第36卷第Z1期 通 信 学 报 Vol.36 No.Z1
2015年11月 Journal on Communications November 2015
doi:10.11959/j.issn.1000-436x.2015289

内核完整性保护模型的设计与实现

田东海[1,2]，陈君华[2]，贾晓启[3]，胡昌振[1]

（1. 北京理工大学 北京市软件安全工程技术重点实验室，北京 100081；
2. 云南民族大学 云南省高校物联网应用技术重点实验室，云南 昆明 650500；
3. 中国科学院 信息工程研究所 信息安全国家重点实验室，北京 100093）

摘　要：非可信内核扩展模块是对操作系统内核完整性安全的重要威胁之一，因为它们一旦被加载到内核空间，将可能任意破坏操作系统内核数据和代码完整性。针对这一问题，提出了一种基于强制访问控制对操作系统内核完整性保护的模型—MOKIP。该模型的基本思想是为内核空间中的不同实体设置不同的完整性标签，然后保证具有低完整性标签的实体不能破坏具有高完整性标签的实体。基于硬件辅助的虚拟化技术实现了原型系统，实验结果表明，本系统能够抵御各种恶意内核扩展模块的攻击，其性能开销被控制在13%以内。

关键词：内核扩展模块；操作系统内核；完整性保护；虚拟化技术

中图分类号：TP309　　　　**文献标识码**：A

Design and implementation of a model for OS kernel integrity protection

TIAN Dong-hai[1, 2], CHEN Jun-hua[2], JIA Xiao-qi[3], HU Chang-zhen[1]

(1. Beijing Key Laboratory of Software Security Engineering Technology, Beijing Institute of Technology, Beijing 100081 China;
2. Key Laboratory of IDT Application Technology of Universities in Yunnam Province, Yunnam Minzu University, Kunming 650500, China;
3.State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China)

Abstract: Untrusted kernel extensions were considered to be a big threat to OS kernel integrity because once they were loaded into the kernel space, then they may corrupt both the OS kernel data and code at will. To address this problem, MAC-based model named MOKIP for OS kernel integrity protection was presented. The basic idea of MOKIP was to set different integrity labels for different entities in the kernel space, and then ensure that the entities with low integrity label cannot harm the entities with high integrity label. A prototype system based on the hardware assisted virtualization technology was implemented. The experimental results show that proposed system is effective at defending against various malicious kernel extension attacks within a little performance overhead which is less than 13%.

Key words: kernel extensions; OS kernel; integrity protection; virtualization technology

1　引言

操作系统内核是操作系统最核心的部分，负责管理和维护系统的整体运行。操作系统内核的任何部分受到损害都有可能影响上层应用程序的安全。当前主流操作系统如Windows和Linux均采用宏内核(monolithic-kernel)架构，即操作系统内核和内核扩展模块都运行在同一地址空间。这种架构的优点是操作系统内核无需切换地址空间就可以直接调用内核扩展模块，其性能明显优于微内核(micro-

收稿日期：2015-10-24

基金项目：国家高技术研究发展计划（“863”计划)基金资助项目(2012AA013101)；中科院先导科技专项基金资助项目(XDA06030601)；国家自然科学基金资助项目(61100228, 61202479); 云南省高校物联网应用技术重点实验室开放基金资助项目(2015IOT03)

Foundation Items: The National High Technology Research and Development Program of China (863 Program) (2012AA013101); The Strategic Priority Research Program of the Chinese Academy of Sciences(XDA06030601); The National Natural Science Foundation of China(61100228, 61202479); Open Found of Key Laboratory of IOT Application Technology of Universities in Yunnan Province (2015IOT03)

kernel)架构。然而宏内核的缺点是外部非可信扩展模块一旦被加载到内核，将可以不受限制地访问操作系统内核的任何资源，从而给操作系统带来了极大的安全隐患。目前针对内核的常见攻击是将恶意扩展模块载入到内核，使其直接破坏操作系统内核的资源，比如通过替换系统调用表来劫持应用程序与操作系统之间的交互。

为了保护操作系统内核的安全，最有效地方法之一是对内核扩展模块的行为进行访问控制。现有许多方法都是基于这一思想，他们虽然能够在一定程度上保护内核得安全，但存在以下一些局限：1)缺乏清晰地安全模型，比如文献[1]仅仅考虑了 Biba 模型[2]中不能上写的思想；2)过多地依赖专家知识，比如文献[3]要求安全专家详细指明内核中那些客体应受到保护；3)大部分保护机制的实现都是基于模拟器(如 Bochs emulator)的方法，其性能开销非常大，比如文献[1,3]都要求截获操作系统运行的每一条写指令。

