HybridHP：一种轻型的内核完整性监控方案及其形式化验证

钱振江[1),2),3)] 刘 苇[1),2)] 黄 皓[1),2)]

[1)](南京大学软件新技术国家重点实验室 南京 210046)

[2)](南京大学计算机科学与技术系 南京 210046)

[3)](常熟理工学院计算机科学与工程学院 江苏 常熟 215500)

摘 要 虽然传统的虚拟化监控方法可以在一定程度上保障操作系统安全.然而,虚拟监控器VMM中管理域Domain0的存在以及操作系统级的切换所带来的性能损失是很多具有大型应用的操作系统所不能接受的.注重硬件虚拟化技术的监控能力而摒弃其不必要的虚拟化能力,提出了一个新型的通用的虚拟化监控框架HybridHP,并实现其原型.HybridHP将管理域和虚拟机监控机制两者整合到被监控操作系统的地址空间,具有很好的获取被监控系统操作语义的能力.利用Isabelle/HOL形式化辅助证明工具验证HybridHP的隔离性、安全性和监控能力.最后对HybridHP进行了攻击实验和性能评估,结果显示HybridHP提供了和传统的虚拟化监控方案相同的安全保障,并具有很好的系统性能.

关键词 硬件虚拟化;内核完整性;安全监控;安全攻击;Isabelle形式化验证

中图法分类号 TP309 **DOI号**: 10.3724/SP.J.1016.2012.01462

HybridHP: A Verified Lightweight Approach to Provide Lifetime Kernel Integrity Surveillance

QIAN Zhen-Jiang[1),2),3)] LIU Wei[1),2)] HUANG Hao[1),2)]

[1)](*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210046)

[2)](*Department of Computer Science and Technology, Nanjing University, Nanjing* 210046)

[3)](*School of Computer Science and Engineering, Changshu Institute of Technology, Changshu, Jiangsu* 215500)

Abstract Although traditional virtualization monitoring can help ensure security, the existence of management domain (such as Domain0) and performance loss caused by OS-level switches make these approaches unsuitable for many OSs with large applications. In this paper, focusing on monitoring capability of the hardware virtualization technology without the unnecessary virtualization functionality, we propose HybridHP, a new general-purpose framework of virtualization monitoring, and implement the prototype. HybridHP merges the management domain and virtual machine monitoring functionality into the monitored system, and has strong ability to obtain operational semantics of the monitored system. We use the formal theorem prover Isabelle/HOL to verify isolation, security and monitoring capability of HybridHP. With the systemic experiments and performance evaluation for HybridHP, we show that HybridHP provides at least the same

收稿日期:2011-04-18;最终修改稿收到日期:2012-02-13.本课题得到国家"八六三"高技术研究发展计划项目基金(2007AA01Z409)、江苏省科技支撑计划自然科学基金(BE2008124)以及江苏省"六大人才高峰"高层次人才项目(2011-DZXX-035)资助.**钱振江**,男,1982年生,博士研究生,讲师,中国计算机学会(CCF)会员,主要研究方向为操作系统安全、形式化验证和嵌入式系统.E-mail: zhenjiang.qian@gmail.com. **刘 苇**,男,1986年生,硕士研究生,主要研究方向为虚拟机监控、形式化验证.**黄 皓**,男,1957年生,博士,教授,博士生导师,主要研究领域为系统软件、信息安全.

security guarantees as what can be achieved by the traditional virtualization monitoring approaches, and has well system performance.

Keywords　hardware virtualization; kernel integrity; security monitoring; security attacks; Isabelle formal verification

1　引　言

内核级的攻击和恶意程序可以轻易地破坏整个操作系统的完整性，而传统监控方案的本质缺陷在于恶意程序可能获得和内核一样的权限，导致操作系统没有保护自身的能力. 近年来硬件虚拟化技术(如 Intel-VT 和 AMD-SVM 等)的发展为基于虚拟机(如 XEN[1] 和 KVM[2] 等)的监控方式提供了底层的支持和保障. 总的来说，由于虚拟机监控器位于内核的底层，比内核具有更高的权限，所以可以中断内核的执行，对内核的执行进行审查和检验，从而保证内核的完整性. 这种实时性也是虚拟机监控器区别于其它监控方案的最大优势. 很多的学者对此进行了研究[3-4]. 基于虚拟机的监控器能够捕捉到包括内存在内的资源访问动作，使得在内核执行过程中能够根据内核执行路径中所访问的对象进行选择性的事件触发，从而提高了监控器监控的实时性.

已有的虚拟化监控方案[3,5-11]，普遍存在这么几个方面的问题：

(1) 性能损失. 这些方案注重虚拟化机制，而非监控本身. 在监控单独的操作系统时，它们往往需要付出很大的额外代价，例如采用 XEN 进行监控的方式，需要有单独的管理域 Domain0，像 I/O 之类的操作需要管理域 Domain0 的干预处理，这极大地影响了被监控系统的性能；

(2) 可信基(Trusted Computing Base，TCB)膨胀. 虚拟机自身的代码量往往过于庞大(如 XEN 3.4.1 拥有 230 K SLOC 的代码量)，再加上管理域 Domain0 的系统代码量，我们面临着虚拟机本身以及相应的管理域 Domain0 的正确性问题，很难保证监控方案自身不会引入程序错误；

(3) 语义获取难. 引用监控器位于被监控系统地址空间之外，因此引用监控器看到的视图和被监控系统的是有差别的，很难去理解被监控系统中对某个内存地址的具体操作语义，管理域进行监控的过程需要对被监控系统的具体操作语义进行转化，而这一过程是很难做到实时而全面的. 虽然 Sharif 等人在被监控系统中构建了一个监控器 SIM[12]，以便于取得被监控系统的语义，但是 SIM 与被监控系统的隔离性仍需要依赖于另一个监控器的保护，而本文提出的内嵌式监控器的自我保护能力不依赖于其它任何安全机制.

在本文中，给出了一个通用的虚拟化框架 HybridHP，其特点如下：

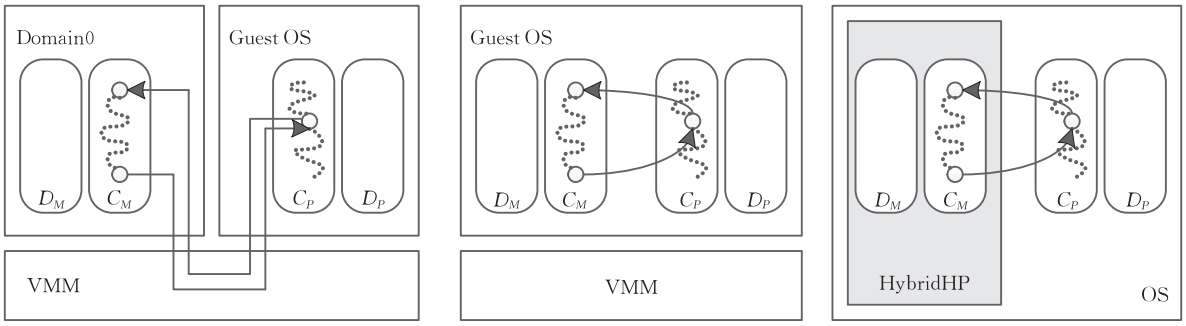
(1) HybridHP 注重于监控本身而非虚拟化技术. HybridHP 没有 Domain0 之类的单独管理域概念，HybridHP 以模块的方式嵌入到被监控系统的内核空间中进行监控服务. HybridHP 只负责监控系统的运行，能够截获被监控系统中的特权操作、异常和对寄存器、内存、I/O 等的访问，且不受原有内核模块的影响和破坏，并且不会干预其 I/O 的操作，为此被监控系统的性能不会受到很大的影响；

(2) 由于采用模块的方式运行，HybridHP 看到的内存视图和被监控系统看到的是一致的，这样容易获得被监控系统的操作语义，监控的正确率和效率可以得到很大的提高；

(3) 由于 HybridHP 监控功能的单一性，其代码量可以控制在很小的范围，这便于对其正确性进行证明，我们利用 Isabelle[13] 形式化辅助证明工具对其进行了正确性证明.

