主体的属性和客体的类型进行判断，满足策略则授予访问权，否则拒绝；同时在访问过程中，仍要进行授权合法性检测。

在 $\mathrm{UCON_{hi}}$ 模型中，对于 Hypervisor 非控制数据的使用决策不仅仅在访问之前进行评估，在访问过程中也要进行评估。如果任何一个谓词在使用过程中取值不保持为真，那么访问权限将被撤回。

同时，$\mathrm{UCON_{hi}}$ 模型中属性的更新也是伴随着整个访问过程。假设进程 p_1 已被授权访问 Hypervisor 非控制数据 VMCS(配置数据)，并在访问过程中对 VMCS 进行了修改，但未更新 VMCS，即当前状态为访问中 $\mathrm{oa}(p_1, \mathrm{VMCS}, r_1)$。此时进程 p_2 请求访问 VMCS，即 $\mathrm{ta}(p_2, \mathrm{VMCS}, r_2)$，那么根据 $\mathrm{UCON_{hi}}$ 模型，需要依据进程 p_2 请求的权利 r_2 和主客体的属性 $H(p_2) \in C \wedge \mathrm{VMCS.type} \notin \{\text{restricted-write}\}$ 进行授权，策略为：

$$\mathrm{ta}(p_2, \mathrm{VMCS}, r_2)\ \mathrm{cause}\ \mathrm{pa}(p_2, \mathrm{VMCS}, r_2)$$
$$\mathrm{if}\ H(p_2) \in C \wedge \mathrm{VMCS.type} \notin \{\text{restricted-write}\}$$

但由于进程 p_1 对 VMCS 的修改导致 VMCS 属性发生了更改，如果按照 VMCS 原来的属性进行授权决策，那么进程 p_2 能够正常访问 VMCS，这可能导致进程 p_2 非法访问 Hypervisor。所以，在主体请求访问 Hypervisor 非控制数据时需要进行属性更新操作。

由此可见，$\mathrm{UCON_{hi}}$ 模型满足 UCON 模型的决策连续性和属性易变性，可以实现 Hypervisor 非控制数据的实时监控和访问决策。

2.2 基于事件模型的策略描述

事件条件行为(Event Condition Action, ECA)[28]是一种很适合描述策略的语言，该语言描述的策略直观、具有较好的表达能力。本文借鉴 ECA，将 ECA 中的条件用谓词替换描述 $\mathrm{UCON_{hi}}$ 策略。一个 $\mathrm{UCON_{hi}}$ 策略按照下列形式声明：当一个事件发生时，如果谓词取值为真，那么事件产生的行为就必须得到执行。具体见定义 2。

定义 2 $\mathrm{UCON_{hi}}$ 策略

$$(e_1 \wedge e_2 \wedge \cdots \wedge e_i)\ \mathrm{causes}\ (\mathrm{act}_1 \wedge \mathrm{act}_2 \wedge \cdots \wedge \mathrm{act}_i)$$
$$\mathrm{if}\ (p_1 \wedge p_2 \wedge \cdots \wedge p_k)$$

其中 $e_1, e_2, \cdots, e_i$ 表示事件，$\mathrm{act}_1, \mathrm{act}_2, \cdots, \mathrm{act}_i$ 表示行为，$p_1, p_2, \cdots, p_k$ 表示谓词。

