软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(9):2229−2247 [doi: 10.13328/j.cnki.jos.005185]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 ARM 虚拟化扩展的安全防护技术*

李舟军[1], 沈 东[1], 苏晓菁[2], 马金鑫[3]

[1](北京航空航天大学 计算机学院,北京 100191)

[2](中国科学院 微电子研究所 微电子器件与集成技术重点实验室,北京 100029)

[3](中国信息安全测评中心,北京 100085)

通讯作者: 李舟军, E-mail: lizj@buaa.edu.cn

摘 要: 近几年来,随着移动平台用户量的增加,移动平台安全成为安全领域关注的焦点.而 ARM 的虚拟化扩展,使得如何基于虚拟化技术进行移动平台的安全防护成为一个研究热点.首先,介绍了虚拟化技术的分类以及早期的相关研究;然后给出了 ARM 虚拟化扩展的相关概念,并与 x86 虚拟化扩展进行了对比;随后,重点介绍了基于硬件辅助虚拟化技术的安全研究现状,主要包括通用的系统框架以及针对特定攻击的安全工具;最后,对基于 ARM 虚拟化扩展的安全防护技术的发展方向进行了展望.

关键词: ARM;虚拟化;系统安全;移动安全;虚拟机监视器

中图法分类号: TP316

中文引用格式: 李舟军,沈东,苏晓菁,马金鑫.基于 ARM 虚拟化扩展的安全防护技术.软件学报,2017,28(9):2229−2247. http://www.jos.org.cn/1000-9825/5185.htm

英文引用格式: Li ZJ, Shen D, Su XJ, Ma JX. Security technology based on ARM virtualization extension. Ruan Jian Xue Bao/Journal of Software, 2017,28(9):2229−2247 (in Chinese). http://www.jos.org.cn/1000-9825/5185.htm

Security Technology Based on ARM Virtualization Extension

LI Zhou-Jun[1], SHEN Dong[1], SU Xiao-Jing[2], MA Jin-Xin[3]

[1](School of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

[2](Key Laboratory of Microelectronics Devices & Integrated Technology, Institute of Microelectronics, The Chinese Academy of Sciences, Beijing 100029, China)

[3](China Information Technology Security Evaluation Center, Beijing 100085, China)

Abstract: In recent years, with the growth in the number of mobile platform users, mobile platform security has become the focal point in the field of information security. The virtualization extension of ARM, which facilitates the security of mobile platform based on virtualization technology, is a hot research topic. This paper first introduces the types of virtualization technology and previous related studies. Then the concepts of ARM virtualization extension are presented, and the comparison with the x86 virtualization extension is given as well. Subsequently, the paper focuses on the current situation of security research based on hardware virtualization extension, including the general system frameworks and security tools for specific attacks. Analysis of future's research trend of ARM virtualization-based security technology is put forward at the end.

Key words: ARM; virtualization; system security; mobile security; virtual machine monitor

* 基金项目: 国家高技术研究发展计划(863)(2015AA016004); 国家自然科学基金(61370126, 61672081, 61502536); 北京成像技术高精尖创新中心项目(BAICIT-2016001)

Foundation item: National High Technology Research and Development Program of China (863) (2015AA016004); National Natural Science Foundation of China (61370126, 61672081, 61502536); Beijing Advanced Innovation Center for Imaging Technology (BAICIT-2016001)

收稿时间: 2016-07-10; 修改时间: 2016-09-04; 采用时间: 2016-11-10; jos 在线出版时间: 2017-02-20

CNKI 网络优先出版: 2017-02-20 14:02:22, http://www.cnki.net/kcms/detail/11.2560.TP.20170220.1402.013.html

近几年来,超过 95%的智能手机以及多数的平板电脑和嵌入式设备都使用 ARM 架构的芯片.ARM 芯片凭借其低能耗的特性[1−3]使其在移动平台上占据了绝大部分的市场份额.与此同时,随着用户群体的逐渐增大,更多的恶意软件将目光聚焦在移动设备的操作系统上,例如安卓[4−7]和苹果[8]操作系统.然而,在 ARM 平台上的安全防护手段却远不及 x86 平台成熟.目前,对移动平台的安全研究多数集中在应用层以及框架层.而针对系统内核的攻击可获取 Root 权限,这些安全问题不但危害性很大,而且当前尚无有效的防护手段.因此,如何增强对移动平台系统内核的安全防护,成为一个亟待研究的课题.

虚拟化技术是一种被业界广泛应用的技术,x86 平台上的虚拟化已成为云计算环境中的主要机制[9].通过虚拟化技术,多个客户操作系统可以在同一个物理硬件上共存,并可通过虚拟机管理程序(hypervisor)提供的有限接口实现隔离.此外,虚拟化还可以用于系统安全防护.由于虚拟机管理程序的权限高于客户操作系统的权限,因此,虚拟机管理程序可有效发现与防御客户操作系统内核中的恶意行为.例如,Overshadow[10],InkTag[11],TrustPath[12]以及 AppShield[13]等,都是在 x86 平台上使用虚拟化技术保护系统安全的重要工作.

受到 x86 平台上研究成果的启发,很多研究者也使用虚拟化技术来保护 ARM 平台上的系统安全.2011 年底,ARM 发布的 ARM-v7 系列处理器芯片增加了虚拟化扩展和安全扩展[14],极大地推进了基于虚拟化技术的移动平台安全防护技术的研究.

本文第 1 节介绍虚拟化技术的分类以及早期的相关研究.第 2 节对 ARM 的虚拟化扩展进行介绍,并与 x86 虚拟化进行比较.第 3 节重点阐述基于硬件辅助虚拟化技术的 ARM 虚拟化框架,包括 Xen、KVM/ARM 及 OKL4 虚拟机管理程序.第 4 节对基于 ARM 虚拟化扩展的安全防护工作进行总结.最后,本文对于 ARM 虚拟化技术技术的发展方向进行展望.

1 虚拟化技术介绍

1.1 虚拟化的概念

最早的虚拟机可以追溯到 20 世纪 60 年代的 IBM M44/44X[15]以及 IBM 360/370 系统主机[16−18].它们最初是用来解决第三代架构和 IBM 操作系统中多道程序的弱点.近 20 年来,虚拟化技术取得了飞速的发展.目前,虚拟化技术已经比较成熟,并且在桌面系统中得到了广泛的应用,可支持安全计算平台[19,20]、内核调试[21,22]、服务器加固[23,24]以及多操作系统[25]等.近几年来,虚拟化技术也逐步应用于移动平台[26],并且成为研究热点之一.

虚拟化是一种结合或划分计算资源来呈现一个或多个操作环境的技术,它使用的方法包括硬件和软件的区分或整合,部分或全部的机器仿真、模拟、时间共享等[27].具体来说,虚拟化可以被看做一种将服务或者计算环境与硬件组件分离的方法.这种分离使得相同的硬件中可以运行多种软件环境,同时,每个环境和其他环境是隔离的.虚拟化也可认为是一个软件框架,在一台机器上模拟另一台机器上的指令[28].通常情况下,虚拟化服务是在操作系统和底层硬件之间的软件层中实现的.该软件层接收来自操作系统的请求,执行相关指令,并且将结果返回给操作系统.这一层通常称为虚拟机监视器(virtual machine monitor,简称 VMM)[29].Popek 等人[30]提出了虚拟机监视器的 3 个本质特征.

- 虚拟机监视器提供了与原机器本质上相同的程序执行环境;
- 运行在该环境中的程序的性能损失很小;
- 虚拟机监视器拥有对系统资源的完全控制.

为了提高性能,只有特权指令的执行通过 VMM,所有非特权指令都直接在硬件上执行.这些特权指令通常是访问硬件组件或改变系统关键数据结构的指令.处理器需要在管理模式中运行,从而能够执行特权指令.