总的来说，HybridHP 是被监控系统内部的一个模块，它利用硬件虚拟化技术，使自身独立于被监控系统，具有比被监控系统中的原有模块更高的权限. 图 1 显示了 HybridHP 与已有虚拟化监控方案的区别，其中 C_M、D_M 是监控器的代码段和数据段，C_P、D_P 是被监控系统的代码段和数据段. 区别于已有的虚拟化监控方案，由于 HybridHP 以模块方式运行，其自身的代码和数据信息暴露在被监控系统的视图下，很容易受到其它恶意模块的破坏，所以它自身的安全性和隔离性是我们成功与否的关键. 为此，我们提出了相应的保护机制，并且在高层使用形式化的方法并借助 Isabelle 证明其安全性和隔离性.

HybridHP 是一种完全内嵌式的监控方案，不借助其它任何的辅助监控措施，并且其正确性经过



(a) 传统的虚拟化监控框架 (b) SIM的监控框架 (c) HybridHP的监控框架

图 1 HybridHP 与已有的虚拟化监控方案的区别

严格的形式化逻辑验证.

本文第 2 节阐述 HybridHP 的整体框架设计;第 3 节说明 HybridHP 原型的具体实现细节和关键技术;第 4 节给出 HybridHP 正确性的 Isabelle 形式化证明;第 5 节阐述对 HybridHP 的测试结果和系统性能评估;第 6 节对本文进行总结和对未来的工作进行展望.

2 HybridHP 的设计

2.1 HybridHP 的目标

HybridHP 的设计目标是能够利用硬件虚拟化技术对操作系统实施监控,但不会对操作系统的性能和效率造成太大的影响,为此需要实现以下几个具体目标:

(1) 监控程序本身是安全的. 为了实现这个目标,要求监控程序非常小,其本身是可以通过形式化验证的,这样才能保证它自身的安全可靠和可控;

(2) 监控程序自我保护. 监控程序本身要有自我保护的能力,防止其它恶意模块的破坏;

(3) 监控程序不可旁路. 监控程序能够截获被监控系统中的特权操作和对关键内存对象、寄存器、I/O 设备的访问等. 恶意模块不能绕过监控程序执行非法操作和修改受保护的关键数据;

(4) 机制与策略分离. 监控程序本身提供的只是监控机制,具体的策略是可以实时更新的;

(5) 被监控系统的性能不会受到太大的影响. 已有的监控方案正是由于架构的原因,极大地损失了系统的性能,所以并不是很适用;

(6) 易于获得被监控系统的语义. 获得操作系统的语义是监控程序一个很重要的任务,而已有的虚拟机监控器由于虚拟机看到的内存视图和被监控系统看到的是有差别的,很难去理解被监控系统的具体操作语义.

2.2 HybridHP 的整体框架

HybridHP 的整体框架如图 2 所示. HybridHP 的核心特点在于 HybridHP 以模块的方式在被监控系统内核空间中运行,由于使用了硬件虚拟化技术,比其它的内核模块具有更高的权限. 整个内核空间处于 0 级模式(Ring 0),但是 Intel-VT 又将 0 级分成两种子模式:VMX Root 模式和 VMX Non-Root 模式. VMX Root 是真正的 0 级,具有所有权限. VMX

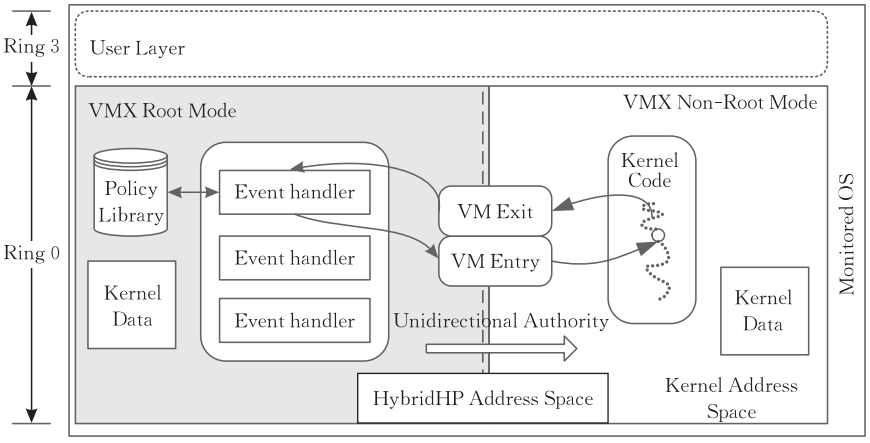


图 2 HybridHP 框架

Root 模式的操作方式和没有开启硬件虚拟化机制(VMX)时的操作基本上是相同的，不同之处在于 VMX Root 模式中能够调用 VMX 的指令，同时，一些寄存器中能够装载的值会受到一定的限制. 在本文的框架中，HybridHP 运行在 VMX Root 模式. VMX Non-Root 是受限的 0 级，其中很多的特权操作和事件会触发控制权转移到 VMX Root 模式的监控程序. 对应地，框架中的被监控系统运行在 VMX Non-Root 模式.

HybridHP 本身不提供虚拟化，没有管理域和驱动域的概念. 对于系统的驱动部分(如 I/O 访问)，被监控系统的驱动模块直接和硬件交互，不需要 HybridHP 的干预，这样对系统的性能不会造成太大的影响. 当然，如果需要 HybridHP 进行监控和管理的话，也可以通过对 HybridHP 增加相应的策略来达到目的. 对于监控能力，HybridHP 只提供监控的机制，采用机制和策略相分离的方式. 为此，HybridHP 启动之后，首先从外部环境获得策略信息. 为了避免外部环境的不可信以及恶意策略对 HybridHP 完整性的破坏，HybridHP 使用数字签名的方式来进行策略的安全更新. HybridHP 根据策略对内核的运行过程进行实时监控，如关键数据的保护、异常行为触发等. 运行在 VMX Non-Root 模式的被监控系统触发受 HybridHP 监控的事件，通过两种模式之间唯一的入口 VM Exit 进入 VMX Root 模式，由 HybridHP 根据策略信息进行处理，处理完成后通过 VM Entry 返回 VMX Non-Root 模式.