(1)事件：一个事件是由一个主体产生的，本文定义 3 类事件：请求访问 ta，访问中 oa，结束访问 ea，如图 1 所示。

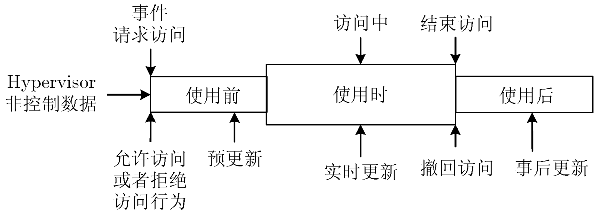

图 1 主体事件和行为分布

它们的语义描述如下：

$\mathrm{ta}(s,o,r)$：主体 s 产生一个以权利 r 访问客体 o 的需求。

$\mathrm{oa}(s,o,r)$：主体 s 正以权利 r 访问客体 o。该事件仅发生在使用过程阶段。

$\mathrm{ea}(s,o,r)$：主体 s 结束以权利 r 访问客体 o 的事件。

其中 $\mathrm{oa}(s,o,r)$ 在使用过程中可以被重复触发，或周期性触发或由某个事件或者行为触发。

(2)行为： 主体发起的访问事件产生的行为是由安全系统执行的。如图 1 所示，$\mathrm{UCON_{hi}}$ 模型中定义了 6 类行为：允许访问 pa，拒绝访问 da，撤回访问 ra，预更新 pru，实时更新 ou，事后更新 pou，语义如下：

$\mathrm{pa}(s,o,r)$： 允许一个请求授权。

$\mathrm{da}(s,o,r)$： 拒绝给一个请求授权。

$\mathrm{ra}(s,o,r)$： 撤回一个 oa 事件。

pru(attribute)：在使用前阶段更新主体或者客体的属性。

ou(attribute)：在使用时阶段更新主体或者客体的属性。

pou(attribute)：在使用后阶段更新主体或者客体的属性。

和 $\mathrm{oa}(s,o,r)$ 类似， ou(attribute) 行为可以被周期性触发或者由其它事件和行为触发。

(3)谓词： 谓词是一个由常量和变量构建的布尔表达式，其中变量和常量是由主体和客体的属性组成的。一个系统中定义有不同类型的谓词。例如，一元谓词 $p(a)$ 可由一个属性和任意数量的常量组成。二元谓词 $p(a_1,a_2)$ 由两个属性和任意数量的常量组成，两个属性可以来自单个实体或者两个不同的实体。

下面结合3个实例说明如何使用ECA语言描述 $\mathrm{UCON_{hi}}$ 策略。

例 1　假设 VM 中一个进程请求写 VMCS 中的某一项。一个合法的策略要求进程的属性是授权过的且 VMCS 某项的类型不能够是受限集合中的一种。策略声明如下：

$$\mathrm{ta}(s,o,w)\ \text{causes}\ \mathrm{pa}(s,o,\mathrm{w})$$
$$\text{if } H(s)\in C \wedge o.\text{type}\notin\{\text{restricted-write}\}$$

其中 $H(s)$ 是 s 的代码段哈希值，C 是被认证过的代码段哈希值，restricted-write 指的是禁止写操作。

例 2　假设进程 s_1 已被授以权利 r_1 访问 VMCS，此时当另外一个进程 s_2 以权利 r_2 访问进程 s_1 时， s_1 的完整性值被度量，如果进程 s_1 的完整性值发生改变，则撤回进程 s_1 访问行为被触发。策略声明如下：

$$\mathrm{oa}(s_2,s_1,r_2)\wedge\mathrm{oa}(s_1,\mathrm{VMCS},r_1)\ \text{causes}\ \mathrm{ra}(s_1,\mathrm{VMCS},r_1)$$
$$\text{if } H'(s_1)\neq H(s_1)$$

其中， $H(s_1)$ 和 $H'(s_1)$ 是 s_1 代码段在事件发生前后各自的哈希值。

例 3　假设 Hypervisor 在一次合法的升级中，修改了 VMCS 某些配置项，那么在升级之后这些项的值被更新。策略声明如下：

$$\mathrm{ta}(s,\mathrm{VMCS},\text{write})\ \text{causes}\ \mathrm{pa}(s,\mathrm{VMCS},\text{write})$$
$$\text{if } p_1\wedge p_2\wedge\cdots\wedge p_i$$
$$\mathrm{ea}(s,\mathrm{VMCS},\text{write})\ \text{causes}\ \mathrm{pou}(\mathrm{VMCS.value})\ \text{if true}$$

