基于虚拟化平台 Xen 的内核安全监控方案①

陈祝红[1,2], 崔超远[2], 王儒敬[2], 周继冬[1,2]

[1](中国科学技术大学 自动化系, 合肥 230027)
[2](中科院合肥物质科学研究院 智能机械研究所, 合肥 230031)

摘 要: 虚拟化技术作为云计算的基础架构, 其安全性随着云计算的发展越来越受到人们的关注. 提出一种针对虚拟机系统内核攻击的入侵检测方案, 借助 Xen 提供的虚拟化平台, 来获取虚拟机系统内核运行情况, 从而达到监视和防止内核被攻击的目的. 该方案可以有效地防御来自动态修改内核代码和内核不变数据结构一类的攻击.

关键词: Xen; VMM; 内核攻击; Hypercall

Kernel Intrusion Detection Method Based on Xen Hypervisor

CHEN Zhu-Hong[1,2], CUI Chao-Yuan[2], WANG Ru-Jing[2], ZHOU Ji-Dong[1,2]

[1](School of Information Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Institute of Intelligent Machine, Chinese Academy of Sciences, Hefei 230031, China)

Abstract: As the foundation of cloud computing, virtualization technology's security problems are gained more and more attention of the specialists with the development of cloud computing. This paper presents a virtual machine system kernel intrusion detection system, which use the introspection technology provided by Xen hypervisor to get internal states of kernel of the virtual machine. To achieve the goal of monitoring the kernel and preventing it from being compromised. This system can effectively defend the case of attack that dynamically modifies the kernel code and kernel unchanged data structure.

Key words: Xen; VMM; Kernel attacks; Hypercall

1 引言

虚拟化技术是计算机资源的逻辑表示方法, 它的实现形式是在系统中加入一个中间虚拟层, 使上层系统可以运行在虚拟层上, 获得与真实环境相同或相似的功能. 系统虚拟化是虚拟化技术中的一种, 其抽象粒度是整个计算机. 云计算采用虚拟化技术整合资源, 形成一个大型虚拟化资源池, 然后按照用户的需求动态调整资源, 为每个用户配置一个独立的虚拟计算环境. 云计算的出现和推广, 使虚拟化技术得到了广泛应用, 同时也引发了人们对其安全问题的关注[1].

在所有针对虚拟机的攻击中, 破坏系统内核是最为致命的一种攻击方式, 它利用内核漏洞或内核执行机制进行攻击[2]. 由于内核是整个系统的可信基础, 内核一旦被攻击或控制, 将会导致整个系统瘫痪. 针对虚拟机内核的安全问题, 不少学者提出了具体的解决方案. Tal Garfinkel 和 Mendel Rosenblum 在 2003 年首先提出一种从虚拟机外部进行入侵检测的方案[3],构筑了检测系统 Livewire. 其提出的入侵检测框架与被监视系统高度隔离, 为后续基于虚拟化安全检测技术提供了新的思路. 由于该系统是建立在商业闭源的 VMWare 之上, 所以不利于监控技术的推广. 且其中制定的入侵检测规则只针对当时的内核攻击技术, 还需改进.

另一类方法, 如 XenAccess 是在被监视虚拟机内部安插写保护安全 hooks, 在分析前挂起虚拟机事件, 实现主动监视[4]. 从虚拟机内部进行入侵检测的主要优点是可以直接获取能够反映系统运行状态的语义信息, 便于分析判断. 但是由于这类方法本身的局限性, 系统一旦被攻击者控制时, 它们将不能向外部提供任何信息.

N. L. Petroni, T. Fraser 等人提出了一种基于内核控制流的入侵检测方案, 它通过定时获取内核内存的

① 收稿时间:2012-12-31;收到修改稿时间:2013-01-28

快照，并进行分析，以确定不变的内存区是否发生变化. 该发放属于一种消极监控，它只能在攻击发生之后检测，不能及时防止攻击，而且对于快速攻击，该检测方案是无能为力的.

本文提出一种从虚拟机外部进行内核攻击检测的方案. 该方案基于 Xen 虚拟化技术，用于检测针对 Linux 系统内核的攻击. 检测系统运行于具有高优先级的虚拟机 Dom0 上，通过 libxenctrl 提供的 C 语言接口来调用 Xen 的超级调用 Hypercall[6]，实现对监控虚拟机内核空间的监控，以便有效地检测虚拟机的内核攻击，防止入侵发生.