针对上述问题，本文提出了一种基于强制访问控制(MAC, mandatory access control)的模型 MOKIP。该模型为操作系统内核空间提供了一套安全机制，其基本方法也是通过限制非可信内核扩展模块的行为来保护操作系统内核的完整性安全。MOKIP 采用了信息流完整性模型的思想，将内核空间中的不同主体和客体设置不同完整性的安全标签，然后应用一些访问控制的规则确保具有低完整性标签的主体(即非可信内核模块)不会影响具有高完整性标签的客体(主要包括操作系统内核数据和内核代码)的安全。为了实现 MOKIP 模型，本文采用了最新的硬件辅助虚拟化技术，通过合理使用硬件辅助的页表机制(hardware assisted paging)将非可信内核扩展模块从操作系统内核隔离出来。利用虚拟机管理器(VMM, virtual machine monitor)的特权级动态截获非可信内核模块对操作系统内核数据的访问以及对内核代码的调用，并保证其操作符合本文的访问控制规则。对于那些违反安全规则的行为，VMM 将及时阻止并通知系统管理员。为了验证基于 MOKIP 模型的系统在实际运行环境中的有效性，本文向操作系统内核载入了 8 种包含不同恶意行为的内核扩展模块。实验结果表明本系统能够成功检测出所有这些模块的恶意行为。此外，还通过性能测试证明了本系统对上层应用程序的性能影响较小。

2 模型描述

MOKIP 模型的基本思想是通过限制操作系统内核空间中不同实体间的信息流来保证操作系统内核的完整性安全。

2.1 主体和客体

MOKIP 模型和传统的访问控制模型一样，需要区分实体 E 中的主体 S 和客体 O。在本模型中，主体 S 的定义为在内核空间中发出访问请求的实体；客体 O 的定义为在内核空间中接受主体访问的实体。其形式化表示为：$(\exists s\in S)(\exists o\in O)\ \text{CanAccess}\ (s,o)$。

在操作系统内核空间中，将主体 S 分为 3 类：操作系统内核($S_{\rm OS}$)、可信内核模块($S_{\rm TE}$)、非可信内核模块($S_{\rm UE}$)。其中，$S_{\rm OS}$ 是指操作系统最基本的部分，它主要负责管理系统资源；$S_{\rm TE}$ 是指对操作系统内核的可信扩展，它的引入通常需要额外的验证机制(比如微软的驱动签名技术[4])来证明其可信性；$S_{\rm UE}$ 是指对操作系统内核的非可信扩展，它通常由未知厂商或非可信个人开发。虽然能为用户提供特定的功能，但由于可能包含恶意代码，该类模块被载入后有潜在破坏操作系统内核安全的危险。为了区分可信内核模块与非可信内核模块，可以采用微软的驱动签名技术[5]。

对于客体，主要关注操作系统内核及其扩展模块的代码，数据和堆栈部分，具体包括：内核可信入口点代码($O_{\rm OS\text{-}TC}$)、内核其他代码($O_{\rm OS\text{-}other}$)、内核数据($O_{\rm OS\text{-}data}$)、内核堆栈($O_{\rm OS\text{-}stack}$)、可信扩展模块代码($O_{\rm TE\text{-}code}$)、可信扩展模块数据($O_{\rm TE\text{-}data}$)、可信扩展模块堆栈($O_{\rm TE\text{-}stack}$)、非可信扩展模块代码($O_{\rm UE\text{-}code}$)、非可信扩展模块数据($O_{\rm UE\text{-}data}$)、非可信扩展模块堆栈($O_{\rm UE\text{-}stack}$)。需要说明的是，内核数据($O_{\rm OS\text{-}data}$)仅包括属于内核本身的数据，不包括操作系统为内核模块动态分配的数据。此外，操作系统所依赖的特权寄存器($O_{\rm reg}$)也属于本模型的客体。

2.2 安全标签

为了实现主体 S 对客体 O 进行访问控制，在主体和客体的属性中引入了安全标签 L。每个实体仅有一个安全标签，不同的安全标签反应了不同实体的安全级别，用形式化表示为

$$(\forall s\in S)(\exists l_s\in L)\text{HaveLabel}(s,l_s)\wedge$$
$$(\forall o\in O)(\exists l_o\in L)\text{HaveLabel}(o,l_o)$$

本模型定义了 2 类安全标签：高完整性标签 $L_{\rm h}$ 和低完整性标签 $L_{\rm l}$。进一步定义了安全标签 L_1 和

L_2之间的支配关系≤，即如果 L_1 的完整性高于或等于 L_2，则称 L_1 支配 L_2，记做 $L_2 \leqslant L_1$。该支配关系适用于多级安全模型。由于本模型仅包含 2 类等级，显然有 $L_l \leqslant L_h$，即高完整性标签支配低完整性标签。此外，还定义了安全标签 L_1 和 L_2 之间的强支配关系<，即如果 L_1 的完整性高于 L_2，则称 L_1 强支配 L_2，记做 $L_2 < L_1$。