其中 s 是 Hypervisor 升级进程，$p_1,p_2,\cdots,p_i$ 是谓词，它们必须满足取值为真时 s 才能写 VMCS。

为了更好地说明图 1 中 $\mathrm{UCON_{hi}}$ 模型的使用，我们使用有限状态机模型来进一步论述，如图 2 所示。

从图 2 可知，该状态机模型包括一个初始状态，两个中间状态请求状态和访问状态以及 3 个终止状态结束状态、拒绝状态和撤回状态。初始状态指的是系统在等待主体请求一次客体访问。结束状态指的是访问请求成功终止。拒绝状态指的是访问请求在预授权时被拒绝。撤回状态指的是访问请求在访问过程中被撤销。请求状态表明访问请求进入预授权。访问状态表明访问请求正在进行中。

文献[24]提出了一种攻击 Hypervisor 数据实现 VM 提权的攻击，我们使用该状态机模型识别此攻击。当攻击者申请访问 Hypervisor 中控制 VM 权限的数据时，状态机模型从初始状态经请求访问 ta 事件进入请求状态，在该状态下，模型对此访问事件的合法性进行判断；由于控制 VM 权限的数据是不

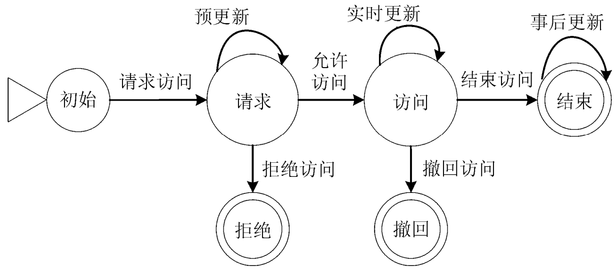

图 2　$\mathrm{UCON_{hi}}$ 模型中一次访问请求的执行过程

允许非 Hypervisor 自身代码进行修改的，故决策时产生拒绝访问 da 事件拒绝此访问。因此，状态机从请求状态经 da 事件进入拒绝状态，从而拒绝非控制数据的非法访问。对于其它类型的 Hypervisor 非控制数据访问，状态机经过不同的状态转移进入不同的终止状态。可见，基于该模型的非控制数据访问控制能够保护 Hypervisor 非控制数据的完整性。

3 架构及实现

本节首先给出了支持 $\mathrm{UCON_{hi}}$ 的实施架构，然后在架构的基础上实现原型系统。

3.1 架构

如图 3 所示，该架构包含 3 个主要的部分：策略执行点、策略决策点和属性仓库。为了提高效率，引入了访问决策缓存组件用于缓存最近的访问决策。策略执行点用于捕获主体访问 Hypervisor 非控制数据(客体)的事件，并获取主体和客体的属性，同时负责执行最后的访问控制决定。策略决策点从策略执行点处接受请求，根据策略执行点提供的主客体属性、访问请求评估用于该请求的策略，并将授权决定返回给策略执行点。为了确保策略执行点和策略决策点的代码安全，本文使用了 Hypersafe[18] 的代码和控制数据完整性保护技术确保这两部分功能代码的完整性。

在该框架下系统工作过程如下：

(1)主体产生访问客体的事件，策略执行点捕获到该事件；

(2)策略执行点到属性仓库中查找主体和客体的属性，如果还未有其对应的属性，则收集它们的属性并存入属性仓库中；否则转(3)；

(3)策略执行点询问访问决策缓存组件中是否有该主体访问客体的决策，如果有，接着检查主客体的属性是否未被修改，如若是，则按缓存的决策执行，否则，转(4)；

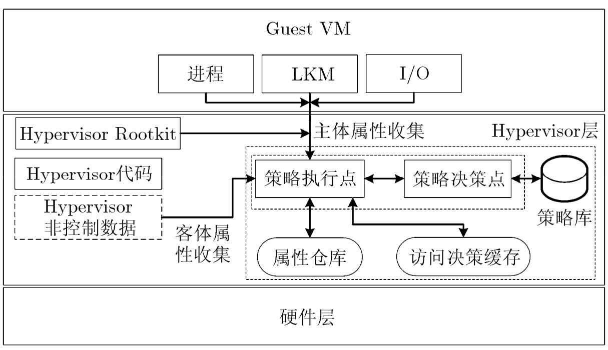

图 3 系统架构图

(4)策略执行点将主客体属性和访问请求发送到策略决策点，策略决策点根据主客体属性评估策略生成访问控制决策，然后传递给策略执行点执行。

此外，由于使用过程中主客体的属性存在变化的可能性，所以在此期间策略执行点将根据对应策略更新主客体属性。策略执行点获取到新的主客体属性，将它们存入属性仓库。同时在访问结束后，策略执行点将决策缓存到访问决策缓存组件中。