2 Xen虚拟化和内核攻击

2.1 Xen 虚拟化技术

源自英国剑桥大学开源工具 Xen 在 x86、x86_64、PowerPC 和其他 CPU 架构上都能提供强大的系统虚拟化特性. 它采用类虚拟化技术(Para-Virtualization)，直接运行于硬件平台之上，可以使一台物理机上同时运行多个虚拟操作系统[7]. Xen 的系统结构如图 1 所示. Xen 支持的虚拟机可分为两种: Dom0 和 DomU. 它们都运行在负责虚拟机托管和管理的 Xen Hypervisor 之上. Dom0 具有高于 DomU 的权限，可直接访问硬件. 由于 Xen 没有提供用户交互接口，整个系统都是通过 Dom0 来管理的，所以每一台 Xen 服务器至少运行着一个唯一的 Dom0. DomU 是分配给一般用户的虚拟机，不具有直接访问硬件的权限，它的资源分配、生命周期通过 Dom0 来管理，本文所述被监控虚拟机是指运行 DomU 的客户机.

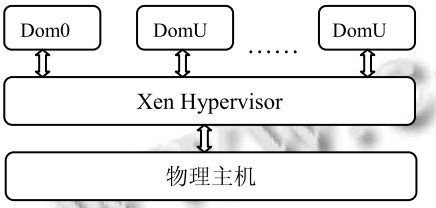

图 1 Xen 系统结构

2.2 内核攻击

内核是整个操作系统的核心，对操作系统的运行起着决定性的作用. 常用的操作系统，如 Windows、Linux 的逻辑地址空间可以分为两个部分: 进程空间和内核空间. 系统内核运行在内核空间，具有高优先级. 用户程序运行在进程空间，优先级较低.

内核是硬件和软件之间的一个中间层，它充当着底层硬件的驱动程序，将计算机硬件抽象到一个高层次，为上层程序提供简单、统一的访问接口. 同时，它还起着资源管理的作用，负责将可用共享资源，如 CPU 时间、磁盘空间、网络连接等分配到各个系统进程，保证系统的完整正常运行.

常见的内核攻击方法是修改内核空间中内核代码或运行过程中的不变数据结构(指从系统启动时被初始化到系统关机之间内容都不会被改变的数据，如系统调用列表 sys_call_table 等). 例如 Rootkit 就是一项专门针对内核的攻击技术[5]，它通过修改这些数据结构来达到隐藏其他程序进程、文件和网络通信的效果.

针对上述内核攻击的检测和防御，本文提出了基于 Xen 虚拟化技术的入侵检测框架，该框架下的虚拟机均为 Linux 操作系统. 下面详细叙述该框架的结构、功能、实现方式和性能评估.

3 检测框架

内部检测工具运行在被监控系统的内部，能够直接获取被监控系统的诸多信息，拥有高可见性，便于判断是否受到攻击. 但是当系统被成功入侵时，内部监控工具同样会暴露在攻击者的控制之下，不能发挥正常的检测功能. 而本文提出的外部监控系统在保持对被监控系统的高可见性的同时，保持着与被监控系统很好的隔离度. 即使在被监控虚拟机被攻击者成功入侵的情况下，依然能够免于受到攻击的影响，保持对虚拟机有效的监控.

3.1 前提假定

为简化模型，本文规定了一些前提假设. 首先，假定 Xen Hypervisor 是安全的. 虚拟机监视器拥有最高的优先级，整个框架的可信机制都是建立于它之上的. 其次，假定 Dom0 是安全的. Dom0 具有最高的优先权，所有的虚拟机都通过它来管理. 最后，假定监控系统是安全的. 因为监控系统是运行在 Dom0 上. 除此之外，假定虚拟机系统内核可执行文件是安全的. 为确保安全本文保留该内核文件安全版本的 md5 值，并在每次内核启动之前计算 md5 值. 当两值相等时说明可执行文件是安全的.

3.2 框架结构

基于 Xen 的内核攻击检测框架通过 Hypercall 获得被监控系统的内存、CPU、虚拟设备信息等. 拥有对被监控系统的高可见性的同时还保持与被监控系统高度隔离. 对于不同的被监控虚拟机系统，需要根据系统内核代码、数据等在内存中的分布结构知识，获得被监控

虚拟机的内存分布信息. 具体的系统框架如图 2 所示.

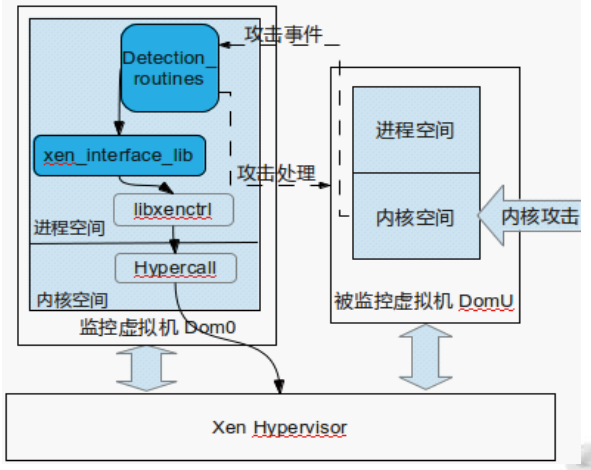

图 2　基于 Xen 的内核攻击检测框架

图 2 中左侧方框 Dom0 内的 xen_interfece_lib 和 Detection routines 是本文实现的主要功能模块, 分别为系统调用接口库和监控程序. Hypercall 是 Xen 提供给虚拟机内核使用的调用接口, 在虚拟机启动时, 由 Xen 直接映射到系统的内核空间, 内核代码可直接调用. libxenctrl 是由 Xen 提供的 C 函数库, 可对 Hypercall 进行调用. xen_interfece_lib 用来访问 libxenctrl 与 Detection routines 进行交互.