1.2 虚拟化技术分类

1.2.1 全虚拟化技术

全虚拟化技术(full virtualization technology)是允许一个未修改的客户操作系统运行在一个完全虚拟化的环境中.由于客户操作系统并不知道其运行在一个虚拟化的环境中,因此它不需要经过任何的修改.全虚拟化通

常需要结合二进制翻译[31]和指令仿真[32]技术来实现.在全虚拟化环境中,大多数运行在客户操作系统中的特权指令被虚拟机监视器捕获,虚拟机监视器在这些指令执行前捕获并模拟这些指令.一些用户模式下无法被捕获的指令将通过二进制翻译技术处理.通过这种技术,小的指令块被翻译成与该指令块语义等价的一组新的指令.同时,为提高性能,非特权指令会直接在硬件上执行.

图 1 表示了全虚拟化的系统结构.

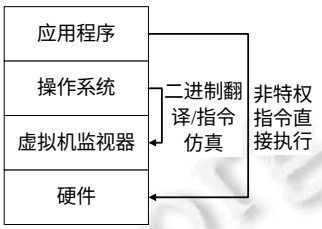

Fig.1 System structure of full-virtualization

图 1 全虚拟化的系统结构

1.2.2 半虚拟化技术

半虚拟化技术(para-virtualization technology)需要对客户操作系统进行修改.特权指令被替换为一个虚拟化调用(hypercall)来跳转到虚拟机监视器中.虚拟机监视器为客户操作系统提供了一些系统服务的虚拟化调用接口,例如内存管理、设备使用及终端管理等,从而确保全部的特权模式活动都从客户操作系统转移到虚拟机监视器中.半虚拟化通常比全虚拟化速度更快,这主要是由于半虚拟化通过改变客户操作系统的代码来避免调用特权指令,从而减少了二进制翻译和指令仿真带来的动态开销.但这种做法需要维护一个修改过的客户操作系统,必将带来一定的额外开销.由于半虚拟化系统的性能接近于原生系统,因此这个开销是可接受的.

图 2 表示了半虚拟化的系统结构.

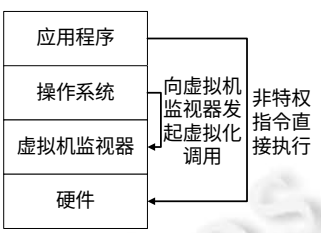

Fig.2 System structure of para-virtualization

图 2 半虚拟化的系统结构

1.2.3 硬件辅助虚拟化技术

由于虚拟化技术的广泛应用,硬件制造商也在硬件层面提供了虚拟化支持.Intel 和 AMD 都支持虚拟化技术(分别为 VT-x 和 AMD-V),而 ARM 也在 ARMv7 架构中增加了虚拟化扩展.这些新的架构都为虚拟机监视器提供了一个新的操作模式,在 x86 架构中,一般被称为根模式(root mode);而在 ARM 架构中,则被称为虚拟化模式(hyp mode).该模式拥有比操作系统所在的管理模式(svc mode)更高的权限.因此,在管理模式中执行的全部特权指令会被自动捕获,并且控制权被转移到虚拟化模式中的虚拟机监视器中.同时,管理模式中寄存器的状态会被保存,以便在返回的时候读取.因此,硬件辅助虚拟化不需要二进制翻译和指令仿真技术,同时也不需要对操作系统进行修改.该技术与全虚拟化技术相比效率更高,也不同于半虚拟化技术需要对操作系统进行修改.因此在硬件的辅助下,移动平台上的虚拟化技术具有较大的发展前景.

图 3 表示了硬件辅助虚拟化的系统结构.

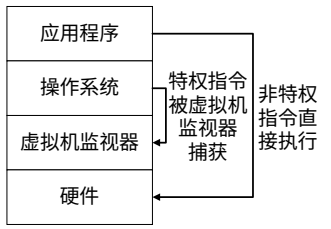

Fig.3 Structure of hardware-based virtualization

图 3 硬件辅助虚拟化的系统结构

1.3 全虚拟化与半虚拟化技术相关工作

由于缺少硬件虚拟化的支持,早期对虚拟化技术的研究,多数以全虚拟化技术和半虚拟化技术为主.QEMU[33,34]是比较知名的使用全虚拟化技术的工作.QEMU 通过指令仿真运行未修改的操作系统,支持 x86,ARM,MIPS 等多个平台.同时,它也可以实现跨平台仿真,例如,在 ARM 平台上运行 Windows.Oh 等人[35]提出了一个在手机系统中基于全虚拟化的虚拟机监视器——ViMo.ViMo 在系统运行时扫描系统代码,并执行关键指令.同时,ViMo 为每个虚拟机分配内存空间,并保证了虚拟机间的内存隔离.

全虚拟化技术由于采用了二进制翻译和指令仿真技术,导致系统性能明显降低.相比而言,半虚拟化和硬件辅助虚拟化的性能就有了明显提升.在硬件虚拟化支持出现之前,研究者主要研究半虚拟化技术,并提出了许多技术框架,如 Xen on ARM[36],KVM for ARM[37]及 OKL4 microvisor[38]等.后续的许多基于半虚拟化技术的安全研究工作也都基于上述框架.

Lee 等人[39]设计了一个多层访问控制机制,使得 Xen on ARM 更为安全.Park 等人[40]研究了让实时操作系统在 Xen on ARM 上运行的问题,使用了 EDF 算法[41]来保证任务的实时性问题.基于 Xen on ARM 的架构,李大江[42]在 ARM 平台上设计并且实现了一种轻量级虚拟机,使其可以在一个模拟的 goldfish 平台上同时运行 Android 和 mini-os 系统.Rossier 等人[43]基于 Xen 开发了一个 EmbeddedXEN,使其能以尽可能小的性能开销运行在不同的 ARM 内核上.钟木忠[44]通过签名认证的方法消除了 Xen on ARM 上两种可能的安全威胁——Xen 监控器通信安全与虚拟机系统镜像[45]安全.赵亚辉[46]基于 KVM for ARM 实现了一个无需修改客户操作系统的虚拟机管理程序.Ding 等人[47]基于 KVM 提出了 ARMvisor,并且成功在 ARM 开发板上运行了 Linux 系统.

2 ARM 虚拟化扩展

2.1 ARM虚拟化扩展概述

2011 年 11 月 23 日,ARM 在 ARMv7 架构中增加了虚拟化扩展.虚拟化扩展对于 ARMv7 架构来说是一个可选的扩展,使用虚拟化扩展必须同时包括安全扩展以及高位物理地址扩展.

ARM 的虚拟化扩展仅仅应用在非安全模式中.ARM 引入了一个新的处理器模式——虚拟化模式.该模式拥有比现有的非安全管理模式更高的权限,因此,原始的客户操作系统和应用程序不需修改,依然能够运行在原有的管理模式以及用户模式(usr mode)中,如图 4 所示.

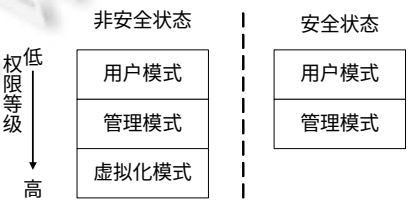

Fig.4 Structure of ARM processor

图 4 ARM 处理器架构

进入虚拟化模式的方法分为主动进入和被动进入两种.在管理模式下,可以通过一条 hvc 汇编指令主动进入虚拟化模式.同时,通过相关寄存器的配置,ARM 芯片可以捕获一些来自用户模式或者管理模式的异常,从而进入虚拟化模式.ARM 虚拟化扩展在不同的模式下提供了单独的寄存器.例如:在用户模式下,栈指针寄存器(stack pointer,简称 SP)为 SP_usr;而在虚拟化模式下,该寄存器为 SP_hyp,从而避免了在进行模式切换时的寄存器值的存取,提高了系统性能.