3.2 实现

我们在该架构和开源 Hypervisor Xen 的基础上实现了原型系统 Xen-$\mathrm{UCON_{hi}}$。但是该框架能够适用于不同类型的 Hypervisor，如 Bitvisor[15], KVM[29] 等。

实现该原型系统的关键在于系统能够识别主客体的类别和属性以及主体访问客体的事件。为了能够实现不同主体对这几类非控制数据访问事件的捕获，Xen-$\mathrm{UCON_{hi}}$ 首先根据主体类型采用不同的监控手段，然后根据主体访问对象类型采用不同的 $\mathrm{UCON_{hi}}$ 安全策略判断访问事件的合法性。

对于客户机进程或者 LKM，通过捕获内核级的写指令，获取主客体的属性并判断客体是否为 Hypervisor 中的对象，如果是 Hypervisor 中的对象，则将主客体属性和访问事件传递给策略执行点，否则略过。

对于 I/O 设备的 DMA 操作，Xen-$\mathrm{UCON_{hi}}$ 基于 IOMMU 的 DMA Remapping 技术获取 I/O 设备属于的保护域，然后使用保护域和 I/O 设备请求地址查看 TLB 是否存在该请求地址，若存在，则取得读写权限标记以及用于转译的目标系统地址，传递给策略执行点；若不存在，IOMMU 继续查看设备相关的 I/O 页表，提取 I/O 页表中存储的连接到目标地址的权限信息并传递给策略执行点；最后策略决策点根据安全访问控制策略决策 I/O 设备访问的合法性。

对于 Hypervisor Rootkit，通过捕获与 Hypervisor 相关的特权指令，如 vmcall, vmread, vmwrite, vmptrld, vmclear 等，发现 Hypervisor Rootkit 的创建行为，然后判断主体类型并根据策略决策访问事件。

4 实验及结果分析

本节从有效性和性能两方面对 $\mathrm{UCON_{hi}}$ 进行评测。有效性测试用于验证 $\mathrm{UCON_{hi}}$ 能否实现主体对客体的访问控制。性能测试用于验证 $\mathrm{UCON_{hi}}$ 的效率和开销。实验环境如下：主机 CPU 为 Intel(R) Core(TM) i5-750 @ 2.67 GHz，内存大小 4GB.Xen 版本为 4.4.0, Host OS 和 Guest OS 均采用的是 3.2.0-24-686 内核的 Ubuntu 12.04，各分配 2 GB 内存。

4.1 有效性测试

为了更好地测试 $\mathrm{UCON_{hi}}$ 是否能够发现并阻止攻击 Hypervisor 非控制数据的行为，我们首先选择了前文提出的 5 类 Hypervisor 非控制数据的攻击样例，它们分别修改 VMCS 配置、特权相关数据、调度数据、安全策略数据等非控制数据，其中前 3 种攻击样例来自文献[24]，后两种攻击样例为自实现。测试结果如表 1 所示。

表 1 攻击测试

攻击者	攻击的非控制数据类型	攻击方式	阻止
Attack_pd	特权数据	ROP 攻击	√
Attack_sd	调度数据	DMA 攻击	√
Attack_spd	安全策略数据	DMA 攻击	√
Attack_vcd	VMCS 配置数据	伪装合法进程	√
Attack_upd	访问控制策略数据	DMA 攻击	√