xen_interface_lib 是对 libxenctrl 接口的进一步抽象, 使得监控程序无需与底层复杂的 API 交互,从而简化上层监控程序. 并且提供一些 libxenctrl 没有提供的底层接口, 例如修改客户机页表等. 它实现的接口包括: 将监控虚拟机内核的代码段映射到本进程空间和设置监听事件. Detection_routines 运行于 Dom0 的进程空间, 通过调用 xen_interface_lib 得到虚拟机相关信息, 再根据获得的信息对客户机运行状态进行判断. 根据不同的监控规则对内核代码段和不变数据结构等不同内存段进行监视. 当发现有恶意攻击时, 立即进行相应的处理, 例如重启或暂停虚拟机.

本文提出的检测系统对不同的内存段采取的监控规则如下. ①代码段监控: 由于内核代码段所在内存段的属性为只读的, 所以如有攻击者试图修改内核代码时, 会出发页面异常. 监控程序据此判断是否有攻击. 为防止代码段被修改, 监控程序会周期性地对客户机相应的内存段进行完整性判断. 一旦发现存在修改, 立即进行处理. ②不变的数据结构监控: 由于数据结构对应的内存段属性为可读写, 故恶意程序可直接对其修改, 而不触发任何异常. 为对其进行监控, 监控程序通过 Hypercall 调用, 将其对应的内存段设为只读, 捕获非法修改触发的页面异常也会被监控程序捕获.

3.3 系统实现

根据图 2 所述的入侵检测框架, 结合虚拟机环境的运行特点, 系统分为以下几个步骤来实现.

1) 获取内存地址. 在虚拟机环境下, 内存地址分别对应被监控虚拟机的内核地址空间、监控程序的进程空间和机器地址空间. 通过特定的转换得到相应的两个空间地址, 获取虚拟机内核地址空间中被监控的内存段地址, 如图 3 所示. 图中 xc_translate_foreign_address() 和 xc_map_foreign_range()函数为 libxenctrl 提供的接口, 分别执行从被监控虚拟机的内核逻辑地址到机器地址和从机器地址到监控程序的进程地址的转换.

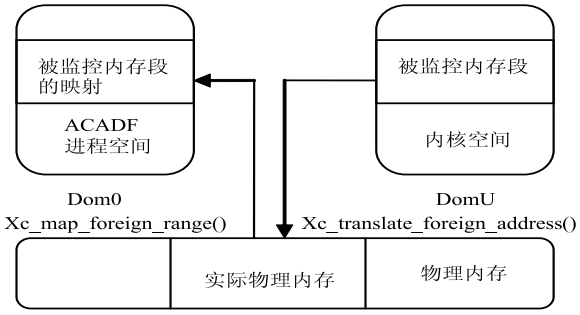

图 3　检测系统中不同地址间的转换关系

2) 修改页表. 为监控不变数据结构, 将存放数据结构的内存段设为只读, 即将指向该内存页的页表项的第一位设为 0. 通过 xc_vcpu_getcontext()函数, 得到存储 vcpu 所有信息的结构, 提取寄存器 CR3 值. 结合监控虚拟机的内核地址, 可找出指向该内存页的页表项的机器地址.

3) 截获非法写入. 当攻击程序向代码段或不变数据结构的内存段写入数据时, 页面异常会被触发. 通过 Xen 的 grant table 和 event 机制, 在 Dom0 和监控虚拟机之间建立一条事件通道, 使监控程序获取异常. 本文通过修改监控虚拟机的页面异常处理函数向监控事件通道发送通知. 当被监控虚拟机有非法写入时, 会调用页面异常处理函数捕获非法写入的攻击.

4) 检测代码段的完整性. 由于监控虚拟机的代码段可能分布在多个不同的物理内存页, 所以本文将虚拟机的代码段的不同页先后映射到监控程序的进程空间内. 之后使用 ELFHash()函数对其进行运算, 并将结果存于数组中. 根据 3.1 节中的前提假定, 认为第一次

计算的数组值是可信的. 每隔一段时间监控程序会对代码段不同页的 hash 值重新计算一次, 并与第一次的计算值进行比较, 如不相等判断存在攻击.