2.2 ARM虚拟化扩展机制

2.2.1 可配置的指令捕获

ARM 虚拟化扩展提供了一些配置选项,以决定一条指令是否被捕获进入虚拟机监视器.虚拟化配置寄存器(hyp configuration register,简称 HCR)中的不同位,决定了不同种类的捕获方式.例如:当 HCR 的捕获通用异常位(trap general exception bit,简称 TGE bit)被设置为 0×1 时,所有的系统调用都会被捕获进入虚拟机监视器.因此,通过合理配置 HCR,虚拟机监视器可对异常进行选择性拦截,而其他异常都可由客户操作系统处理,从而提高系统性能.在虚拟机监视器拦截异常后,虚拟化状态寄存器(hyp syndrome register,简称 HSR)记录了被捕获到虚拟机监视器中的异常信息,其格式如图 5 所示.其中,*HSR*[31:26]为异常类型(exception class,简称 EC),它记录了捕获的原因.例如,如果 *EC* 的值为 0×12,说明虚拟机监视器捕获到了一个虚拟化调用.

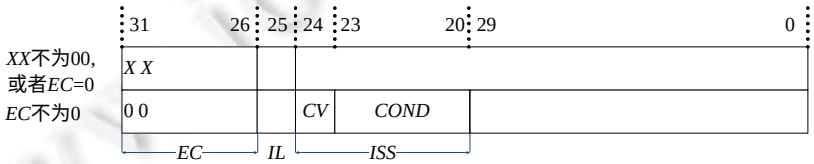

Fig.5 Format of HSR

图 5 HSR 的格式

2.2.2 二级页表翻译

为了更好地控制客户的虚拟内存,ARM 使用二级页表翻译:第 1 级页表由客户操作系统维护,将客户虚拟地址映射为客户物理地址(ARM 称作中间物理地址);第 2 级页表由虚拟机监视器维护,将客户物理地址映射为机器物理地址,并对客户操作系统保持透明.因此,虚拟机监视器可以通过配置第 2 级页表项中的属性控制位来实现客户操作系统对内存的访问控制.

2.2.3 虚拟中断

对中断控制器的仿真将显著增加系统的复杂性,并且需要频繁地陷入虚拟化模式.为避免出现这种情况,ARM 引入了虚拟中断的概念.ARM 通过引入一个新的硬件组件——虚拟 CPU(VCPU)接口,以支持虚拟中断.该接口能被映射到客户操作系统,并作为通用中断控制器(general interrupt controller,简称 GIC)的 CPU 接口.因此,客户操作系统可在不陷入虚拟机监视器的情况下使用该接口确认和清除中断.虚拟机监视器仍然要模拟中断分配器,并且捕获所有对中断分配器的客户访问.这不会引起性能问题,因为分配器通常只在启动时(或者模块加载时)被访问,以便为特定的中断注册驱动并且引导他们到特定的(虚拟)CPU 中.

2.3 与x86虚拟化扩展的对比

与 ARM 相同,如今 x86 平台也支持虚拟化扩展.由于 CPU 的架构不同,二者的虚拟化扩展也不尽相同.下面将对 ARM 虚拟化扩展和 x86 虚拟化扩展(以 Intel VT-x 为例)进行比较.

2.3.1 权限模式

ARM 和 x86 的虚拟化扩展都拥有一个更高的权限模式,在 x86 架构中被称为根模式,而在 ARM 中被称为虚拟化模式.然而,这两个模式有很大的不同.

- 在 x86 架构中,与根模式相对应的是非根模式(non-root mode),非根模式即传统的 CPU 模式,包括 Ring 0~Ring 3 的权限级;而在根模式中,也有对应的 Ring 0~Ring 3 的权限级.也就是说,如图 6 所示,根模式

和非根模式是平行的;

- 而在 ARM 中,如图 4 所示,虚拟化模式和原有的几种模式是并列的,只是虚拟化模式拥有比管理模式更高的权限等级.

对比图 6 和图 4,x86 中的根模式在结构上与 ARM 的安全扩展更为相似.

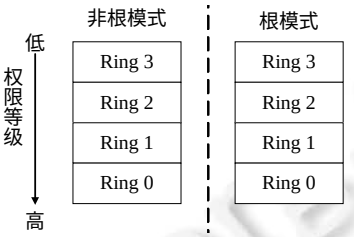

Fig.6 Structure of x86 virtualization

图 6 x86 虚拟化架构

2.3.2 虚拟内存

ARM 和 x86 的虚拟化扩展都引入了一次额外的页表寻址,这在 x86 中称为扩展页表(extended page tables,简称 EPT),而在 ARM 中称为二级页表翻译.二者都是将客户虚拟机中翻译得到的客户物理地址转换为机器物理地址,不同的是:x86 架构中的扩展页表使用了现有的页表格式,而 ARM 中的二级页表使用了新的页表格式.

2.3.3 虚拟机入口

在 Intel 处理器中,每个虚拟机都存在一个虚拟机控制区域(virtual machine control state,简称 VMCS).它是一段映射的内存,用于上下文切换.虚拟机管理程序在进入虚拟机之前会通过 VMCS 恢复相应的状态.而在 ARM 中,进入虚拟机更为简单.虚拟机管理程序只需要设置程序计数器(program counter,简称 PC)并执行一个 ERET 指令,即可进入虚拟机中.

2.3.4 上下文切换性能

由于虚拟机管理程序需要经常拦截客户虚拟机中的事件,因此上下文切换的性能开销对于整个系统的性能至关重要.Dall 等人[48]测试了在 ARM 和 x86 架构中上下文切换所需要的 CPU 时钟周期,测试结果表明:在 ARM 架构中,一次上下文切换需要 27 个时钟周期;而在 x86 架构中,一次上下文切换需要 600 个~800 个时钟周期.对于 x86 架构中的每次上下文切换,CPU 会保存全部的寄存器状态;而对于 ARM 架构中的每次上下文切换,CPU 只需保存某些模式特定的寄存器(如 SP).因此,ARM 架构的这种上下文切换方式在保证性能的基础上也会更加灵活,虚拟机管理程序可以根据具体情况有选择地保存需要的寄存器.

2.3.5 对比总结

综上所述,ARM 虚拟化扩展和 x86 虚拟化扩展在设计思路上是相似的.但由于 ARM 和 x86 指令集的不同以及平台的差别,两者在具体的实现方式上存在一些差异.表 1 总结了 ARM 虚拟化扩展与 x86 虚拟化扩展的对比情况.

Table 1 Comparison between ARM and x86 virtualization extension

表 1 ARM 与 x86 的虚拟化扩展对比

比较类别	ARM 虚拟化扩展	x86 虚拟化扩展
权限模式	虚拟化模式	根模式
虚拟内存	二级页表	扩展页表
客户标识	快表标记	快表标记
虚拟机入口	直接进入	VMCS
仿真支持	快速捕获仿真	无
I/O	无	安全直接内存存取
上下文切换性能	27 时钟周期	600~800 时钟周期

3 硬件辅助虚拟化技术框架

在 ARM 虚拟化扩展提出之后,硬件辅助虚拟化成为了一个新兴的 ARM 虚拟化发展趋势.许多团队提出了自己的虚拟化框架,它们一般是开源的,给出了一个通用的虚拟化框架,实现了一些通用的虚拟化接口,开发者可根据自己的需求添加额外的功能.随后的很多研究成果都是基于这些虚拟化框架或者仿照这些虚拟化框架提供的思路来实现的.

3.1 Xen

Xen[49]最早是由 Barham 等人实现的一个面向桌面操作系统的开源虚拟化程序,因此,开发者可以改进 Xen,为 Xen 的完善做出贡献.

Xen1.0 发布于 2004 年,随后发布了 Xen2.0.自 Xen4.3.0 开始,Xen 加入了对 ARM 虚拟化扩展的支持.目前,Xen 的最新版本为 Xen4.7.0,可支持 x86,x86_64,IA64,ARM 以及其他 CPU 架构,并已被应用于多种客户操作系统的虚拟化,包括 Windows,Linux,Solaris 以及 BSD 操作系统的多种版本.下文详细介绍了在 ARM 虚拟化平台上运行的 Xen 的系统架构与设计等方面的内容,后文中的 Xen 均表示在 ARM 平台上的 Xen.