从表 1 中我们可以看出，Attack_pd 利用 ROP 攻击修改 Xen 中 domain 数据结构的 is_privileged 字段使得普通的 VM 成为具有特权的 Dom0，导致其可以创建、开启、暂停、关闭其他的 VMs。Attack_pd 修改该字段时触发 $\mathrm{UCON_{hi}}$，策略执行点捕获该事件及与事件相关的主客体属性，然后将这些信息传递给策略决策点，策略决策点经过计算判断事件的合法性并将结果反馈给策略执行点。策略描述如下所示。

ta(attack_pd,domain.is_privileged,w) causes

da(attack_pd,domain.is_privileged,w)

if $H(\text{attack\_pd}) \notin C \wedge$

domain.is_privileged.type $\in$ {restricted-write}

Attack_sd 通过修改 csched_dom 结构中的 weight 和 cap 字段使得 VM 的性能受到影响。在 Attack_sd 修改这两个字段时 $\mathrm{UCON_{hi}}$ 触发决策事件，成功阻止 Attack_sd 的篡改。策略与上一攻击类似。

Attack_spd 则是对 Xen 安全模块(Xen Security Module, XSM)中的安全策略数据进行篡改使得 VMs 之间的信息流违反 XSM 规定的安全策略。在 Attack_spd 修改 XSM 的数据时，$\mathrm{UCON_{hi}}$ 产生决策，拒绝 Attack_spd 发起的修改事件，成功阻止篡改行为。策略描述如下。

ta(attack_spd,chwall_bin_pol,w) causes

da(attack_spd,chwall_bin_pol,w)

if $H(\text{attack\_spd}) \notin C$

Attack_vcd 则是利用 Xen 与上层应用程序的通信接口实现对 VMCS 字段的篡改。在 Attack_vcd 修改 VMCS 时，$\mathrm{UCON_{hi}}$ 触发决策，判断访问主体的合法性，最终成功阻止它对 VMCS 的篡改。

Attack_upd 则是对策略库中的数据进行篡改。由于本文使用 TPM 对策略数据库进行保护，策略库的内容使用了 TPM 提供的加密功能进行加密，且任何策略库修改均需要通过认证。故 Attack_upd 无法实现对策略库的篡改。

接下来我们与 HyperVerify[20]进行对比，设定 HyperVerify 的监控周期为 15 s。首先测试两者是否能够发现现有的 Hypervisor 非控制数据攻击，然后测试两者能否发现任意时刻的 Hypervisor 非控制数据攻击，不妨设为 20 s。测试结果如表 2 所示。

从表 2 可知，在监控点 15 s 时，攻击者发动攻击，HyperVerify 不能够发现后面两种攻击，这是由于 HyperVerify 目前只针对前 3 种非控制数据进行监控；在 20 s 时，攻击者再次发动攻击，HyperVerify 无法发现任何攻击，而 Xen-$\mathrm{UCON_{hi}}$ 则能够发现所有攻击，这是由于 HyperVerify 设定的监控周期为 15 s，第 2 次触发监控检查是在 30 s, 20 s 介于两者之间，此时的 HyperVerify 并未进行任何监控操作，所以未能发现任何攻击行为，而 Xen-$\mathrm{UCON_{hi}}$ 采用的事件监控实现了实时的 HyperVisor 非控制数据访问控制监控，故 Xen-$\mathrm{UCON_{hi}}$ 发现了 20 s 的 HyperVisor 非控制数据攻击行为。因此，Xen-$\mathrm{UCON_{hi}}$ 能够实时监控非控制数据的完整性，较 HyperVerify 具有更好的保护能力。

表 2 对比测试

攻击者	攻击的非控制数据类型	HyperVerify(15 s)	Xen-$\mathrm{UCON_{hi}}$(15 s)	HyperVerify(20 s)	Xen-$\mathrm{UCON_{hi}}$(20 s)
Attack_pd	特权数据	√	√	×	√
Attack_sd	调度数据	√	√	×	√
Attack_spd	安全策略数据	√	√	×	√
Attack_vcd	VMCS 配置数据	×	√	×	√
Attack_upd	访问控制策略数据	×	√	×	√

4.2 性能测试

为了更好地说明 $UCON_{hi}$ 对系统的性能影响，实验中首先使用 LMBench 3.0-a9 工具分别对未加入 $UCON_{hi}$ 的 Xen 和融合 $UCON_{hi}$ 的 Xen-$UCON_{hi}$ 进行测试。测试结果数据均是10次重复实验的平均值，10次测试过程中未运行其余程序，测试环境一致，测试结果如图4，图5所示。