4 实验与分析

4.1 实验环境

本文对该框架进行了实现, 软件版本和硬件配置信息为 OS 采用 Ubuntu11.10-x86_64 Server; Linux 版本为 3.2.0-33-generic-x86_64; Xen 版本为 Xen-4.1.2; CPU 为 8Intel(R)-Core(TM)-i3 2100; 被监控虚拟机采用 Simple OS.

其中 Simple OS 是本文实现的一个支持 DomU 的最小内核系统, 能够在 Xen 上正常运行. 同样包含代码段和一些不变数据结构, 这里可视为一个简化的 Linux 内核. 本文提出的监控程序在 Simple OS 启动时, 调用 xen_interface_lib 将代码段和不变数据结构对应的物理内存设为只读, 并对它的代码段和不变数据结构的内存块进行监控, 以确保这些内存段不会被非法修改.

Simple OS 的不变数据结构中存在一个 sys_call_table, 其中一个成员指针 shutdown_p 指向执行系统关闭的函数 shutdown, 但我们需要退出该 Simple OS 时, 会通过该指针 shutdown_p 调用 shutdown. 我们利用一个函数 data_attack, 该函数会将指针 shutdowna_p 指向一个空函数, 使得无法通过 sys_call_table 来关闭 OS. 在特定的条件下触发 data_attack, 用来模拟修改内核静态数据的攻击.

在运行 Simple OS 时, 不启动监控程序. 触发 data_attack 后, 我们无法通过 sys_call_table 来关闭 OS. 当我们启动监控程序时, 触发 data_attack 时, Simple OS 被暂停运行, 并且监控程序发出非法修改内核静态数据提示.

同样, 我们实现一个 code_attack 函数, 该函数将一个空函数的代码段直接复制到函数 shutdown 所在的内存区, 这样调用 shutdown 函数时, 实际执行的是一个空函数, 什么都没做. 在特定的条件下触发 code_attack, 用来模拟修改内核代码的攻击. 在运行 Simple OS 时, 不启动监控程序, 触发 code_attack 后无法关闭 Simple OS. 启动监控程序时, 触发 code_attack 时, Simple OS 被暂停运行, 并且监控程序发出非法修改内核代码的提示.

4.2 结果分析

在 Simple OS 启动后, 监控程序对其进行监控. 当 data_attack 发生时, 监控程序输出一条消息, 提示 Simple OS 有非法修改内核静态数据攻击, 同时将被监控虚拟机 Simple OS 暂停. 在 code_attack 发生时, 监控程序同样检测出了异常并将 Simple OS 暂停. 由此可见本文提出的监控系统可以检测到修改代码段和不变数据结构这类内核攻击并实施具体干预措施.

5 结论与展望

本文提出内核入侵检测框架对动态修改内核代码和只读数据结构的攻击有很好的防御效果, 但目前框架只局限检测这一特定类型的内核攻击, 对于其他攻击, 如对动态修改可变数据等方式还不能够检测. 今后的研究中, 我们将会在该框架的基础之上进行扩展, 根据特定的攻击制定相应的检测规则.

此外, 随着 Linux 内核版本的发展, 针对不同的虚拟机系统, 实现不同的 xen_interface_lib, 这样可以让上层监控程序的开发独立于被监控操作系统.

参考文献

1 Avanade Research & Insights. Global Survey: Has Cloud Computing Mature? http://www.avanade.com/Documents/Research%20and%20Insights/FY11_Cloud_Exec_Summary.pdf

2 McAllister K. Writing kernel exploits. [2012-09-19]. http://ugcs.net/~keegan/talks/kernel-exploit/talk.pdf.

3 Garfinkel T, Rosenblum M. A Virtual Machine Introspection Based Architecture for Intrusion Detection. Proc. of the 10th Annual Network and Distributed System Security Symposium (NDSS '03), San Diego, CA, Feb.2003.

4 Payne BD, Carbone M, et al. Xenaccess Library. 2009. http://code.google.com/p/xenaccess/

5 Petroni NL, Fraser T, Molina J, Arbaugh WA. Copilot-aCoprocessor -based Kernel Runtime Integrity Monitor. Proc. for the 13th USENIX Security Symposium. August 2004.

6 Petroni NL, Hicks M. Automated detection of persistent kernel control-flow attacks. CCS'07: Proc. of the 14th ACM conference on Computer and Communications Security. 2007. 103–115.

7 wikipedia.org.Rootkit. http://en.wikipedia.org/wiki/Rootkit.

8 The Xen Team. Xen Interface manual Xen v3.0 for X86. Xen is Copyright(c)2002-2005.University of Cambridge,UK.

9 石磊,邹德清,金海.Xen 虚拟化技术.武汉:华中科技大学出版社,2009.

10 Mauerer W. Professional Linux Kernel Architecture, 2008.