当某个主体 s 或客体 o 产生时，需要对其设置相应的安全标签。对于主体，为 S_{OS} 和 S_{TE} 设置高完整性标签 L_h，而为 S_{UE} 设置低完整性标签 L_l。对于客体，按照以下 2 条规则设置安全标签。

1）当客体 o 被主体 s 创建时，客体 o 继承主体 s 的完整性标签，记做 $\text{int}(o) \leftarrow \text{int}(s)$。该规则表明由高(低)完整性主体创建的客体将拥有高(低)完整性标签。例如，由操作系统内核 S_{OS} 创建的内核堆栈 $O_{OS\text{-}stack}$ 和内核数据 $O_{OS\text{-}data}$ 都将拥有高完整性标签 L_h。这里需要注意的是，认为特权寄存器(O_{reg})由 S_{OS} 创建，因此 O_{reg} 拥有高完整性标签 L_h。

2）当主体 s_1 请求另一主体 s_2 创建客体 o 时，该客体的完整性标签为 s_1 和 s_2 中被支配的，记做 $\text{int}(o) \leftarrow \text{dom}(\text{int}(s_1), \text{int}(s_2))$，即如果 $\text{int}(s_1) \leqslant \text{int}(s_2)$，则 $\text{dom}(\text{int}(s_1), \text{int}(s_2)) = \text{int}(s_1)$，$\text{int}(o) \leftarrow \text{int}(s_1)$。该规则适用于这样一些客体，它属于某个主体，但不由该主体创建。例如，非可信内核模块(S_{UE})请求操作系统内核(S_{OS})分配一片缓冲区，而该区域中的数据区属于 S_{UE}，其安全标签为 L_l。

对于静态客体，将非可信扩展模块数据($O_{UE\text{-}data}$)、非可信扩展模块堆栈($O_{UE\text{-}stack}$)和内核其他代码($O_{OS\text{-}other}$)标记成低完整性标签 L_l，而将内核可信入口点代码($O_{OS\text{-}TC}$)和内核数据($O_{OS\text{-}data}$)设置成高完整性标签 L_h。

2.3 保护规则

在内核空间中，操作系统内核和扩展模块之间及内部存在许多操作。例如，内核扩展模块向操作系统内核申请一片内存空间，然后向这片内存区写入数据。从信息流的观点[6]来看，所有的访问操作都可以映射成读类型(read)和写类型(write)操作。基于这 2 种操作，可以找出主体和客体在信息流之间的关系。如果主体 s 能够写客体 o，则存在信息流从 s 流向 o，记做 $\text{write}(s, o)$。另一方面，如果主体 s 能够读客体 o，则存在信息流从 o 流向 s，记做 $\text{read}(o, s)$。此外，信息流之间的关系还可以描述成流转换关系。如果主体 s_1 能写客体 o，而该客体 o 又能被另一主体 s_2 读取，则存在信息流从 s_1 到 s_2 的转换，记做 $\text{flowtrans}(s_1, s_2)$。

传统的信息流完整性模型(如 Biba 模型[2])要求信息流不能从低完整性客体流向高完整性主体。这样虽然能保证系统具有较高的完整性，但很难在实际环境中应用。例如，现实系统中许多高完整性主体都可能接收低完整性客体的输入。因此，完备的信息流完整性模型需考虑信息流从低完整性流向高完整性的情况。Clark-Wilson[7]完整性模型能较好地处理这些情况，它通过引入过滤器(filter)来完成对低完整性数据的更新或舍弃。然而该模型的不足是它要求所有接收数据的输入接口(input interface)都必须为过滤器，这极大影响了该模型的最终实现。为了在操作系统内核保护中应用完整性保护模型，借鉴 Clark-Wilson-Lite[8]模型的思想，重点关注于那些可能会接收低完整性数据的输入接口，只要能保证这些接口能正确处理非可信数据，就能从很大程度上确保操作系统内核的完整性安全。为了实现这个目标，本文设置了以下 3 条保护规则。

1) 当主体 s 执行客体 o 时，必须满足条件：客体 o 的完整性强支配主体 o 的完整性(即 $\text{int}(s) < \text{int}(o)$)。该规则表明当且仅当客体 o 的完整性高于主体 s 的完整性时，s 才能执行 o。例如，非可信内核模块(S_{UE})可以调用内核可信入口点代码($O_{OS\text{-}TC}$)，但不能执行内核的其他代码($O_{OS\text{-}other}$)或者非可信内核模块堆栈($O_{UE\text{-}stack}$)。该规则的形式化表示如下

$$(\forall s \in S)(\forall o \in O)(\text{int}(s) < \text{int}(o) \rightarrow \text{CanExecute}(s,o))$$