从图4可以看出，由于 Xen-$UCON_{hi}$ 未对 null call 和 null I/O 操作进行监控，直接由处理器处理，故未带来任何开销；而 open/close 为 I/O 操作指令，Xen-$UCON_{hi}$ 监控并处理了 I/O 操作，捕获这些操作的主客体属性等并决策，故这些操作需要的时间开销较大；同时，Xen-$UCON_{hi}$ 在 fork proc, exec proc 和 sh proc 3项测试指标下的性能远大于其它指标，这是因为这3项指标的测试过程中需要计算可执行文件的 hash 值。

从图5可知，Xen-$UCON_{hi}$ 对于 0k file create/delete 和 10k file create/delete 操作没有带来任何的开销，这是由于 Xen-$UCON_{hi}$ 未对文件的创建和删除进行任何额外的监控和处理；prot fault 指的是保护异常带来的开销，$UCON_{hi}$ 只是捕获该操作，捕获之后将该操作的处理返还给操作系统内核；page fault 指的是缺页异常带来的开销，由于 Xen-$UCON_{hi}$ 设置了关键数据所在页的属性，当产生写关键数据操作时，触发缺页异常，由 VMM 处理该异常，处理之后再返回内核，所以 page fault 所需要的开销较大，且大于 prot fault 的开销。

为进一步评测 $UCON_{hi}$ 的性能，我们采用了5种常用应用程序来测试其性能开销，如表3所示。

表3 基准测试应用程序

项目	版本	命令
解压缩 bzip2	1.0.6	tar zxf linux-3.2.43.tar.bz2
压缩 gzip	1.4	tar zcf linux-3.2.43
文件拷贝 cp	8.13	cp –r linux-3.2.43 otherwhere
内核编译	3.2.43	make allnoconfig && make
linux 启动	Ubuntu 12.04	Linux boot

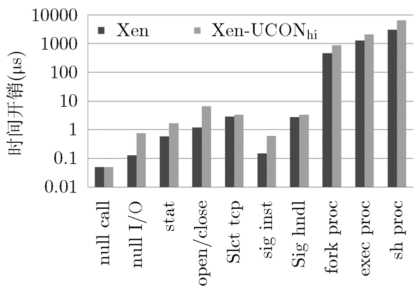

图4 LMBench 进程测试

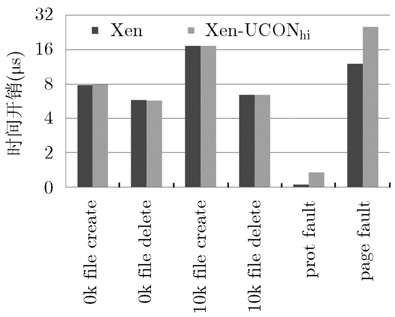

图5 LMBench 文件及 VM 系统测试

图6给出了以 Xen 为基线 Xen-$UCON_{hi}$ 的应用程序基准测试结果，从整体上看，加入 $UCON_{hi}$ 给 Xen 带来了一定的开销。对于解压缩和压缩程序，Xen-$UCON_{hi}$ 分别引入了 1.75%和 2.42%的性能开销，但这两种程序均为计算密集型作业，带来的开销相对较小。文件拷贝是 I/O 密集型作业，其中包含大量的读写操作，由于 Xen-$UCON_{hi}$ 对写操作进行了监控和处理，所以文件拷贝带来的开销相对较大。内核编译过程耗时较长，编译过程中重复加载少量应用程序和链接库，而 Xen-$UCON_{hi}$ 引入了决策缓存组件，已经过验证的程序可避免重复计算，大大减少决策时间，故内核编译带来的开销也相对较小。linux 启动指的是从上电到客户机加载完成所需要的时间，从图6可知 Xen-$UCON_{hi}$ 所需要的时间比 Xen 约多 5%。因此，从以上的测试可知，Xen-$UCON_{hi}$ 对系统引入的开销在 10%以内，这是一个可以接受的范围。与 Hyperverify 的 4%相比，尽管带来了额外 5%左右的负载，但是 Xen-$UCON_{hi}$ 实现了实时的非控制数据完整性保护，克服了 Hyperverify 存在的监控间歇问题。