3.1.1 系统架构

Xen 的系统架构图如图 7 所示.在 Xen 中,不同的操作系统被分为不同的域(domain).其中,域 0(domain 0)为 Xen 中的管理域,其上运行着一个可信的操作系统,负责执行一些管理进程以及设备驱动.而域 *U*(domain *U*)为用户域,其上运行着不可信的客户操作系统.Xen 中只有一个域 0,但可以拥有多个域 *U*,不同的域 *U* 中的操作系统互相隔离,同时,一些特殊请求会通过虚拟机监视器与域 0 通信完成.

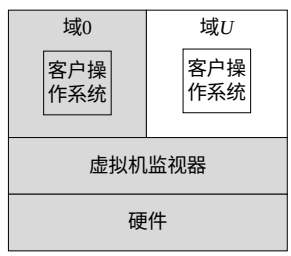

Fig. 7 System structure of Xen (shadow parts are TCB)

图 7 Xen 的系统架构图(阴影部分为可信计算基)

3.1.2 系统设计

Xen 最初主要应用于 x86 架构平台,由于 ARM 指令集的特性以及虚拟化扩展的加入,运行在 ARM 平台上的 Xen 中的系统设计都与运行在 x86 平台上的 Xen 在各方面均有区别.

- 虚拟化方式

最初在 x86 平台上运行的 Xen 是一个半虚拟化的虚拟机管理程序.虽然半虚拟化在性能上比全虚拟化提高很多,但是由于核心操作需要软件来实现,因此依然存在一定性能开销.当 Xen 开始实现 ARM 的支持时,ARM 已经拥有硬件虚拟化扩展,因此,Xen 在 ARM 上采用硬件辅助虚拟化的方式来实现.尽管 Xen 依然使用域 0 来进行辅助操作,但诸如状态切换等操作都可以直接通过硬件实现,从而提升了系统性能.同时,基于硬件辅助虚拟化的 Xen,不需要额外修改操作系统源码,只要实现相关驱动即可进行系统的移植,因此,可移植性与半虚拟化相比也有很大提升.

- 内存管理方式

在 x86 架构中,Xen 通过实现一个影子页表来实现内存管理.虽然 Xen 对影子页表的页表翻译流程做了许多优化,但整个过程依然复杂并产生了很大的性能开销.同时,虚拟机管理程序对于内存访问的拦截均通过虚拟化调用来实现,并不是真正由虚拟机管理程序进行拦截.而在 ARM 上运行的 Xen,直接利用 ARM 硬件虚拟化扩

展提供的二级页表机制,通过配置二级页表上面的访问控制属性位即可实现对内存的管理控制及拦截.在代码的实现量上比 x86 平台的 Xen 减少了上万行代码,从而缩小了可信计算基,增加了系统的安全性.从性能上考虑,通过硬件实现的页表翻译流程也减少了相应的系统开销.

- 中断控制方式

ARM 在虚拟化扩展中定义了一个通用中断控制器(generic interrupt controller,简称 GIC)架构.GIC 负责将设备的中断传递给 CPU,同时,CPU 通过 GIC 可以查询中断的来源.Xen 利用了 ARM 提供的 GIC 来处理对客户虚拟机中断的拦截与处理.当 Xen 拦截到客户虚拟机产生的中断时,会使用 GIC 将该中断传递给域 0 进行处理;当域 0 完成对该中断的处理时,Xen 使用 GIC 将事件返回给客户虚拟机.Xen 用这种方式实现了对客户虚拟机的中断虚拟化,并在一定程度上保证了虚拟机的安全性.

3.1.3 基于 Xen 的相关研究

由于 ARM 平台的硬件虚拟化扩展还属于一个比较新的方向,因此,基于硬件虚拟化扩展的相关研究工作并不多.Lengyel 等人[50]提出了基于 ARM 虚拟化扩展和安全扩展的多层式安全架构,在这个架构中,包含了如下的安全特征.

- 加载时对关键安全组件的认证;
- 运行时用于虚拟机管理程序认证的自定义可信域(TrustZone)组件;
- 组件的强隔离(多个客户虚拟机,一个安全 I/O 虚拟机,一个加密引擎,一个自省引擎);
- 利用 Xen 虚拟机管理程序的 Xen 安全模块-先进安全内核架构策略框架用于严格的策略执行.

通过这种多层安全架构,可以使得软件组件之间被很好地进行隔离,同时,这种架构也提供了一个细粒度的访问控制.

3.2 KVM/ARM

基于内核的虚拟机(kernel-based virtual machine,简称 KVM)[51]也是一种常用的虚拟机监视器,在 Linux 2.6.20 版本之后,KVM 就已经存在于 Linux 内核中了,并已广泛应用于基于 x86 或 PowerPC 处理器的传统服务器中.

Xen 是完全运行在虚拟化模式的虚拟机管理程序,这种虚拟机管理程序可以拥有更好的性能以及更小的可信计算基(trusted computing base,简称 TCB).然而,由于 ARM 硬件在许多方面比 x86 更加多样化,因此,硬件组件通常是由不同的设备制造商以非标准方式集成在 ARM 设备上的.因此,为使 Xen 支持不同的系统级芯片,开发者必须在 Xen 中为每种芯片实现新的串口设备驱动.

虽然 ARM 硬件相比 x86 架构缺乏标准化,但它们几乎都支持 Linux 操作系统.因此,通过与 Linux 的结合,基于 KVM 的 ARM 虚拟化技术 KVM/ARM[48]可在任何运行新版本 Linux 内核的设备上使用.同时,利用现有的 Linux 内核,KVM/ARM 无需重新实现复杂的内核功能,并可避免引入致命的 BUG.

3.2.1 系统架构

KVM/ARM 引入了分割模式虚拟化,这是一种新的虚拟机管理程序设计方法.它将核心虚拟机管理程序分割成两部分,让整个程序在不同的特权 CPU 模式下运行,从而可以利用每个 CPU 模式提供的特定功能.KVM/ARM 使用分割模式虚拟化,在利用虚拟化模式提供的 ARM 硬件虚拟化支持的同时,也利用了在管理模式中运行的 Linux 内核服务.分割模式虚拟化也使得 KVM/ARM 可以在不修改现有 Linux 内核代码的基础上,将其整合到 Linux 系统中.如图 8 所示,KVM/ARM 根据分割模式虚拟化分成了两部分:底层管理程序(lowvisor)和高层管理程序(highvisor).

底层管理程序利用虚拟化模式提供的硬件虚拟化,可支持实现如下 3 个主要功能.

1) 底层管理程序通过硬件的合理配置设置正确的执行上下文,并在不同的执行上下文之间实行保护和隔离.底层管理程序可以直接与硬件保护功能交互,因此它非常关键并且代码要保证尽可能地小;
2) 底层管理程序负责虚拟机和主机的执行上下文之间的互相切换.因为底层管理程序是运行在虚拟化模式的唯一组件,它负责切换相关的硬件配置;

3) 底层管理程序提供了一个虚拟化捕获处理器,用来处理被捕获到虚拟机管理程序的中断和异常.底层管理程序只提供少量的必要处理,主要的工作都在切换到高级管理程序后由高级管理程序来完成.

高层管理程序在管理模式中运行,它也作为主机 Linux 内核的一部分,可直接利用现有的 Linux 功能,并使用标准的内核软件数据结构和机制.因此,在高层管理程序中更容易实现高级功能.例如,底层管理程序负责虚拟化捕获的处理与上下文的切换,高层管理程序则处理虚拟机的二级页面错误并执行指令仿真.由图 8 可以看出:虚拟机的内核部分与高层管理程序均运行在拥有相同权限等级的管理模式中,但虚拟机的内核部分使用二级页表翻译,并可通过寄存器的设置被底层管理程序捕获到虚拟化模式中.因此,该部分同样处在 KVM/ARM 的监视和管理之下.