由于 HybridHP 和被监控系统采用的是相同的页表视图，HybridHP 对于被监控系统的对内存地址的操作语义不需要经过转换而可以直接理解，提高监控处理的效率. 同时，HybridHP 具有自我保护的能力，被监控系统虽然可以看到 HybridHP 的页表视图，但是无法对其进行修改，而 HybridHP 拥有修改被监控系统内存视图的权限. 因此，从地址空间的角度来讲，被监控系统内核的地址空间其实是 HybridHP 地址空间的子集，HybridHP 和内核其它模块对于地址空间视图的权限是单向的.

3　HybridHP 的实现

为了验证本文的方案，实现了 HybridHP 的原型. 本节对 HybridHP 的实现和主要关键技术进行阐述.

HybridHP 以模块的方式在被监控系统内核空间中运行，区别于系统中的其它模块，在 HybridHP 中开启硬件虚拟化(VMX)，HybridHP 运行在 VMX Root 模式，而整个被监控系统运行在 VMX Non-Root 模式，所以 HybridHP 具有比其它的内核模块更高的权限. 如何做到在 HybridHP 框架中开启硬件虚拟化，使得 HybridHP 运行于 VMX Root 模式，而被监控系统运行于 VMX Non-Root 模式是需要解决的第 1 个问题.

在 HybridHP 框架中，控制 VMX Root 和 VMX Non-Root 运行模式的关键是 VMX 控制结构 VMCS，主要包括 3 个部分：

(1) 被监控系统的状态区(Guest State Area)，用于在 VMX Non-Root 向 VMX Root 模式切换时保存被监控系统的状态信息，如段寄存器、CR 寄存器、指令指针 IP、栈顶指针 SP 等，以及从 VMX Root 返回 VMX Non-Root 模式时的 VM Entry 入口地址；

(2) 主机状态区(Host State Area)，存放监控系统自身的状态信息. 在 HybridHP 框架中，主机状态区主要包括 HybridHP 运行过程的各种状态信息，如与 HybridHP 运行相关的段寄存器、CR 寄存器、HybridHP 栈顶指针 SP，以及从 VMX Non-Root 向 VMX Root 模式切换时的 VM Exit 处理程序的入口地址；

(3) HybridHP 执行控制区(Execution Control Area)，主要包括 HybridHP 监控过程的配置信息，如需要保护的寄存器信息(CR0 等)、HybridHP 需要捕获的特权指令(如 CPUID 等)和异常(如 page-fault 等)信息以及相应的处理方式和处理函数入口地址. 同时在 HybridHP 执行控制区中还存放了产生 VMX Non-Root 到 VMX Root 模式切换的原因，包括错误码、触发切换的特权指令以及被监控系统期望修改的寄存器或者对象的地址等信息.

在被监控系统启动后，以模块的方式加载 HybridHP. HybridHP 启动的初始化流程包括：

(1) 分配 HybridHP 的内存空间，主要包括内核栈、动态数据区、以及 VMX 控制结构 VMCS 所需的空间；

(2) 通过 VMXON 指令开启硬件虚拟化，设置被监控系统的状态区；

(3) 设置 VMCS 中的主机状态区，其中 VM Exit 的入口地址指向 HybridHP 的处理函数入口，页目录地址寄存器 CR3 设置为与被监控系统相同，

使得 HybridHP 与被监控系统采用相同的页表视图；

(4)设置 HybridHP 执行控制区，包括需要保护的寄存器(如 CR0 等)、需要捕获的特权指令(如 CPUID 等)以及异常(如 pagefault 等)信息，这是 HybridHP 监控的核心，同时根据策略信息将关键数据区域所在的页面以及被监控系统页表所在的页等数据设置成在 VMX Non-Root 模式下只读，开启对数据的保护；

(5)通过 VMLANCH 指令返回被监控系统 VMX Non-Root 模式，使得被监控系统开始正常运行.

下面对 HybridHP 框架的具体实现和关键技术进行详细阐述.

3.1 HybridHP 的关键技术之一：关键数据保护

为了尽量地压缩 HybridHP 的代码量，HybridHP 框架没有单独的管理域 Domain0 的概念，更没有像传统虚拟机监控器方案中自身实现的虚拟存储管理功能，HybridHP 和被监控系统处于相同的页表视图下，被监控系统甚至可以看到 HybridHP 的存在，如何实现对关键数据(如被监控系统的页表和代码所在页、需要保护的特定数据页等)的保护是需要解决的关键问题.

HybridHP 的监控能力主要体现在两个方面：特权指令的捕获和关键数据区域的保护.

对特权指令的实时捕获是 HybridHP 作为轻型内核完整性监控方案的基础. HybridHP 通过自身的初始化过程在 VMCS 的 HybridHP 执行控制区中对需要监控捕获的特权指令(如 CPUID 等)进行配置. 在被监控系统的运行过程中，由于被监控系统运行于 VMX Non-Root 模式，这些特权指令的运行将触发指令陷入(Trap)事件，被监控系统的运行将被打断，通过 VM Exit 的入口地址进入 HybridHP 的相应处理程序，运行模式也从 VMX Non-Root 切换到 VMX Root 模式，控制权转移至 HybridHP，从而实现对特权指令的捕获. HybridHP 在处理完指令陷入之后，通过 VM Entry 接口返回被监控系统.

对于关键数据区域的保护，涉及到 HybridHP 框架中对寄存器和异常等的监控事件类型的捕获，主要由以下技术提供保证：

(1)始终开启 CPU 的页保护机制. 开启页保护机制是 HybridHP 监控的基础，因为如果让操作系统具有去除页保护机制的能力，那么内核恶意模块就可以不通过页保护机制而更改任意的内存，那么也就没有了关键数据区域的概念. 在 X86 架构中，页保护机制由寄存器 CR0 的 WP 位控制. 在 HybridHP 的控制策略中，设置对 CR0 寄存器的访问控制，同时在 VMCS 的 HybridHP 执行控制区设置对 CR0 寄存器的保护. 在 HybridHP 的监控框架下，被监控系统的运行过程中如果出现修改 CR0 寄存器的操作，将触发 CRA(Control Register Accesses)异常，运行模式从 VMX Non-Root 切换到 VMX Root 模式，同时控制权转移到 HybridHP. HybridHP 通过对被监控系统的操作语义进行分析并根据控制策略进行判断，对于非法的 CR0 修改动作，HybridHP 禁止该动作的执行，从而使得被监控系统中的恶意模块没有禁止页保护机制的能力，即始终开启页保护机制(CR0.WP=1).

(2)在上述(1)的基础上，HybridHP 根据策略将需要保护的关键数据在内存中的页设置成在 VMX Non-Root 模式下只读. 一旦发生对这些数据的修改操作就会触发 pagefault 异常，通过模式转移 VM Exit 从 VMX Non-Root 模式切换为 VMX Root 模式，控制权交由 HybridHP，通过和策略信息比较，判断修改动作的合法性，如果合法，HybridHP 负责将该修改动作执行，否则，通过注入返回的方式，向被监控系统发送错误警告.

(3) HybridHP 将被监控系统的页表所在的页设为在 VMX Non-Root 模式下只读，那么恶意模块试图通过去掉页表中相应页的保护属性而修改内存的操作必然会触发 pagefault 异常，被 HybridHP 所捕获，处理方式与(2)类似.