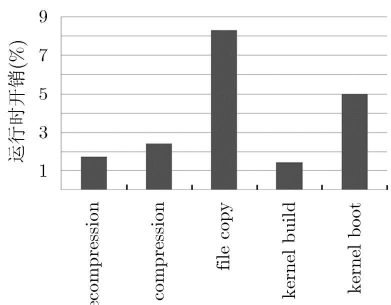

图6 基准程序测试结果

5 结论

Hypervisor 安全问题已成为国内外的研究热点。本文针对现有的 Hypervisor 完整性保护方法在非控制数据完整性保护上存在的不足提出了一种基于 UCON 的非控制数据保护模型 $\mathrm{UCON_{hi}}$，并基于 Xen 设计实现了原型系统 Xen-$\mathrm{UCON_{hi}}$。$\mathrm{UCON_{hi}}$ 具备属性易变性和决策连续性，能够保证 Hypervisor 非控制数据的实时完整性；对非控制数据的分类减少了安全策略，提高了决策效率；采用 ECA 描述的策略与系统基于事件的策略决策相吻合，易于模型和系统的实现。同时，我们对 Xen-$\mathrm{UCON_{hi}}$ 的有效性和性能进行了评测，结果表明，Xen-$\mathrm{UCON_{hi}}$在有效发现阻止 Hypervisor 非控制数据攻击的同时，引入的性能开销不超过 10%。

参 考 文 献

[1] Garfinkel T and Rosenblum M. A virtual machine introspection based architecture for intrusion detection[C]. Proceedings of the 10th Network and Distributed System Symposium, San Diego, USA, 2003: 191–206.

[2] Lanzi A, Sharif M I, and Lee W. K-Tracer: a system for extracting kernel malware behavior[C]. Proceedings of the 16th Network and Distributed System Security Symposium, San Diego, USA, 2009: 191–203.

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

[4] 李博, 沃天宇, 胡春明, 等. 基于 VMM 的操作系统隐藏对象关联检测技术[J]. 软件学报, 2013, 24(2): 405–420.
Li Bo, Wo Tian-yu, Hu Chun-ming, *et al.*. Hidden OS objects correlated detection technology based on VMM[J]. *Journal of Software*, 2013, 24(2): 405–420.

[5] Criswell J, Dautenhahn N, and Adve V. KCoFI: complete control-flow integrity for commodity operating system kernels[C]. Proceedings of the 35th IEEE Symposium on Security and Privacy, Oakland, 2014: 14–29.

[6] 殷波, 王颖, 邱雪松, 等. 一种面向云服务提供商的资源分配机制[J]. 电子与信息学报, 2014, 36(1): 15–21.
Yin Bo, Wang Ying, Qiu Xue-song, *et al.*. A resource provisioning mechanism for service providers in cloud[J]. *Journal of Electronics & Information Technology*, 2014, 36(1): 15–21.

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

[8] Wojtczuk R. Subverting the xen hypervisor[R]. Black Hat, USA, 2008.

[9] Rutkowska J and Tereshkin A. Bluepilling the xen hypervisor[R]. Black Hat, USA, 2008.

[10] Zovi D D. Hardware virtualization rootkits[R]. Black Hat Briefings, USA, 2006.

[11] National vulnerability database[OL]. http://nvd.nist.gov/. 2014-09-20.

[12] Klein G, Elphinstone K, Heiser G, *et al.*. SeL4: formal verification of an OS kernel[C]. Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, New York, USA, 2009: 207–220.

[13] Barthe G, Betarte G, Campo J D, *et al.*. Formally verifying isolation and availability in an idealized model of virtualization[C]. Proceedings of the 17th International Symposium on Formal Methods, Limerick, Ireland, 2011: 231–245.

[14] Baumann C, Bormer T, Blasum H, *et al.*. Proving memory separation in a microkernel by code level verification[C]. Proceedings of the 14th IEEE International Symposium on/ Object/Component/Service-OrientedReal-Time Distributed Computing Workshops, Reno, NV, USA, 2011: 25–32.