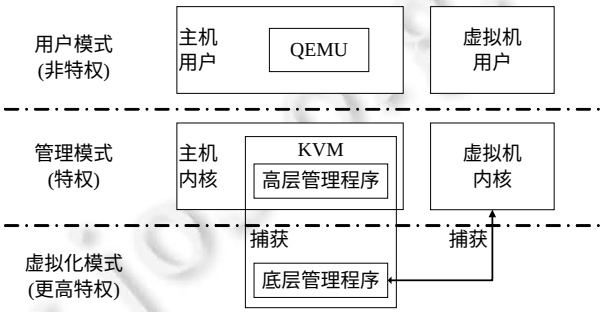

Fig.8 System structure of KVM/ARM

图 8 KVM/ARM 的系统架构图

因为虚拟机管理程序被分割在管理模式和虚拟化模式中,因此在虚拟机和高层管理程序之间的切换涉及多次模式切换.虚拟机中需要被虚拟机管理程序处理的中断或异常,首先要被捕获到底层管理程序中.底层管理程序随后产生另一个异常进入高层管理程序.反之,从高层管理程序进入虚拟机,也需要先从管理模式切换到虚拟化模式,再切换到虚拟机中.因此,分割模式虚拟化在上下文切换时会产生两次模式切换.

3.2.2 系统设计

基于这种分割模式的虚拟化,KVM/ARM 具体实现了多方面的虚拟化功能,主要包括如下 4 点:CPU 虚拟化、内存虚拟化、I/O 虚拟化以及中断虚拟化.

(1) CPU 虚拟化

为实现 CPU 虚拟化,KVM/ARM 必须给虚拟机提供一个与物理 CPU 完全相同的接口,同时确保虚拟机管理程序能控制硬件.因此,KVM/ARM 需确保运行在虚拟机中的软件与运行在物理 CPU 上的软件可以访问到相同的寄存器上下文,同时还要确保在运行虚拟机时,虚拟机管理程序以及其主机内核的物理硬件状态是稳定的.在虚拟机和主机之间相互切换时,通用寄存器的状态可以简单地通过内存读写来实现上下文切换.如果某个对硬件状态的访问可能影响虚拟机管理程序或向虚拟机泄露硬件信息,则 KVM/ARM 将捕获这个敏感指令,并对其进行仿真.

(2) 内存虚拟化

当虚拟机运行时,KVM/ARM 通过开启二级页表翻译来实现内存虚拟化.二级页表翻译只能在虚拟化模式中配置,并对虚拟机完全透明.高层管理程序对二级页表翻译进行管理,使得虚拟机只能访问自己申请的指定内存空间.若虚拟机试图访问其他内存,则产生一个二级页面错误异常,并被虚拟机管理程序捕获与处理.该机制保证了虚拟机不能访问属于虚拟机管理程序或其他虚拟机的内存空间.由于高层管理程序和底层管理程序拥有对系统的完全控制权,因此当它们运行时,二级页表是禁用的.当虚拟机管理程序切换到虚拟机时,会开启二级页表翻译机制,并对二级页表基址寄存器进行相应配置.通过内存虚拟化机制,虽然高层管理程序和虚拟机使用相同的 CPU 模式(管理模式),二级页表翻译确保了高层管理程序不会被其他虚拟机访问.

(3) I/O 虚拟化

KVM/ARM 利用 QEMU 和 Virtio[52]现有的用户空间设备仿真技术来提供 I/O 虚拟化.在硬件级别上,ARM 架构的全部 I/O 机制都是基于对内存映射 I/O(memory mapping I/O,简称 MMIO)设备区域的读取/保存操作.除了被直接分配给虚拟机的设备外,其他 MMIO 设备区域对虚拟机来说都是不可访问的.KVM/ARM 使用二级页表翻译机制来确保物理设备不能被虚拟机直接访问.虚拟机管理程序会捕获虚拟机的每次内存跨界访问,并将相关信息转发给 QEMU 中对应的仿真设备.

(4) 中断虚拟化

由于 KVM/ARM 与 Linux 是紧密结合的,因此它能够重用现有的设备驱动和相关功能,例如中断处理.当程序在虚拟机中运行时,在 KVM/ARM 对 CPU 进行配置后,可将全部的硬件中断捕获到虚拟化模式中.当中断产生时,它会进行状态切换进入高层管理程序,再由主机处理.因此,虚拟机管理程序可以确保对硬件资源的完全控制.当程序在主机以及高层管理程序中运行时,中断就直接被捕获到管理模式中,从而避免了进入虚拟化模式的额外开销.在以上两种情况中,所有硬件中断的处理都在主机中,通过重用 Linux 现有的中断处理功能完成.

3.2.3 基于 KVM/ARM 的扩展

Paolino 等人[53]基于 KVM/ARM 提出了基于内核的可信虚拟机(T-KVM).T-KVM 是一个基于 KVM/ARM 的安全架构,它集成了软件和硬件组件来保护客户操作系统,并支持在 ARM 虚拟机中的可信计算.T-KVM 共包含 4 个独立的组件:ARM 虚拟化扩展、ARM 安全扩展(TrustZone)、可信执行环境(TEE)接口以及 SELinux 的强制访问控制(MAC)模块.T-KVM 架构为客户应用提供了较好的隔离性且支持可信计算.

3.3 OKL4虚拟机管理程序

由于虚拟化技术和微内核技术在很多方面具有相似之处,因此二者的优劣也成为许多人讨论的焦点.Open Kernel Labs 将两者结合在一起,提出了一种基于微内核的虚拟机管理程序——OKL4 虚拟化微内核(OKL4 microvisor)[38].OKL4 虚拟化微内核是 L4 微内核[54]的一个变种,可运行在单核或多核的 ARM,x86 以及 MIPS 处理器平台上.

基于硬件虚拟化扩展,Varanasi[55]在 ARM 上将 OKL4 移植到虚拟化模式中,实现了一个虚拟机管理程序.OKL4 运行在虚拟化模式中,既能发挥其 TCB 小的优势,又能在虚拟化扩展的辅助下实现更好的管理功能.

3.3.1 系统设计

为了让基于 OKL4 的虚拟机管理程序尽可能简单,并保持一个小的 TCB,Varanasi 给出了如下的设计.

- 通过静态编译的方式配置客户虚拟机的上限数量.

一般而言,虚拟机管理程序可通过静态编译配置客户虚拟机的数量上限,也可动态地分配新虚拟机.然而,虚拟机的动态创建比静态编译配置更为复杂,因此需要更多的实现代码,从而增加了 TCB 的代码量,进而增大了系统的安全风险.同时,由于 ARM 平台主要用于移动设备,而一台移动设备上的客户虚拟机数量不会像桌面平台那样大,因此,设计者选择通过静态编译的方式配置客户虚拟机的上限数量.

- 虚拟机管理程序中的内存采用平板映射.

在虚拟机管理程序中,内存映射通常可以采用页表寻址或平板映射.虚拟机管理程序并不会被上层客户操作系统访问,并且采用页表寻址格式需要编写更多代码来实现页表配置.因此,设计者采用简单的平板映射方式来实现虚拟机管理程序中的内存访问.这种方式能使得虚拟机管理程序尽可能的简单,并减小了 TCB.而在客户操作系统中,设计者采用二级页表翻译机制来实现内存访问,如图 9 所示.

- 虚拟机管理程序的组件全部位于虚拟机管理程序中.

不少微内核并没有众多管理组件,而是通过进程间通信(inter-process communication,简称 IPC)和外部管理组件通信来实现管理工作,从而确保微内核的 TCB 足够小.但组件如果存放于上层客户操作系统中,每次操作都需要进行通信,从而对系统性能产生影响.同时,中断管理、二级页表翻译以及状态存取等都是对虚拟机安全至关重要的组件.因此,将这些组件存放于虚拟机管理程序中,在保证系统的性能的同时,也保证了系统的安全性.

- 虚拟机间的通信采用共享内存和异步通信两种方式.