从以上的技术可以看出，对关键数据的修改动作不会绕过 HybridHP 的监控处理，满足不可旁路的要求.

3.2 HybridHP 的关键技术之二：自我保护

基于 XEN 之类的虚拟机监控框架中，由于使用了影子页表[1]，虚拟机监控器以及管理域与被监控系统采用的是不同的页表，为此监控器部分的相关数据结构是不在被监控系统的内存视图之内的，所以监控程序不会受到被监控系统的破坏和干扰. 也就是说由于 XEN 中有自己的虚拟存储管理，所以可以做到监控器自身的数据对客户机完全不可见. HybridHP 以模块化的方式嵌入在被监控系统中运行，采用了与被监控系统相同的页表视图，没有对被监控系统完全隐藏物理内存的能力，为此 HybridHP 如何达到自我保护也是需要解决的关键问题.

HybridHP 框架通过将自己的代码页、数据页和策略所在页等数据设置为在 VMX Non-Root 模式下只读来起到自我保护. 也就是说，HybridHP 将

自身作为关键数据的一部分进行监控保护，那么即使自身暴露在被监控系统的内存视图中，利用 3.1 节中的关键数据保护技术，如果被监控系统中的恶意模块试图修改 HybridHP 的代码、数据或者策略信息，都会被 HybridHP 捕获到，HybridHP 可以否决所有这些非法的修改.

总的来说，HybridHP 的自我保护能力使得虽然自身的所有数据暴露于被监控系统的视图之内，但是被监控系统没有对其进行修改的权限，所以可以保证自身的安全.

3.3　HybridHP 的关键技术之三：策略更新

为了实现监控策略和机制的分离以及策略的更新，HybridHP 本身实现的只是监控的机制，监控策略需要由外部环境提供. 在传统的基于虚拟机的监控方案中，虚拟机监控器所采用的监控策略由管理域 Domain0 提供，为此策略的安全性由 Domain0 保证. 在 HybridHP 框架中，HybridHP 自身是一个可信计算基(TCB)，而被监控系统是不可信的，如果依赖被监控系统为其提供策略信息，被监控系统中的恶意模块可以修改策略信息或者任意封装恶意的策略并发送给 HybridHP，那么整个监控框架是不可信的. 同时，HybridHP 没有管理域 Domain0 的概念，为此需要解决 HybridHP 策略信息的来源问题，以及策略信息从被监控系统的用户层输入到 HybridHP 空间这样一条安全的策略输入路径的问题.

利用一套独立的可信平台作为策略中心，其中存放了为 HybridHP 所准备的各种策略信息，该策略中心作为 HybridHP 策略信息的来源保证了策略信息源头的安全性. 同时，采用数字签名加密认证的方法来解决上述的策略安全输入路径问题. 对从策略中心获得的 HybridHP 策略信息文件使用 Hash 算法(如 MD5、SHA 算法等)计算 Hash 值，即做“数字摘要”，再对数字摘要用签名私钥做非对称加密(如 RSA、ElGamal[①] 等)，即做“数字签名”. 之后将以上的签名和策略信息文件原文一起进行封装，被监控系统在用户层将这一封装的结果通过 VMX 系统调用 Hypercall 的形式向 HybridHP 发送，HybridHP 对接收的经过加密的策略信息进行验证. HybridHP 收到的数字签名结果，其中包括数字签名和策略信息文件原文. HybridHP 首先用发方公钥解密数字签名，导出数字摘要，并对策略信息文件原文做同样 Hash 算法得出一个新的数字摘要，并将这两个摘要进行结果比较，结果相同则签名得到验证，否则，说明策略在输入过程存在被恶意的模块修改的情况，HybridHP 判定输入的策略信息为无效，不予采用. 对于经过验证的策略信息文件，HybridHP 还需检查其是否为预定义的策略格式，如三元组(主体，动作，客体)等. 如果是正确的策略信息，那么将其加入策略库中，并进行相应的关键数据保护设置，否则忽略这些信息，这样可以确保 HybridHP 得到的策略一定是管理人员所希望的策略，整个框架如图 3 所示.

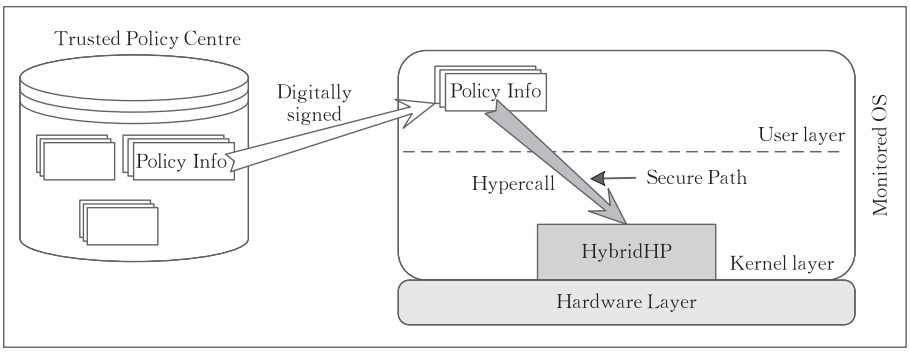


图 3　HybridHP 策略更新

由于 Hash 算法(MD5、SHA 等)存在安全性的问题，不可信的被监控系统可以生成一个与可能的策略文件具有相同“数字摘要”的恶意策略文件，并传递给 HybridHP，但其内容只有极小的可能性也是一段策略内容，而更多的是一段乱码，或者说不是预定义的格式，如三元组(主体，动作，客体)等，对此 HybridHP 仍能将其识别. 同时，在保证签名私钥安全不泄露的情况下，外部不可信的被监控系统无法将恶意的策略信息进行封装并发送给 HybridHP. 因此，恶意策略无法破坏 HybridHP 的完整性，保证了策略信息的机密性、不可修改性和不可伪造性.

HybridHP 只提供监控的机制，采用机制和策略相分离的方式，具体的策略由外部环境提供，这给具体的策略配置提供了便利. 在本文中，可以采用三

① ElGmal Encryption. http://en.wikipedia.org/wiki/ElGamal_encryption

元组(主体,动作,客体)的格式来定义策略,如表 1 所示.其中表头的第 2 行表示客体.

表 1 HybridHP 策略示例表

主体	动作					
	CR0	PageTable	IDT	GDT	LDT	Predefined CriticalData
可信模块	R	R	R/W	R/W	R/W	R/W
不可信模块	R	R	R	R	R	R
内核	R	R	R/W	R/W	R/W	R/W
HybridHP	R/W	R/W	R/W	R/W	R/W	R/W

HybridHP 策略主体可以分为可信模块、不可信模块、内核以及 HybridHP.可信模块主要包括得到第三方(工业界)认证的各种设备驱动程序,如显卡和网卡驱动等;不可信模块主要包括上层用户的各种第三方应用程序,这些程序模块未得到有效的认证,因而是不可信的,它们对于系统资源的访问是只读的权限;被监控系统的内核部分在可信的情况下对系统资源具有大部分的权限;在 HybridHP 框架中,HybridHP 自身作为可信基,对系统资源具有所有的访问权限.