2) 当主体 s 读取客体 o 时，必须满足以下 2 个条件之一：①客体 o 的完整性支配主体 s 的完整性(即 $\text{int}(s) \leqslant \text{int}(o)$)；②主体 s 通过具有过滤功能的输入接口 I 读取客体 o。该规则表明只有客体 o 的完整性高于或等于主体 s 的完整性或者主体 s 通过过滤器接口 I 才能读取客体 o。例如，非可信内核模块(S_{UE})能够读取操作系统内核数据($O_{OS\text{-}data}$)，S_{OS} 可以通过滤器接口(即部分可信函数入口点)读取来源于 S_{UE} 的低完整性数据。该规则的形式化表示如下

$$(\forall s \in S)(\forall o \in O)(\text{int}(s) \leqslant \text{int}(o) \vee \text{I}(s,o) \rightarrow \text{CanRead}(s,o))$$

3) 当主体 s 写客体 o 时，必须满足条件：主体 s 的完整性支配客体 o 的完整性(即 $\text{int}(o) \leqslant \text{int}(s)$)。该规则表明当且仅当主体 s 的完整性高于或等于客体的完整性时，s 才能写 o。例如：操作系统内核(S_{OS})

可以修改非可信内核模块数据($O_{\text{UE-data}}$)，但非可信内核模块(S_{UE})却不能修改操作系统内核数据($O_{\text{OS-data}}$)。该规则的形式化表示如下

$$(\forall s \in S)(\forall o \in O)(\text{int}(o) \leqslant \text{int}(s) \rightarrow \text{CanWrite}(s,o))$$

2.4　转换规则

虽然本文的保护规则能保证系统的完整性，但是却从一定程度上影响了系统的可用性。例如，本文的保护规则不允许高完整性主体从非过滤器接口读取低完整性数据，然而在实际中却可能出现一些相反的情况。为增强 MOKIP 模型的可用性，设置了以下 2 条转换规则。

1) 当高完整性主体 s 从非过滤器接口 I 读取低完整性客体 o 时，该主体的高完整性将降至低完整性，该形式化表示为

$$(\forall s \in S)(\forall o \in O)(I(s,o) \wedge (\text{int}(s)=L_{\text{h}}) \rightarrow (\text{int}(s)=L_{\text{l}}))$$

该规则适用于这样的可信内核模块，当它通过非过滤器接口接收低完整性数据以后，其行为将受到限制，比如该模块将无法写高完整性数据。

2) 低完整性主体 s 能通过转换器接口 T 提升为高完整性，从而能使该主体 s 能写高完整性客体 o，该形式化表示为

$$(\forall s \in S)(T(s) \wedge (\text{int}(s)=L_{\text{l}}) \rightarrow (\text{int}(s)=L_{\text{h}}))$$

该规则可看成是保护规则的例外，它允许低完整性主体通过某种方式写高完整性客体来实现某些特定功能，然而这些例外规则必须由系统管理员显式指明。

3　模型的基本实现

在宏内核构架下，操作系统内核和内核扩展模块处于同一地址空间，可以不受限制地相互访问，因此在传统系统架构下很难对它们之间的交互进行访问控制。针对这一问题，可以利用了硬件辅助的虚拟化技术。在虚拟化环境中，虚拟机管理器运行在最底层，负责管理和调度上层客户机操作系统(guest OS)的运行。本文通过修改开源虚拟化平台 Xen[9]基本实现了 MOKIP 模型。假设操作系统在启动时除操作系统内核以外仅包含可信内核模块，并且可信内核模块总是通过过滤器接口读取低完整性数据。因此，本文的实现可以简化成怎样利用虚拟机管理器监控从操作系统外部载入的非可信内核模块的行为，其具体实现包括：对操作系统内核对象的标记机制和对内核模块的访问控制机制。图 1 给出了本原型系统的基本架构，其中系统的核心模块位于虚拟机监控器层，具体包括访问控制模块和标记模块。系统的工作流程可分为 2 步：首先，标记模块对内核层的实体设置安全标签；其次，访问控制模块将根据安全标签的属性控制内核扩展模块对内核代码和内核数据的访问。