虚拟机间的通信方式主要包括异步通信、同步通信和共享内存.对于内容较大的信息,需采用共享内存的方式;而对于内容较小的信息,通过异步通信的方式可以保证其性能.如果虚拟机之间使用同步机制通信,会让虚拟机因等待返回消息而阻塞.因此,设计者选择共享内存和异步通信的两种方式实现虚拟机之间的通信.

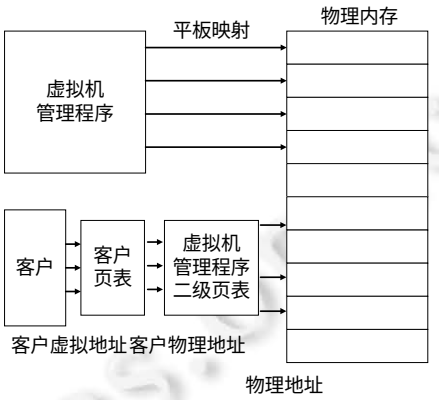

Fig.9 Virtual memory mapping in OKL4 hypervisor

图 9 OKL4 虚拟机管理程序中虚拟内存映射

3.3.2 系统实现

基于 OKL4,Varanasi 在实现中增加了如下的扩展:

- 增加了一个从内核模式进入虚拟化模式的系统调用;
- 为每个客户虚拟机实现了客户页表;
- 增加了对中断和时钟的支持,同时生成中断用于模式切换;
- 扩展了虚拟中断.如果中断不准备进入虚拟机管理程序,则会产生一个虚拟中断传递给客户虚拟机;
- 对一些设备进行了虚拟化;
- 增加了对 Linux 的支持,并可以运行多个 Linux 系统.

4 基于硬件辅助虚拟化技术的安全研究

随着硬件虚拟化技术及其相关研究的不断深入,越来越多的研究者将硬件虚拟化技术和系统安全问题联系起来.在 ARM 虚拟化扩展的辅助下,虚拟机管理程序可以运行在权限更高的虚拟化模式中.因此,虚拟机管理程序可以对操作系统进行有效的管理.因此近几年来,很多人都基于硬件辅助虚拟化技术对系统安全问题进行了研究.

4.1 DroidVisor

由于虚拟机管理程序的权限高于操作系统内核,因此,通过虚拟化来保护内核是一种非常有效的手段.杨永等人[56]利用 ARM 虚拟化扩展实现了一个 Android 内核保护监控器——DroidVisor.

DroidVisor 运行在虚拟化模式中,可以隔离可疑模块,并保护内核中对象的完整性.同时,它还能够通过检测隐藏模块和隐藏进程来检测 rootkit,从而有效地保护内核.

4.1.1 内核完整性保护

操作系统中存在一些可加载内核模块,用来提供额外的功能.攻击者可利用第三方的可加载内核模块进行攻击.DroidVisor 监控可加载内核模块的安装和卸载,如果该模块不在自己的可信列表中,则对其使用单独的二级翻译页表,从而实现内存隔离.当这些模块有异常行为时,DroidVisor 可对这些行为进行拦截和处理.

同时,DroidVisor 还可保护内核中关键对象的完整性.DroidVisor 通过 System.map 文件得到关键对象的虚拟地址,接着,通过二级页表翻译得到对应的物理页表项,并通过设置对应页表项的访问控制位实现访问控制.

DroidVisor 将关键对象的物理页设为只读,当该页被修改时,程序会产生页面错误,该错误会被虚拟机管理程序捕获.虚拟机管理程序捕获到该异常后,可以检查该异常是否由恶意代码导致:如果是由恶意代码导致的,虚拟机管理程序将该异常返回给客户操作系统的异常处理程序,从而实现异常拦截;如果是正常的修改,虚拟机管理程序会负责进行修改,随后返回客户操作系统.

4.1.2 Rootkit 检测

对内核完整性的保护可以防止攻击者对关键对象的修改,但无法有效防护使用进程隐藏和模块隐藏的rootkit.在保护内核完整性的同时,DroidVisor 还在进程切换时对模块列表和进程列表进行检查.如果发现有模块或进程不在已知列表中,则说明存在这种 rootkit.

在进程切换时,操作系统会更改页表的基址寄存器.在 ARM 中通过某些配置,可使该寄存器在修改时被虚拟机管理程序捕获.因此,DroidVisor 选择在进程切换时进行拦截,这样可以检测即将运行的进程是否为隐藏的进程,从而在可疑进程运行前将其阻止.具体的流程如图 10 所示.

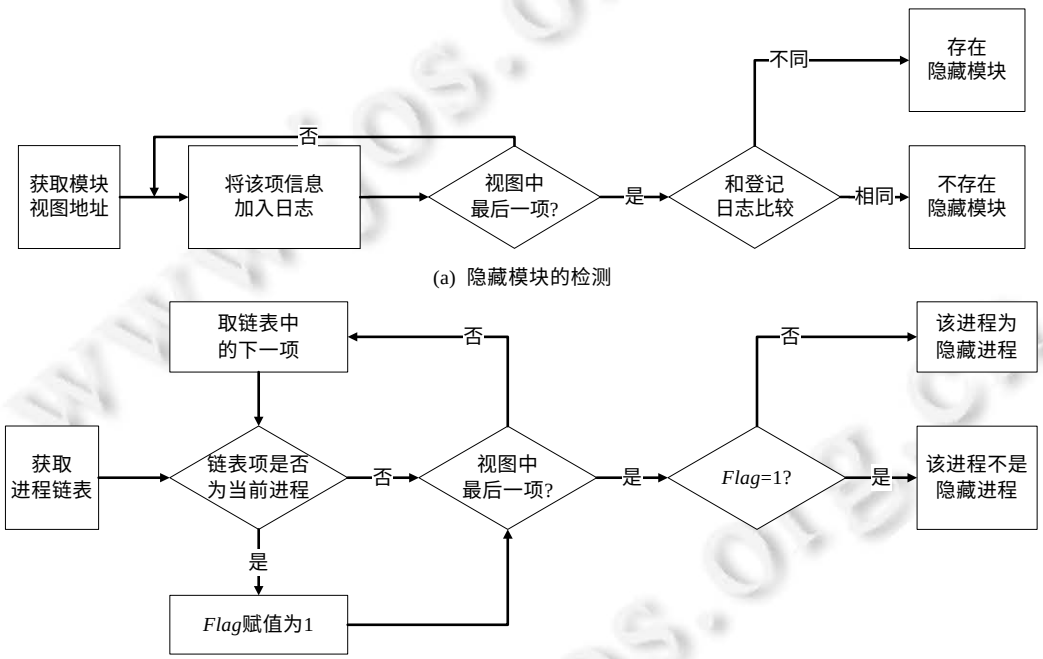

Fig.10 Workflow of rootkit detection

图 10 Rootkit 检测流程图

4.1.3 DroidVisor 自身的保护

DroidVisor 本身是一个轻量级的虚拟机管理程序,因此代码量比较小,即,可信计算基比较小,降低了出现 BUG 的风险.同时,DroidVisor 通过对页表翻译的控制,将虚拟机管理程序的内存空间置于客户操作系统可以访问的范围之外,从而使得客户操作系统无法访问到虚拟机管理程序的内存空间.

同时,对于 DroidVisor 在客户操作系统中加载的辅助模块,DroidVisor 通过二级页表翻译进行保护的同时,也在 TrustZone 的辅助之下实现了一个白名单验证机制,从而更好地保护了 DroidVisor 的功能.

4.2 控制流追踪框架

控制流分析是发现系统是否被攻击的一种重要手段.通过适当的控制流分析,研究者可以了解程序的执行流程,并且通过与正常的控制流对比来确定程序是否正确运行.因此,Horsch 等人[57]使用硬件辅助虚拟化技术建

立了一个控制流追踪的虚拟机管理程序框架,以达到保护程序正常控制流的目的.