3.4 HybridHP 的关键技术之四:可信启动

HybridHP 保证了在其启动以后的时间内监控系统的安全性,但是在 HybridHP 启动之前的阶段无法保证系统的安全性.被监控系统中的恶意模块可以在 HybridHP 启动之前破坏 HybridHP,或者禁止 HybridHP 的启动.

在监控框架中,HybridHP 启动之前的安全保护由基于 TPM① 的可信启动来负责,如图 4 所示.

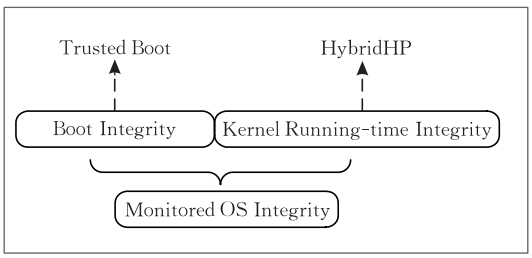


图 4 系统完整性保护

在 HybridHP 原型中整合了 tboot② 软件,通过监控启动过程中所有关键数据的完整性,能够有效地防止系统的核心组件被替换,从而保证系统启动过程的完整性,这也是 HybridHP 监控服务的前提.

4 HybridHP 的验证

HybridHP 代码量控制在 8K SLOC.本节使用 Isabelle 形式化辅助证明工具阐述 HybridHP 框架的隔离性、安全性和监控能力的验证,并进行安全分析.

验证部分包括:HybridHP 对"去掉页保护机制"、"修改页表"和"修改关键数据"动作的不可旁路性,HybridHP 自我保护机制,以及 HybridHP 策略更新.

4.1 Isabelle 的符号表示

这一小节描述验证过程用到的符号表示方法.采用 Isabelle 系统的符号表示,这种符号表示方法与传统数理逻辑中的方式基本相同.Isabelle 可以实现对计算系统的形式化描述和验证,采用的 Isabelle/HOL 是 Isabelle 中对高阶逻辑(Higher Order Logic,HOL)的支持和实现.

HOL 作为一种类型化的逻辑系统,其类型系统与函数式编程语言(如 ML、Haskell 等)是类似的.类型变量可以用$'a$,$'b$ 等符号进行表示.对于类型项,可以用如 $x::'a$ 定义方式.Isabelle 对类型支持构造子操作,例如 *nat list* 用于定义由自然数组成的列表,*int set* 定义由整形变量组成的集合.

在 HybridHP 的描述和验证的过程中,对于新类型的构造主要采用 3 种定义方式:datatype、types 和 record. datatype 实现对代数数据类型的构造,例如,对于寄存器类型定义为

datatype *reg*=CR0|CR1|CR2|CR3|CR4.

types 表示类型的简化记号,如 types *nat_set*=*nat set*,定义了新的类型 *nat_set*,它表示自然数组成的集合类型.

record 用于构造带名称的元组类型,例如,对于 3.3 节描述的策略信息定义为

record *policy*=*subject*::*string*,
cond::*string set*,
object::*string*.

新类型 *policy* 表示策略信息,它含有 3 个成员:*subject* 表示策略信息的动作主体,*cond* 表示需要对对象设置的操作语义信息,*object* 表示策略信息中的具体客体对象.对于 record,引用成员信息可以表达为如 *subject policy*,表示引用 *policy* 中的 *subject* 成员.假设 *policy* 拥有值(|*subject*=HybridHP,*cond*=READ_ONLY,*object*=CR0|),

① Trusted Computing Group. http://www.trustedcomputinggroup.org/developers/trusted_platform_module/specifications

② Trusted Boot. http://tboot.sourceforge.net

更新操作可以表达为如 *policy*(|*cond* := READ_WRITE|)，表示将 *policy* 中的 *cond* 成员修改成 READ_WRITE，但 *subject* 和 *object* 成员保持不变.

对于函数运算，使用“⇒”符号表示函数定义域和值域的映射对应关系. 函数更新操作使用如 $g(x:=y)$方式来表达. 函数在集合上的值域运算使用如 $g'Z \equiv \{y \mid \exists x \in Z.\ y = gx\}$表达，表示函数 g 以集合 Z 为定义域的值域.

4.2　HybridHP 对象模型

HybridHP 以模块的方式在被监控系统的内核空间中运行，对被监控系统的行为进行监视，根据预定义的策略信息对系统中的操作进行判断. 被监控系统中其它模块不直接与 HybridHP 交互，并不需要知道 HybridHP 的存在. 系统中各种动作的执行效果，可以看成是对系统状态的改变或者迁移，HybridHP 在此过程中起到监控和决定的作用，如图 5 所示.

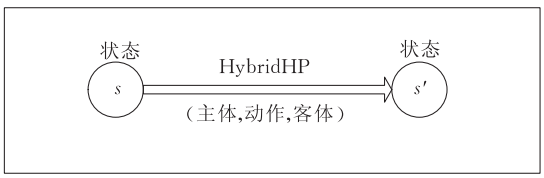


图 5　状态转换图

文中使用 Isabelle 验证工具对 HybridHP 的安全性进行验证，需要对 HybridHP 建立模型. 以对象模型的方式建立 HybridHP 的语义模型，将操作的主体和客体作为对象来看待，同时将系统状态之间的转换看成是操作的主客体对象以及 HybridHP 相互作用的结果. 在这样一个场景下，在下面的章节中使用 Isabelle 对 HybridHP 对象模型进行符号化表示，并在此基础上阐述验证 HybridHP 对“去掉页保护机制”、“修改页表”、和“修改关键数据”动作的不可旁路性，HybridHP 自我保护机制，以及 HybridHP 策略更新行为的正确性定理.

4.3　HybridHP 系统模型形式化描述

系统执行进程标识：types *entity_id* = *nat*.

系统的执行进程使用 record 定义：

record *process* = *pid* :: *entity_id*,
pcb :: *pcb_struct*,

其中 *pcb_struct* 为进程控制块 PCB 结构.

系统的状态定义为

record *state* = *mem* :: *entity_id* ⇒ *process*,
regs :: *reg set*,
pagetable :: *pageitem set*,
next_id :: *entity_id*,
curr_mode :: *string*,

其中，*mem* 表示系统中内存视图，通过进程的标识可以找到所有的进程个体；*regs* 表示系统中的寄存器集合；*pagetable* 表示系统的页表，是由页表项 *pageitem* 组成的集合，*pageitem* 是虚拟地址 *va* 和物理地址 *pa* 组成的 record；*next_id* 表示系统中可用的进程标识；*curr_mode* 指明系统目前的模式，取值如 *VMX_Non_Root* 和 *VMX_Root*.

针对要验证的动作类型定义如下：

datatype *action* = Write CR0
| Write PageTable
| Write CriticalData
| Write HybridHPData
| PolicyUpdate *policy_info*
| Null_Action.

action 包含对 CR0、页表、关键数据和 HybridHP 数据信息的修改动作、策略更新动作以及空操作.

HybridHP 的策略集合定义为 *PolicySet* :: *policy set*.

系统所有的状态集合定义为 *S* :: *state set*.

获得动作主体的语义函数 *SubjectofAction*：*action* ⇒ *subject*.