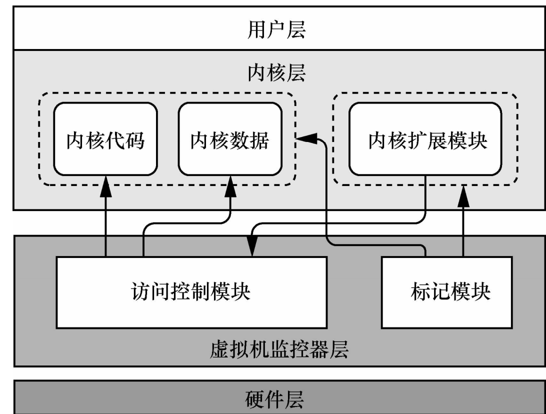

图 1　系统架构

3.1　硬件辅助的虚拟化技术

MOKIP 模型的实现依赖于硬件辅助的内存虚拟化技术。利用 Intel 扩展页表技术(EPT, extended page tables)[10]来监控内核模块在内核空间中的活动。当 EPT 功能被打开时，客户机内存地址转换过程如图 2 所示，客户机页表负责客户机虚拟地址(GVA, guest virtual address)到客户机物理地址(GPA, guest physical address)的转换，EPT 页表负责客户机物理地址(GPA)到主机物理地址(HPA, host physical address)的转换。EPT 页表的结构与传统页表相类似，其页表项包含物理页帧和相应的权限位(RWX)，其中，R 代表可读，W 代表可写，X 代表可执行。若权限位被置 1，则相应物理页帧上的权限被打开；若权限位被置 0，则相应物理页帧上的权限被关闭。

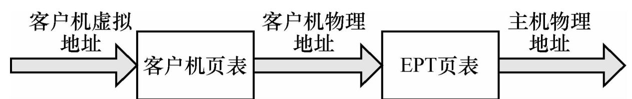

图 2　EPT 地址转换过程

3.2　标记机制

实现标记机制的目的是为了对内核空间中的主体和客体设置相应安全标签。以页粒度为单位，利用了 EPT 页表项中的保留位对内核空间中不同的页面设置不同的安全标签。因为 EPT 中的保留位

没有被VMM使用，改变它们的值将不会影响VMM的正常功能。目前，只使用了其中的一位，通过置1表示为高完整性标签 L_h，通过置0表示低完整性标签 L_l。该标记机制的好处是不需要额外的内存空间来存储安全标签，并且还能从很大程度上减少了查找安全标签的时间开销。

由于操作系统内核所维护的内存总是动态变化的，很难仅依赖VMM对这些动态内存设置相应安全标签。为了克服这种语义鸿沟[11]，通过修改操作系统内核让其主动向VMM报告动态内存信息，具体包括起始地址、物理页帧和所属类型(包括可信和非可信2类)，然后VMM会根据这些信息对相应的EPT页面设置安全标签。图3给出了基本的安全标记算法。

```
1) Initializing security labels for the entire kernel;
2) if ( S_UE is loaded into kernel memory ){
3)        int(S_UE)=L_l;
4) }
5) if ( S_UE requests allocating memory M ){
6)        if (The allocation succeeds){
7) int(O_M)=L_l;
8) }
9) }
```

图3 安全标记算法

当操作系统启动时，将内核空间中所有已分配的页面设置成高完整性标签 L_h。当非可信内核模块被载入时，操作系统将动态为其分配内存。如果该非可信模块所需的内存是按页大小对齐，操作系统将直接从空闲页面中分配一些内存页面供它使用。对于这样的情况，本系统直接将这些页面设置成低完整性标签 L_l。另一方面，如果非可信内核模块所需的内存不是按页大小对齐，操作系统将从现有非空闲页面中为其分配内存空间，这样将导致内核中的一些内存页面内既包含可信部分(操作系统内核和可信内核模块)又包含非可信部分(非可信内核模块)。对于这种情况，本系统将无法为这些混合页面设置安全标签。

为了避免混合内存页面的出现，本文修改了Linux操作系统中的slab内存分配器[12]。其基本思想是使目标操作系统为非可信实体分配的内存不与为可信实体分配的内存处于同一物理页面。在Linux系统中，slab内存分配器为不同的内核对象生成不同的缓冲。某个对象的缓冲由一连串的slab构成，每个slab又由一个或多个连续的物理页组成，包含了若干同种类型的对象。通过修改Linux内核将slab分成2类：一类为可信slab，专门为操作系统内核和可信内核模块分配内存；另一类为非可信slab，专门为非可信内核模块分配内存。通过截获slab的分配函数判断其内存分配请求是来源于可信部分还是非可信部分。对于操作系统内核或可信内核模块的请求，使用可信slab分配内存；对于非可信内核模块的请求，使用非可信slab分配内存。这种分离可信和非可信slab的方法极大方便了本系统的标记机制，将可信slab所在的内存页面设置成高完整性标签 L_h，将非可信slab所在的页面设置成低完整性标签 L_l。

3.3 访问控制机制

为了限制非可信扩展模块的行为，必须在内核空间对其进行访问控制。根据MOKIP模型的保护规则，低完整性主体可以访问低完整性客体但不能随意访问高完整性客体，即非可信扩展模块能够读写自身的数据，执行自身代码；但不能写高完整性内核数据，不能任意执行高完整性内核代码(可信入口点代码除外)。基本的安全访问控制算法如图4所示。

```
1) if ( S_UE requests accessing memory M ){
2)        if (int(O_M)≤int(S_UE)){
3)               allow access;
4)        }
5)        else if(M is a trusted code entry point ){
6)               allow access;
7)        }
8)        else
9)               disallow access;
10) }
```