该框架共分为 5 个部分:异常处理模块、控制模块、追踪模块、二级页表管理模块和分析模块,如图 11 所示.其中,

- 异常处理模块用于捕获客户操作系统的异常(包括虚拟化调用和页面错误):如果捕获到的异常是虚拟化调用,则传递给控制模块;如果是页面错误,则传递给追踪模块;
- 控制模块负责解析虚拟化调用的参数,并根据参数的值对整个框架的功能进行配置;
- 追踪模块接收到页面错误后,在二级页表管理模块的辅助之下,进行控制流追踪,生成相关的数据,供分析模块使用;
- 分析模块作为一个单独模块,可以根据使用者的不同需求,对拿到的数据进行不同类型的分析,给出相应的结果或进行相应的处理.

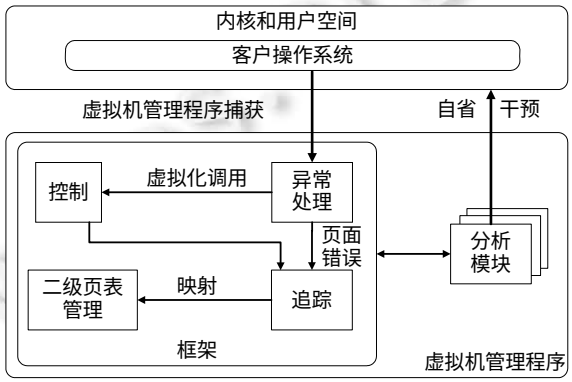

Fig.11 System architecture of control-flow tracing framework

图 11 控制流追踪框架的系统架构

4.2.1 执行流程

在系统启动时,虚拟机管理程序将二级页表的页表项全部设置为不可执行.页表的大小均设置为最小值 4KB,这样可以让控制流的追踪更为精确.虚拟机管理程序随后初始化一个大小为 n 的空列表,用于保存当前的可执行页.其中,n 是可配置的参数,n 越小,表示当前可执行页的总数越小,对控制流的追踪会更为精确;但同时,性能的开销也会更大.因此,通过对控制流分析的精确度和性能的综合考虑,使用者可以配置一个合适的 n 值进行分析.

在系统运行期间,系统的控制流会在页面间进行切换.当页面切换到一个不在可执行页面列表中的页面时,会产生页面错误(预取错误).此时,虚拟机管理程序会拦截该错误,并产生相关的控制流数据,将其发送给分析模块进行分析.

接下来,虚拟机管理程序会将该页设置为可执行,并且将该页加入可执行页面列表中.如果此时可执行页面列表中页面数目大于 n,则虚拟机管理程序依据不同的替换策略,将可执行页面列表中某个页面替换为该页面.

4.2.2 控制流数据

当产生页面错误时,虚拟机管理程序通过追踪模块的分析可以得到多种控制流数据.其中,可以被分析模块用于分析的数据有如下几种.

- 控制流目标:当页面错误产生时,页面错误产生的地址也会保存在相应的寄存器中.该寄存器中保存的是目标页面的虚拟地址,而虚拟机管理程序可根据虚拟地址,通过页表翻译得到对应的物理地址以及物理页;
- 控制流源:控制流源获得的精确程度取决于可执行页面列表的大小 n.由于只有可执行页面列表中的页是可执行的,因此控制流源存在于可执行页面列表中.当 n 越小,这个源就越精确.当 n=1 时,控制流源可

以精确到页.而对于某一种特定的情况来说,如果程序是由带链接分支指令执行的(例如 blx 或 bl),那么链接寄存器(link register,简称 LR)中的值就是精确的源地址;

- 客户上下文:当程序陷入到虚拟机管理程序中时,虚拟机管理程序可以读取到地址空间标识符(address space identifier,简称 ASID)以及进程标识符(process identifier,简称 PID);
- 客户执行状态:当页面中断产生时,虚拟机管理程序可以拿到很多数据,其中包括当前的程序计数器、当前程序状态寄存器(current program status register,简称 CPSR)及堆栈指针等.

4.2.3 基于页表哈希的控制流保护

该框架中的分析模块是相对独立的.开发者可根据需求对收到的控制流数据进行不同种类的分析,从而给出相应的结果.分析的结果可以是系统的运行情况或相关数据,也可以是针对不同情况进行的不同控制或操作.作者给出了一个依据控制流数据进行控制流保护的分析模块的示例.

该分析模块使用白名单机制,它会提前将正常控制流可能的页表跳转的源和目标页通过哈希表的形式连接在一起.当客户操作系统由于页面错误进入虚拟机管理程序时,分析模块会分别对源和目标页进行哈希,随后,通过目标页的哈希找到对应项,再从对应项的链表中寻找对应的源哈希.如果存在对应源哈希,则说明该控制流是正常的,否则说明该控制流是非法的,虚拟机管理程序会返回异常,并中止该控制流,从而保护系统的安全运行.

4.3 XNPro

由于 rootkit 是针对操作系统内核的攻击,而且与操作系统内核拥有相同的权限,因此,在内核中直接抵御 rootkit 攻击比较困难.硬件辅助虚拟化技术可以让虚拟机管理程序在更高权限模式下对内核进行保护,因此,很多人都使用虚拟化技术抵御 rootkit 攻击.Nordholz 等人[58]提出了 XNPro,通过对内核代码执行的严格限制来保证操作系统免于受到代码注入攻击.

4.3.1 工作原理

在正常的操作系统中,在用户模式下运行的程序,无法执行内核内存中的代码;而在管理模式下运行的程序,可以执行用户内存中的代码.而在管理模式下运行时,程序并没有必要执行用户内存中的代码,这种权限只会让被攻击的内核跳转到用户内存中执行一段攻击者编写的代码,并导致系统被攻击.因此,XNPro 对于可执行的代码做了严格的限制,当程序在管理模式下运行时,只有内核中的必要部分(如.text 区域等)被允许执行.

当程序在用户模式下运行时,只有用户模式的内存被设置为可执行并且可写,而内核的内存空间被设置为不可访问.当程序在管理模式下运行时,则只有内核的.text 区域等必要部分被设置为可执行而不可写(防止被恶意篡改),而其他部分被设置为可写而不可执行.这种设计同样可以防止被攻击的内核跳转到内核中某处攻击者编写的代码中.

4.3.2 加载模块的验证

上述机制保护了内核中的固定组件.与此同时,内核中还有一些可加载内核模块.这些合法的内核模块在内核中加载后,同样需要可执行权限.XNPro 采用白名单方式管理这些可加载内核模块.XNPro 预先保存了这些内核模块的哈希值,当这些内核模块被加载时,操作系统会得到这些模块的加载位置,并将其发送给虚拟机管理程序.XNPro 收到消息后,计算对应位置模块的哈希值,并在白名单中寻找对应项.如果白名单中存在相同的哈希值,说明该模块是经过验证的合法模块;否则说明该模块是恶意模块或未验证的可疑模块,XNPro 便会阻止该模块的执行.

5 未来研究展望

随着 ARM 硬件虚拟化扩展的发展,基于硬件辅助虚拟化技术的安全防护技术得到业界的高度重视.由于虚拟机管理程序拥有比客户操作系统更高的权限,因此,多数研究致力于保护系统安全,保护内核完整性.然而,移动平台的安全需求不仅仅限于内核保护.由于移动平台在用户的生活中占有越来越重的地位,移动设备中也存储了更多的用户隐私.因此,硬件虚拟化扩展技术可以用于更多样的安全防护中.总的来说,对于基于 ARM 虚

拟化扩展的安全防护工作,其未来的研究方向可能主要包括以下几个方面.

(1) 系统安全

虽然基于硬件辅助虚拟化技术的系统安全已有不少研究成果,但系统安全依然是一个研究热点.对于虚拟机管理程序来说,语义鸿沟的问题是阻碍虚拟机管理程序保护系统安全的重要问题[59].虚拟机管理程序无法知道操作系统中的关键数据结构,只能通过读取虚拟机内存来分析操作系统状态.如果虚拟机管理程序可以克服语义鸿沟障碍,从而准确知道操作系统状态,那么虚拟机管理程序可以根据设计者的需求保护操作系统的各个方面.此外,虚拟机自省(virtual machine introspection,简称 VMI)技术也是一种有效的分析虚拟机内部信息从而保护系统安全的手段,该技术同样可以移植到 ARM 平台上,用于保护客户操作系统的完整性和安全性.