获得动作客体的语义函数 *ObjectofAction*：*action* ⇒ *object*.

获得动作的操作语义函数 *CondofAction*：*action* ⇒ *cond*.

HybridHP 策略监控判定语义定义：

policy_judge :: *action* ⇒ *policy set* ⇒ *bool*,
policy_judge a ps ≡
(|*subject* = *SubjectofAction a*,
cond = *CondofAction a*,
object = *ObjectofAction a*|) ∈ *ps*.

policy_judge 判断动作是否符合预定的策略.

执行单个动作引起的系统状态转化语义函数 *step*：*state* ⇒ *action* ⇒ *state*.

4.4　形式化验证

本节从攻击的角度来说明修改系统属性的操作，阐述 HybridHP 如何防范这些修改方式，并验证其正确性，同时对策略更新进行验证.

定理 1.　CR0 保护.

Theorem 1.　$\forall s \in S.\ [|\ action = \text{Write CR0};\ (policy\_judge\ action\ PolicySet) = \text{False};\ s' = step$

s action|] → CR0 (*regs s'*) = CR0 (*regs s*).

第 1 种攻击方式试图去掉系统的页保护机制(CR0.WP 位),从而可以修改任意的内存页.由于 HybridHP 对 CR0 进行保护,这种对 CR0 的写操作立即引起 CRA 异常,HybridHP 通过 VMX 的控制结构 VMCS 读取触发原因,此时 CR2 寄存器的值指向写数据错误的地址.通过 CR3、task_struct 等获得执行主体的信息,即对操作语义进行解析,并根据策略信息判断出这种操作的违法性,然后使用注入返回的方式对被监控系统进行错误警告.执行该操作前后,系统状态中 CR0 保持不变.定理 1 说明在系统的运行过程中,如果当前动作为修改 CR0(Write CR0),而在策略判断中,该动作的主体没有权限修改 CR0,此时策略检测(*policy_judge action PolicySet*)无法通过(False),那么系统在该动作后状态(s')环境中的 CR0 寄存器保持不变.如此可以保证 HybridHP 监视系统中所有对 CR0 的操作,使得被监控系统没有禁止页保护机制的能力,即始终开启页保护机制.

定理 2. 页表保护.

Theorem2. $\forall s \in S.$ [|*action* = Write PageTable; (*policy_judge action PolicySet*) = False; s' = *step s action*|] → *pagetable s'* = *pagetable s*.

这种攻击方式试图通过修改页表中的页表项的只读属性,从而对系统的内存页进行修改.对于这种攻击方式,没有 HybridHP 监控的系统中很容易受到破坏.HybridHP 框架根据策略信息对系统中的页表所在的页进行保护,因此修改页表的操作将引起 pagefault 异常,由 HybridHP 在 VMX Root 模式下进行处理,此时 CR2 寄存器的值指向页表所在的页.定理 2 说明 HybridHP 监控对页表的修改动作,如果当前动作为修改页表(Write PageTable),而在策略判断中,该动作的主体没有权限修改页表,此时策略检测(*policy_judge action PolicySet*)也无法通过(False),那么系统在该动作后状态(s')环境中的页表项保持不变,从而保证了对页表的保护.

定理 3. 关键数据保护.

Theorem 3. $\forall s \in S.$ [|*action* = Write CriticalData; (*policy_judge action PolicySet*) = False; s' = *step s action*|] → *mem s'* = *mem s*.

Proof apply (rule Theorem1, rule Theorem2)

第 3 种攻击方式对内存中的关键数据页进行修改,而这些页受到 HybridHP 的策略保护.定理 1 保证了外部攻击无法去除系统的页保护机制,定理 2 保证了外部攻击无法修改内存页在页表中的保护属性,因此这第 3 种攻击方式必然会触发 HybridHP 对 pagefault 异常的捕获,无法绕过 HybridHP 的监控.为此,定理 3 的证明过程需要借助定理 1 和定理 2.

定理 1、定理 2、定理 3 阐述了 HybridHP 监控的不可旁路性.

定理 4. 自我保护.

Theorem4. $\forall s \in S.$ [|*action* = Write HybridHP-Data; (*policy_judge action PolicySet*) = False; s' = *step s action*; *curr_mode s* = VMX_Non_Root|] → (*mem s'*) HybridHP = (*mem s*) HybridHP.

Proof apply (*case_tac* (*curr_mode s*) and rule Theorem1, rule Theorem2, rule Theorem3).

最后 1 种攻击方式假设通过页表视图和系统模块信息找到 HybridHP 代码和数据以及策略信息所在的内存页,对 HybridHP 进行破坏.HybridHP 框架的自我保护能力是通过将自身的代码段、数据段以及策略信息的内存页设置成在 VMX Non-Root 模式下只读来达到的.同时,系统中的恶意模块运行在 VMX Non-Root 模式,因此这种修改操作必然受到 HybridHP 的干预.为此,定理 4 说明,如果当前状态(s)的运行模式为受限模式(VMX_Non_Root),策略检测(*policy_judge action PolicySet*)对修改 HybridHP 自身信息的非法动作判断为没有相应的权限(False),那么该非法动作无法修改 HybridHP 的信息,即 HybridHP 的数据信息保持不变.我们可以看出定理 4 的证明需要定理 1、定理 2、定理 3 的辅助,并对系统的当前的状态模式采用分情况的证明策略 *case_tac* 来拆解.

定理 5. HybridHP 策略更新.

Theorem5.

$$\begin{aligned}&\forall s \in S.\ [|action = \text{PolicyUpdate } policy\_info|] \rightarrow\\ &\quad (validate\_policy(policy\_info) = \text{True} \wedge\\ &\quad (policy\_judge\ action\ PolicySet) = \text{True} \rightarrow\\ &\qquad PolicySet = PolicySet \cup policy\_info) \wedge\\ &\quad (validate\_policy(policy\_info) = \text{False} \vee\\ &\qquad (policy\_judge\ action\ PolicySet) = \text{False} \rightarrow\\ &\qquad s' = step\ s\ Null\_Action \wedge\\ &\qquad PolicySet \ominus s = PolicySet \ominus s').\end{aligned}$$

HybridHP 采用数字签名加密认证的方式进行策略更新,主要是为了避免外部环境的不可信对策略信息安全性的影响.PolicyUpdate 是策略更新动作,策略信息定义为 *policy_info* :: *policy set*.本文对主要的加密算法(如 MD5、SHA 算法以及 RSA、ElGamal 等)和数字签名过程进行了 Isabelle 建模验证,并将其封装在库中,对外提供 *validate_policy*

接口进行解码验证和策略信息的预定义格式(如三元组)识别. 如果策略信息通过验证,那么 HybridHP 认为这些策略信息是可取的,并加入系统的策略集合中;如果策略信息没有通过验证($validate_policy(policy_info)=$False),或者不允许恶意的策略更新动作(($policy\_judge\ action\ PolicySet$)=False),那么此更新动作无法执行,以空操作 $Null\_Action$ 来表示. 同时,策略信息保持不变,其中 $PolicySet\ \Theta s$ 表示状态 s 下的策略集. 定理 5 阐述了 3.3 节中描述的在被监控系统不可信的环境下,策略更新的安全性以及恶意策略不会破坏 HybridHP 的完整性.