图4 安全访问控制算法

为了使非可信扩展模块在运行时满足保护规则，根据不同安全标签隔离不同完整性的主体和客体。当非可信内核模块被执行时，利用VMM将操作系统内核数据和代码所在的EPT页面设置成不可写和不可执行。如果此时非可信模块试图写内核数据或执行内核代码，将会产生EPT页异常而陷入到VMM中，使本系统可以通过VMM中的异常处理函数阻止其非法操作。另一方面，非可信模块可能会执行某些合法操作，比如调用内核可信入口函数。为保证其正常运行，使用VMM重置操作系统内核及非可信扩展模块所对应的EPT权限位，具体如图5所示。将操作系统内核数据置成可写，将内核代码置成可执行，同时将非可信模块置成不可执行。当操作系统返回或又调用非可信模块时，再次

利用 VMM 重置 EPT 权限位，保证非可信内核模块与操作系统内核之间的隔离。

由于每次操作系统内核和非可信内核模块的执行进行切换时都需要重置 EPT 页表，这将给系统带来较大的性能开销。为了提升系统的性能，引入了 2 套 EPT 页表分别对应操作系统内核和非可信模块的执行状态。当不同执行状态进行切换时，本系统只需要改变 EPT 基指针而无需重新设置 EPT 页表。为了减少 2 套页表切换时所引起的 TLB 刷新，为每套 EPT 页表设置不同的 VPID(virtual process ID)。此外，为保证 2 套 EPT 页表的一致性，本系统需要截获所有对 EPT 的修改操作来完成它们之间的数据同步。

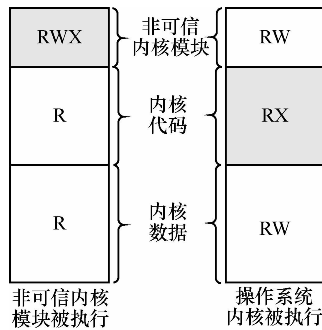

图 5　系统不同执行状态所对应不同的 EPT 权限位

除了限制非可信内核模块访问操作系统内存数据以外，还需要防止非可信内核模块非法修改特权寄存器，比如本系统不允许非可信模块使用 LIDT 指令重新加载 IDT 表。为此，利用 Intel VT 技术[10]将虚拟机控制结构(VMCS)中描述符表退出(descriptor table exiting)字段值设为 1，使 VMM 可以直接捕获非可信内核模块对 IDTR、GDTR 等特权寄存器的修改操作。对于非法修改，通过 VMM 进行指令模拟以跳过这些操作，从而保证特权寄存器的安全。

4　实验评估

本文使用了 Dell PowerEdge T310 作为测试环境，其配置为 Intel Xeon X3430 2.4 GHz CPU，4 GB 内存；Xen 64 位 3.4.2 版本，Dom0 主机为 x86_64 Fedora 12(内核版本为 Linux-2.6.31)，客户机操作系统为 x86_64 Ubuntu 8.04(内核版本为 Linux-2.6.24)。本文测试了原型系统对客户机操作系统内核保护的有效性和性能 2 方面。

4.1　有效性测试

有效性测试的目的是检测基于 MOKIP 模型的系统能否有效地抵御恶意内核模块对操作系统内核的攻击。通过 Linux insmod 命令向操作系统内核分别载入了 8 种包含不同恶意行为的内核扩展模块，具体见表 1，其中前 6 种为真实的 rootkit，后 2 种为自行开发的恶意内核模块。对于新引入的 ROP rootkit，由于其没有使用能接收低完整性数据的输入接口（即可信内核 API 函数入口点）调用内核代码，所以违反了本文的保护规则。实验结果表明所有这 8 个内核模块的恶意行为都被本系统成功检测。

需要指出的是 MOKIP 模型仅适用于检测那些破坏系统完整性的内核模块。如果有恶意内核模块用于破坏系统的机密性（比如读取内核中的敏感信息），本系统将无法检测这类内核攻击。

4.2　性能测试

本文设计了 5 个性能实验，测试了 MOKIP 系统和原始的虚拟化平台 Xen 对上层应用程序的性能影响。实验 1 和实验 2 是通过监控 ext3 文件系统测试解压 linux 内核压缩包（即 linux-2.6.24.tar.gz）和编译 linux 内核源代码的时间；实验 3 和实验 4 是通过监控 8139too 网卡驱动测试 Apache 服务器的传输率和 Lighttpd 服务器的吞吐率（2 个 Web 服务器