[15] Shinagawa T, Eiraku H, Tanimoto K, *et al.*. Bitvisor: a thin hypervisor for enforcing I/O device security[C]. Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, New York, USA, 2009: 121–130.

[16] Steinberg U and Kauer B. NOVA: a microhypervisor-based secure virtualization architecture[C]. Proceedings of the 5th European Conference on Computer Systems, New York, USA, 2010: 209–222.

[17] Nguyen A, Raj H, Rayanchu S, *et al.*. Delusional boot: securing hypervisors without massive re-engineering[C]. Proceedings of the 7th ACM European Conference on Computer Systems, New York, USA, 2012: 141–154.

[18] Wang Z and Jiang X. HyperSafe: a lightweight approach to provide lifetime hypervisor control-flow integrity[C]. Proceedings of the 31st IEEE Symposium on Security and Privacy, Oakland, USA, 2010: 380–395.

[19] Azab A M, Ning P, Wang Z, *et al.*. HyperSentry: enabling stealthy in-context measurement of hypervisor integrity[C]. Proceedings of the 17th ACM Conference on Computer and Communications Security, Chicago, USA, 2010: 38–49.

[20] Wang J, Stavrou A, and Ghosh A. HyperCheck: a hardware-assisted integrity monitor[J]. *IEEE Transactions on*

Dependable and Secure Computing, 2014, 11(4): 332–344.

[21] Ding B, He Y, Wu Y, *et al*.. HyperVerify: a vm-assisted architecture for monitoring hypervisor non-control data[C]. Proceedings of the IEEE 7th International Conference on Software Security and Reliability-Companion, Gaithersburg, MD, USA, 2013: 26–34.

[22] Liu Z, Lee J H, Zeng J, *et al*.. CPU transparent protection of OS kernel and hypervisor integrity with programmable DRAM[C]. Proceedings of the 40th Annual International Symposium on Computer Architecture, Tel-Aviv, Israel, 2013: 392–403.

[23] Chen S, Xu J, Sezer E C, *et al*.. Non-control-data attacks are realistic threats[C]. Proceedings of the 14th Usenix Security Symposium, Baltimore, MD, USA, 2005: 177–192.

[24] Ding B, He Y, Wu Y, *et al*.. Systemic threats to hypervisor non-control data[J]. *IET Information Security*, 2013, 7(4): 349–354.

[25] 俞能海, 郝卓, 徐甲甲, 等. 云安全研究进展综述[J]. 电子学报, 2013, 41(2): 371–381.
Yu Neng-hai, Hao Zhuo, Xu Jia-jia, *et al*.. Review of cloud computing security[J]. *Acta Electronica Sinica*, 2013, 41(2): 371–381.

[26] Park J and Sandhu R. Towards usage control models: beyond traditional access control[C]. Proceedings of the 7th ACM Symposium on Access Control Models and Technologies, New York, NY, USA, 2002: 57–64.

[27] 熊厚仁, 陈性元, 张斌, 等. 基于双层角色和组织的可扩展访问控制模型[J]. 电子与信息学报, 2015, DOI: 10.11999/JEIT141255.
Xiong Hou-ren, Chen Xing-yuan, Zhang Bin, *et al*.. Scalable access control model based on double-tier role and organization[J]. *Journal of Electronics & Information Technology*, 2015, DOI: 10.11999/JEIT141255.

[28] Alferes J J, Banti F, and Brogi A. An event-condition-action logic programming language[C]. Proceedings of the 10th European Conference on JELIA, Liverpool, 2006: 29–42.

[29] Kivity A, Kamay Y, Laor D, *et al*.. KVM: the linux virtual machine monitor[C]. Proceedings of the 2007 Linux Symposium, Ottawa, Canada, 2007: 225–230.

陈志锋： 男，1986 年生，博士生，研究方向为信息安全与可信计算.

李清宝： 男，1967 年生，教授，研究方向为信息安全与可信计算.

张　平： 女，1969 年生，副教授，研究方向为并行识别、信息安全.

曾光裕： 女，1966 年生，副教授，研究方向为信息安全与可信计算.