上述的定理都依赖于策略信息的安全性,即其正确性和所构造的 $PolicySet$ 有关. $PolicySet$ 可以看成是 3.3 节中从外部环境输入的策略信息,而策略信息源头的安全性和策略更新的安全路径保证了输入到 HybridHP 中的策略信息的合法性,因此这些定理的正确性得到了保证.

Isabelle 的验证环境配置为 Studio XPS 9100 台式电脑,2.8GHz Intel i7 930 处理器,3GB 内存,操作系统采用 openSUSE Desktop 11.3 版本,Isabelle 采用 Isabelle2009-2_bundle_x86-linux 版本. 整个 Isabelle 验证工程代码量大概在 15K SLOC 左右,完整的验证耗时 40min 左右.

Isabelle 的验证结果如图 6 所示."No subgoals"说明 Isabelle 验证逻辑完整,不存在任何未证明的子目标.

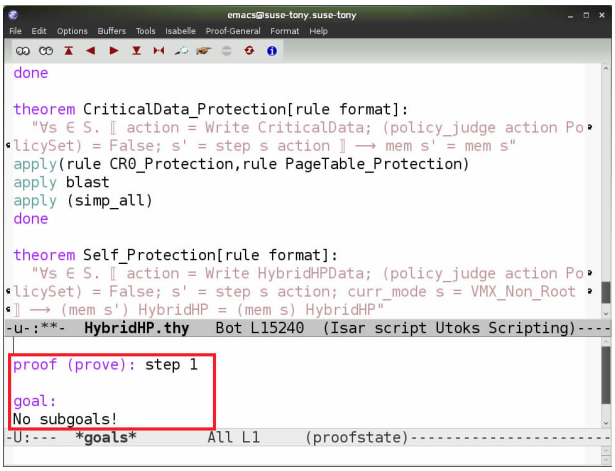


图 6　Isabelle 验证结果图

5　实验和系统性能评估

HybridHP 对操作系统的性能影响及其监控的有效性是实验关注的重点.

在传统的虚拟化监控方案中,性能评估往往关注于监控机制本身所带来的性能损失(将这部分损失的性能定量记为 ε),而忽视被监控系统所获得的真实物理性能(定量记为 α). 我们认为在传统的虚拟化监控框架下,需要引入考虑虚拟框架(包括管理域如 Domain0 以及虚拟机如 XEN)耗用的性能(定量记为 β). 因此,整个监控框架下整体的性能(定量记为 θ)应该是被监控系统所获得的物理性能(α)、虚拟框架耗用的性能(β)和监控机制导致的损失性能(ε)这 3 个部分之和,即 $\theta=\alpha+\beta+\varepsilon$,取 θ 的值为 100%.

被监控系统所得到的真实性能是关注的重点,所以在性能测试中,加入了这一个评估指标. 主要从以下两个角度进行性能评估:(1)被监控系统所获得的真实性能(α);(2)监控本身所带来的性能损失(ε).

为了评估 HybridHP 的性能,选择了几个 Linux 的基准测试程序,包括 UnixBench① 以及其它的一些真实应用软件,通过和传统的基于 XEN 的监控框架以及 SIM 的监控方案进行对比来说明 HybridHP 的性能. 测试平台是 Studio XPS 9100 台式电脑,2.8GHz Intel i7 930 处理器,3GB 内存. 被监控系统的各种软件配置信息如表 2 所示.

表 2　被监控系统软件配置信息

名称	版本	配置
openSUSE Desktop	11.3	Standard Installation
UnixBench	4.1.0	Make run
Kernel build	2.6.34.12	Make world
Bzip2	1.0.4	tar −zxvf ⟨kernel file⟩

此外,为了有效地进行性能对比,本实验还将被监控系统作为客户机,部署到传统的基于 XEN 的以及 SIM 的监控框架中. 其中管理域 Domain0 采用半虚拟化(Paravirtualization)配置,XEN 采用标准配置,软件配置信息如表 3 所示.

表 3　Domain0 和 XEN 软件配置信息

名称	版本	配置
Domain 0	OpenSUSE 11.3	Standard Paravirtualization
XEN	3.4	Standard Installation

图 7 说明了被监控系统在 3 种框架下所获得的性能(α)对比,其中"解压文件"操作是面向 CPU 计算的,"编译"是面向 I/O 操作的,而 UnixBench 是

① UnixBench. http://ftp.tux.org/pub/benchmarks/System/unixbench

一个综合测试.与传统的基于 XEN 的监控方案和 SIM 不同,HybridHP 没有管理域 Domain0 的存在,因此在这 3 个对比测试中,采用 HybridHP 框架的被监控系统所获得性能均接近于物理主机性能.特别是在"编译"测试中,由于 HybridHP 不对被监控系统的 I/O 操作进行干预,系统的 I/O 性能基本没有影响.当然,HybridHP 对于系统页表操作的影响,和 XEN 中影子页表操作对系统的性能影响类似,主要的时间开销包括:模式转换产生的上下文切换、策略搜索时间、HybridHP 对异常处理的时间以及页表切换(CR3)的时间.

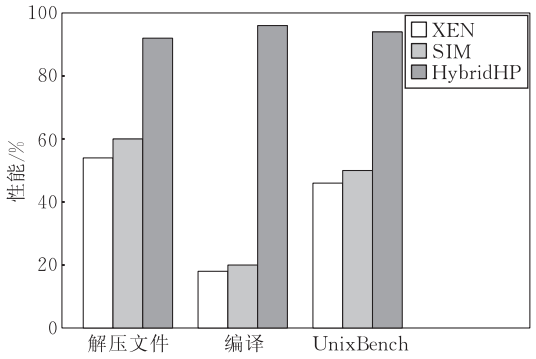


图 7 被监控系统所获得的性能(α)对比图

表 4 说明了以创建进程操作为例,3 种框架下监控机制本身所带来的性能损失(ε).可以看出,监控性能损失接近并略低于 SIM 的性能损失,HybridHP 对被监控系统的性能影响控制在很小的范围,满足 2.1 节第 5 点对 HybridHP 的性能要求,这也说明 HybridHP 是一种保证内核完整性的轻型方案.

表 4 创建进程操作的监控性能损失(ε)对比

监控类型	平均时间/μs	相对性能损失/%
裸机	3.487	NULL
XEN	28.039	704.1
SIM	3.967	13.7
HybridHP	3.853	10.5

为了验证 HybridHP 监控能力的有效性,我们选择 4 种不同类型的 rootkit 攻击方式[14]:Mood-nt、adore-ng、Synapsys 和 SucKIT2,并对它们的攻击目标进行了配置,使其适应 HybridHP 监控框架.第 1 种攻击方式试图通过写 CR0 寄存器来设置 WP 为 0,从而去掉页保护机制;第 2 种攻击方式试图修改系统的页表中某一项的只读权限,从而可以任意修改内存页;第 3 种攻击方式试图修改系统调用表,模拟对系统关键数据的修改;第 4 种攻击方式查找 HybridHP 所在的页,试图修改 HybridHP 的代码和数据页.这 4 种攻击代表了公认系统漏洞数据库(NVD①)目前典型的内核攻击方式.实现结果表明,HybridHP 可以有效地监控和阻止这 4 种攻击方式.