表 1　有效性实验测试结果

恶意内核模块	恶意行为	检测结果	检测原因
Adore-ng 0.56	修改内核函数指针	成功检测	低完整性 S_{UE} 写高完整性 $O_{\mathrm{OS\text{-}data}}$
EnyeLKM	修改内核二进制代码	成功检测	低完整性 S_{UE} 写高完整性 $O_{\mathrm{OS\text{-}code}}$
Override	修改内核系统调用表	成功检测	低完整性 S_{UE} 写高完整性 $O_{\mathrm{OS\text{-}data}}$
All-root	修改内核进程链表和系统调用表	成功检测	低完整性 S_{UE} 写高完整性 $O_{\mathrm{OS\text{-}data}}$
Hp	修改内核进程链表	成功检测	低完整性 S_{UE} 写高完整性 $O_{\mathrm{OS\text{-}data}}$
Lvtes	修改内核系统调用表和模块链表	成功检测	低完整性 S_{UE} 写高完整性 $O_{\mathrm{OS\text{-}data}}$
ROP rootkit	修改自身堆栈地址和利用现有内核代码	成功检测	低完整性 S_{UE} 执行高完整性 $O_{\mathrm{OS\text{-}other}}$
破坏 IDT rootkit	修改 IDT 特权寄存器	成功检测	低完整性 S_{UE} 写高完整性 O_{reg}

都加载了同一个 132 Kbit 大小的 html 网页）；实验 5 是通过监控 USB 驱动测试从 U 盘拷贝文件的传输率。最终结果如表 2 所示。

表 2　应用程序在虚拟机中的性能对比

性能	受控内核模块	Xen	MOKIP
解压内核压缩包的时间	ext3	35 367 ms	39 568 ms
编译内核代码的时间	ext3	2 812 s	3 176 s
Apache 服务器的传输率	8 139too	2 253 kbit/s	2 013 kbit/s
Lighttpd 服务器的吞吐率	8 139too	5 362 kbit /s	4 867 kbit /s
拷贝文件的传输率	uhci-hcd	692 kbit/s	611 kbit/s

与原始的 Xen 相比，MOKIP 系统先后隔离并监控了 3 个不同的内核扩展模块，对相关应用程序的性能开销都在 13%以内，因此能够满足一般的应用需求。

为了测试 MOKIP 系统对内核模块装载的内存开销，还测量了隔离 ext3 文件系统、8139too 网卡驱动和 USB 驱动对应 slab 所需的内存页面个数。表 3 给出了 100 次平均采样的测量结果。与原始系统相比，本系统引入的内存开销不到 1%。

表 3　slab 所需内存页面个数对比

操作	受控内核模块	Xen	MOKIP
编译内核代码	ext3	5 183	5 227
访问 Apache 服务器	8139too	4 719	4 736
拷贝文件	uhci-hcd	4 862	4 883

5　相关工作

美国 CMU 大学开发了一种轻量级虚拟机管理器 Secvisor[13]，旨在保护操作系统内核代码的完整性安全。Secvisor 通过解析和检查对 MMU 和 IOMMU 的修改操作，确保了只有被系统管理员认证过的代码才能在内核态执行。本文的工作 MOKIP 虽然借鉴了 Secvisor 利用虚拟机管理器管理 MMU 的思想，但主要不同在于：Secvisor 禁止在内核空间执行非可信代码，MOKIP 能在保证操作系统内核的完整性前提下执行非可信内核代码。NICKLE[14]和 hvmHarvard[15]系统均采用了分离指令取址和数据访问的思想，确保内核指令仅能从可信内核代码区域取址。这种方法虽然也能保证内核代码的完整性，但是却无法抵御利用现有内核代码的攻击[16]。$UCON_{KI}$[3]将 UCON 模型引入到操作系统内核完整性保护中，实现了对非可信内核模块访问控制的连续性和可变性。然而 $UCON_{KI}$ 需要依赖很多专家知识来制定内核保护策略，因此不易在实际环境中应用。美国加州大学 Davis 分校的 Daniela 等[1]通过采用 Biba 模型[2]中不能上写的思想，提出了一种能感知虚拟化的操作系统和虚拟机管理器相协作的方法以保护操作系统内核代码和数据的安全。该方法虽然能较为全面地保护系统内核，但由于需要其基于模拟器的 VMM 截获每一个条对内核代码和数据的写指令，其性能开销较大，因此也不适合在实际环境中使用。LXFI[17]采用源代码插桩技术实现了对脆弱内核模块的隔离。KFUR[18]结合源代码插桩和动态监控方法检测内核模块调用内核函数是否符合其使用规则。然而，LXFI 和 KFUR 都需要修改内核模块源代码，而且不能对内核数据进行有效地访问控制。郑豪等[19]利用虚拟化技术对内核模块中的故障进行了隔离。该方法能有效提高系统可靠性，但无法阻止内核模块对内核资源的非法访问。

6　结束语

为了保护操作系统内核完整性安全，本文提出了一种基于强制访问控制的 MOKIP 模型，该模型根据其标记规则将内核空间中的不同主体和客体设置不同的安全标签，然后运用保护规则限制低完整性主体访问高完整性客体。为了提高模型的可用性，还设置了转换规则以应对例外的情况。为实现基于 MOKIP 模型的系统，采用硬件辅助的虚拟化技术(比如 Intel EPT 和 Intel VT 技术)将非可信内核模块从操作系统内核空间隔离出来，然后对其行为进行访问控制。有效性和性能实验证明了本系统能够检测恶意内核模块破坏内核完整性的攻击，对应用程序的性能开销在可接受的范围内。

参考文献：

[1] DANIELA A S O, WU S F. Protecting kernel code and data with a virtualization-aware collaborative operating system[A]. Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC)[C]. Honolulu, Hawaii, 2009. 451-460.

[2] BIBA K J. Integrity consideration for secure compuer system[R]. Technical report, Mitre Corp. Report TR-3153, Bedford, Mass, 1977.

[3] XU M, JIANG X X, RAVI S, *et al*. Towards a VMM-based usage control framework for OS kernel integrity protection[A]. Proceedings of the 12th ACM Symposium on Access Control Models and Technologies[C]. Sophia Antipolis, France, 2007. 71-80.

[4] Microsoft Corporation. Windows Driver Signing[EB/OL]. http://www.

microsoft.com/.
[5] Windows Vista Security Blog[EB/OL] http://blogs.msdn.com/ windowsvistasecurity/archive/2007/08/16/.
[6] GUTTMAN J, HERZOG A, RAMSDELL J. Information flow in operating systems: eager formal methods[A]. Workshop on Issues in the Theory of Security (WITS)[C]. 2003.
[7] SANDHU R S. Lattice-based access control models[J]. IEEE Computer, 1993, 26(11):9-19.
[8] SHANKAR U, JAEGER T, SAILER R. Toward automated information-flow integrity verification for security-critical applications[A]. Proceedings of the 13th Network and Distributed System Security Symposium (NDSS)[C]. 2006.
[9] BARHAM P, DRAGOVIC B, FRASER K, *et al*. Xen and the art of virtualization[A]. Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP)[C]. 2003. 164-177.
[10] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manuals[EB/OL]. http://www.intel.com/Assets/PDF/manual/253669.pdf.
[11] PETER M C, BRIAN D N. When virtual is better than real[A]. Proceedings of the 2001 Workshop on Hot Topics in Operating Systems (HotOS)[C]. 2001. 0133.
[12] DANIEL B, MARCO C, Understanding the Linux Kernel[M]. O'Reilly & Associates Inc, third edition, 2005.
[13] SESHADRI A L M Q N. PERRIG A. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes[A]. Proceedings of the 24th ACM Symposium on Operating System Principles (SOSP)[C]. 2007. 335-350.
[14] RYAN R, JIANG X X, XU D Y. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing[A]. Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID)[C]. 2008. 1-20.
[15] MICHAEL G, WANG Z, DEEPA S, *et al*. Transparent protection of commodity OS kernels using hardware virtualization[A]. Proceedings of the 6th International Conference on Security and Privacy in Communication Networks (SecureComm)[C]. 2010. 162-180.
[16] RALF H, THORSTEN H, FELIX C F. Return-oriented rootkits: bypassing kernel code integrity protection mechanisms[A]. Proceedings of 18th Usenix Security Symposium (Usenix Security)[C]. 2009. 383-398.
[17] MAO Y D, CHEN H G, ZHOU D, *et al*. Software fault isolation with API integrity and multi-principal modules[A]. Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP)[C]. 2011.115-128.
[18] 马超, 尹杰, 刘虎球, 等. KFUR:一个新型内核扩展安全模型[J]. 计算机学报, 2012, 35(10): 2091-2100.
MA C, YIN J, LIU H Q, *et al*. KFUK: a new rernel extension security model[J]. Chinese Journal of Computers, 2012, 35(10): 2091-2100.
[19] 郑豪, 董小社, 王恩东, 等. VM 内部隔离驱动程序的可靠性架构[J]. 软件学报, 2014, (10): 2235-2250.
ZHENG H, DONG X S, WANG E D, *et al*. Reliability architecture to isolate the driver inside the VM[J]. Journal of Software, 2014, (10): 2235-2252.

作者简介：

田东海（1984-），男，湖南长沙人，博士，北京理工大学讲师，主要研究方向为操作系统安全、虚拟化技术、智能终端安全等。

陈君华[通信作者]（1975-），男，四川仪陇人，博士，云南民族大学副教授，主要研究方向为物联网应用和信息安全等。E-mail: chenjunhuabj@163.com。

贾晓启（1982-），男，北京人，博士，中国科学院信息工程研究所副研究员，主要研究方向为操作系统安全、云计算安全和虚拟化技术等。

胡昌振（1967-），男，湖北汉川人，博士，北京理工大学教授，主要研究方向为网络安全、软件安全和模式识别等。