6 结束语

本文提出了一种用于监控内核完整性的轻型方案 HybridHP,它以模块的方式在被监控系统的内核空间中实施内核完整性监控服务.HybridHP 的功能由 4 个关键技术保证:关键数据保护、自我保护、策略更新和可信启动.关键数据保护技术对由策略信息设定的关键数据实施保护,被监控系统运行于 VMX Non-Root 模式,HybridHP 运行于 VMX Root 模式,保证了 HybridHP 监控的不可旁路性.HybridHP 和被监控系统拥有相同的页表视图,便于获得被监控系统的操作语义,提高监控处理的效率.但 HybridHP 的代码和数据信息暴露于被监控系统的内存视图下,为此加入了 HybridHP 的自我保护功能,防止被监控系统中恶意模块对其进行破坏.HybridHP 采用监控机制和策略分离的思想,本身只提供监控能力,通过策略更新的方式支持策略信息的动态修改,考虑到被监控系统的不可信问题,使用数字签名加密的技术保证策略更新路径的安全性以及策略信息的机密性、不可修改性和不可伪造性.可信启动技术提供在 HybridHP 对被监控系统实施监控服务之前的安全性,保证被监控系统启动过程中不会对 HybridHP 进行破坏.实现了 HybridHP 的原型,并使用 Isabelle 对其正确性进行了证明.同时,本文对 HybridHP 进行了攻击测试和系统性能评估,结果表明 HybridHP 能有效地监控针对内核完整性的攻击,同时对被监控系统的性能影响控制在很小的范围.

接下来的工作计划将 HybridHP 在多核处理器平台进行改进和实现,使得 HybridHP 运行在独立的处理器核上,加快监控处理的速度.HybridHP 提供了监控内核完整性的轻型方案,如何和各种具体的安全策略模型(如 BLP、Lattice 和 Biba 完整性模型等)结合以及如何构建更加有效的安全策略以保证尽可能多的攻击动作和攻击动作序列被检测也是将来研究的一个重要方向.

① National Vulnerability Database. http://nvd.nist.gov/

致　谢　本文作者感谢所有本文的匿名审稿者，感谢您们对本文提出宝贵的意见！

参　考　文　献

[1] Barham P, Dragovic B, Fraser K, Hand S, Harris T L, Ho A, Neugebauer R, Pratt I, Warfield A. XEN and the art of virtualization//Proceedings of the 19th ACM Symposium on Operating Systems Principles(SOSP'03). New York, 2003: 164-177

[2] Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. KVM: The Linux virtual machine monitor//Proceedings of the 2007 Ottawa Linux Symposium. Ottawa, 2007: 225-230

[3] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection//Proceedings of the 10th Network and Distributed System Security Symposium (NDSS'03). San Diego, 2003: 191-206

[4] Grizzard J. Towards self-healing systems: Re-establishing trust in compromised systems[Ph. D. dissertation]. Georgia Institute of Technology, Atlanta, Georgia, 2006

[5] Jiang X, Wang X, Xu D. Stealthy malware detection through VMM-based "Out-Of-the-Box" semantic view reconstruction//Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07). Alexandria, VA, 2007: 128-138

[6] Lanzi A, Sharif M, Lee W. K-Tracer: A system for extracting kernel malware behavior//Proceedings of the 16th Network and Distributed System Security Symposium (NDSS'09). San Diego, 2009

[7] Payne B D, Carbone M, Sharif M I, Lee W. Lares: An architecture for secure active monitoring using virtualization//Proceedings of the 29th IEEE Symposium on Security and Privacy(S&P'08). Oakland, California, 2008: 233-247

[8] Riley R, Jiang X, Xu D. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing//Proceedings of the 11th Recent Advances in Intrusion Detection (RAID'08). Boston, MA, 2008: 1-20

[9] Seshadri A, Luk M, Qu N, Perrig A. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes//Proceedings of the 21st ACM Symposium on Operating Systems Principles(SOSP'07). Stevenson, WA, 2007: 335-350

[10] Wang Z, Jiang X, Cui W, Ning P. Countering kernel rootkits with lightweight hook protection//Proceedings of the 16th ACM Conference on Computer and Communications Security(CCS'09). Chicago, IL, 2009: 545-554

[11] Yin H, Liang Z, Song D. HookFinder: Identifying and understanding malware hooking behaviors//Proceedings of the 16th Network and Distributed System Security Symposium (NDSS 2008). San Diego, 2008

[12] Sharif M, Lee W, Cui W, Lanzi A. Secure In-VM monitoring using hardware virtualization//Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09). Chicago, IL, 2009: 477-487

[13] Nipkow T, Paulson L C. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Berlin: Springer, 2002

[14] Petroni N L. Property-based integrity monitoring of operating system kernels[Ph. D. dissertation]. University of Maryland, College Park, 2008

QIAN Zhen-Jiang, born in 1982, Ph. D. candidate, lecturer. His current research interests include operating system security, formal verification and embedded system.

LIU Wei, born in 1986, M. S. candidate. His current research interests include virtual machine surveillance and formal verification.

HUANG Hao, born in 1957, Ph. D., professor, Ph. D. supervisor. His current research interests include system software and information security.

Background

This work is mainly supported by the National High Technology Research and Development Program (863 Program) of China under grant No. 2007AA01Z409, the Jiangsu Provincial Natural Science Foundation of Science and Technology Support Program under grant No. BE2008124, and the "Six Talents Peak" High-Level Personnel Project of Jiangsu Province under grant No. 2011-DZXX-035. They all aim to develop a secure and trusted microkernel operating system in which novel security technologies and virtual monitoring are researched, system design are formally specified and some of the security properties are verified formally or informally. In past years, the group has done a lot of related

works including a trusted boot revolution, an operating system integrity surveillance prototype based on VMM (OSISS), an operating system object semantics model (OSOSM) and its partial formal specifications and verification. This paper reports the authors' works on how to design and implement lifetime kernel integrity surveillance for OSOSM.

This paper aims to resolve how to provide lifetime kernel integrity surveillance using virtual technology, but without loss of performance of monitored OS. In this paper, the authors explain the existing problems of the traditional virtual machine monitoring approaches, such as system performance loss, trusted computing base (TCB) expansion of the monitoring programs, and difficulty of obtaining the operational semantics of the monitored system. This paper proposes and implements HybridHP, a lightweight approach to provide lifetime kernel integrity surveillance, which resolves the problems above. To the best of our knowledge, HybridHP is the first system that merges the management domain (e.g. Domain0) and virtual machine monitoring functionality into the monitored system, and provides monitoring services in the way of embedded module. The memory view of HybridHP is consistent with the view of the monitored system, so it is easy to obtain the operational semantics of the monitored system. HybridHP does not intervene in the I/O operation of the monitored system, and the performance loss is little. HybridHP has the small amount of codes with 8K SLOC, and is easy to be verified formally. In this paper, the correctness of HybridHP is verified in strict formal logic with the theorem prover Isabelle/HOL. The authors also introduce the systemic experiments and performance evaluation for HybridHP. HybridHP can effectively monitor the kernel integrity, and has good overall system performance.