(2) 隐私保护

随着人们对移动设备的日益依赖,个人和企业的许多隐私信息大量存储在随身的手机、平板以及其他智能设备中.因此,移动设备中的隐私信息成为安全防护的重点.这些信息在移动设备中的输入、传输、保存与读取,均由系统内核以及相关驱动完成的.因此,必须保证这些环节的安全性才能保护用户的隐私.当前,已有许多关于移动平台隐私保护的研究成果[60−62],这些工具均运行在管理模式或用户模式中,一旦系统内核被攻击者攻击,这些保护均会失效.而硬件虚拟化技术可以将虚拟机管理程序架设在虚拟化模式中,该模式拥有高于系统管理模式的权限,同时,对于操作系统来说是完全透明的.因此,在这一层拦截相关的操作,同时对隐私数据进行加密解密,可以有效地保护个人隐私数据.我们的最新工作 H-Binder[63]就利用虚拟化管理程序拦截 Binder 通信数据并进行相关的检查验证,来保护用户 Binder 数据的安全性和完整性.

(3) 可信用户接口

移动设备是由人来进行操作的,这种操作需要用户通过相关的外设实现(如触摸屏、相关按键等).对于攻击者来说,远程控制的操作只能通过控制驱动来实现,外设无法得到响应.虚拟机管理程序可以通过是否存在硬件中断,从而分辨出某个操作是通过直接操作外设还是通过远程控制来实现的.同时,虚拟机管理程序也需要给用户提供一个交互接口,以便从用户处得到指令来决定实现哪种保护.在 x86 平台上,Cheng 等人[64]通过硬件虚拟化技术实现了安全的密码输入,这种技术可以移植到 ARM 平台上实现虚拟机管理程序与用户的交互.

(4) 与安全扩展的结合

在 ARMv7 架构中,同时也包含了安全扩展(security extension)[65].安全扩展又称为可信域(TrustZone),它是一种 ARM 上的硬件安全技术.目前,许多研究者已经针对 ARM 安全扩展研究移动平台的安全防护方法[66−70].ARM 安全扩展和 ARM 虚拟化扩展有不同的特点.ARM 安全扩展基于硬件进行数据保护,其安全性比虚拟化扩展更好.而虚拟化扩展对于异常的拦截更为全面和灵活,同时在模式切换时拥有更好的效率.如果将二者结合在一起,使用安全扩展保存相关的安全密钥以及加密、校验程序,同时利用虚拟化扩展运行虚拟机管理程序对实时的操作系统异常进行拦截处理,便可以同时利用二者的优点,建立一个更为安全高效的系统安全防护体系.

6 结 论

本文主要对当前基于 ARM 虚拟化扩展的安全防护技术的研究进展进行了归纳与综述.早期的 ARM 虚拟化研究由于缺少硬件虚拟化的支持,主要使用全虚拟化与半虚拟化技术.在 ARM 发布了硬件虚拟化扩展之后,人们的研究重心逐渐转向 ARM 硬件虚拟化.ARM 虚拟化扩展提供了新的虚拟化模式,当虚拟机管理程序在虚拟化模式中运行时,拥有比操作系统更高的权限,从而可以更好地管理客户操作系统.在硬件辅助虚拟化的研究中,主要存在 3 种框架:Xen,KVM 以及 OKL4.研究者们也基于硬件辅助虚拟化技术在系统安全的研究中取得了很大进展.最后,基于硬件辅助虚拟化技术的安全研究现状,本文对基于 ARM 虚拟化扩展的安全防护技术的未来发展方向进行了展望.本文认为:基于 ARM 硬件虚拟化,并结合 ARM 安全扩展,研究人员可以在系统安全、隐私保护及可信接口上进行更多有意义的尝试.

References:

[1] Smith B. ARM and Intel battle over the mobile chip's future. Computer, 2008,41(5):15−18. [doi: 10.1109/MC.2008.142]

[2] Aroca RV, Goncalev LMG. Towards green data centers: A comparison of x86 and ARM architectures power efficiency. Journal of Parallel & Distributed Computing, 2012,72(12):1770−1780. [doi: 10.1016/j.jpdc.2012.08.005]

[3] Ou Z, Pang B, Deng Y, Nurminen JK. Energy- and cost-efficiency analysis of ARM-based clusters. In: Proc. of the IEEE/ACM Int'l Symp. on Cluster, Cloud and Grid Computing. 2012. 115−123. [doi: 10.1109/CCGrid.2012.84]

[4] You DH, Noh BN. Android platform based Linux kernel rootkit. In: Proc. of the Int'l Conf. on Malicious & Unwanted Software. 2011. 79−87. [doi: 10.1109/MALWARE.2011.6112330]

[5] Li WX, Wang JB, Mu DJ, Yuan Y. Survey on android rootkit. Microprocessors, 2011,32(2):68−72 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-2279.2011.02.020]

[6] Mulliner C, Robertson W, Kirda E. VirtualSwndle: An automated attack against in-app billing on Android. In: Proc. of the ACM Asia Conf. on Computer and Communications Security. 2014. 459−470. [doi: 10.1145/2590296.2590335]

[7] Arzt S, Huber S, Rasthofer S, Bodden E. Denial-of-App attack: Inhibiting the installation of Android apps on stock phones. In: Proc. of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices. 2014. 21−26. [doi: 10.1145/2666620.2666621]

[8] Spaulding J, Krauss A, Srinivasan A. Expoloring an open wifi detection vulnerability as a malware attack vector on iOS devices. In: Proc. of the Int'l Conf. on Malicious and Unwanted Software. 2012. 87−93. [doi: 10.1109/MALWARE.2012.6461013]

[9] Uhlig R, Neiger G, Rodgers D, Santoni AL, Martins FCM, Anderson AV, Bennett SM, Kägi A, Leung FH, Smith L. Intel virtualization technology. Computer, 2005,38(5):48−56. [doi: 10.1109/MC.2005.163]

[10] Chen X, Garfinkel T, Lewis EC, Subrahmanyam P, Waldspurger CA, Boneh D, Dwoskin J, Ports DRK. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. In: Proc. of the Int'l Conf. on Architectural Support for Programming Languages & Operating Systems. 2008. 2−13. [doi: 10.1145/1346281.1346284]

[11] Hofmann OS, Kim S, Dunn AM, Lee MZ, Witchel E. InkTag: Secure applications on an untrusted operating system. In: Proc. of the Int'l Conf. on Architectural Support for Programming Languages & Operating Systems. 2013. 253−264. [doi: 10.1145/2451116.2451146]

[12] Zhou ZW, Gligor VD, Newsome J, McCune JM. Building verifiable trusted path on commodity x86 computers. In: Proc. of the IEEE Symp. on Security and Privacy. 2012. 616−630. [doi: 10.1109/SP.2012.42]

[13] Cheng YQ, Ding XH, Deng RH. Efficient virtualization-based application protection against untrusted operating system. In: Proc. of the 10th ACM Symp. on Information, Computer and Communications Security. 2015. 345−356. [doi: 10.1145/2714576.2714618]

[14] ARM. Architecture Reference Manual (ARMv7-A and ARMv7-R edition). USA: ARM, 2012.

[15] Belady L. A study of replacement algorithms for virtual storage computers. IBM Systems Journal, 1966,5(2):78−101. [doi: 10.1147/sj.52.0078]

[16] Seawright LH, Mackinnon RA. VM/370—A study of multiplicity and usefulness. IBM Systems Journal, 1979,18(1):4−17. [doi: 10.1147/sj.181.0004]

[17] Creasy RJ. The origin of the VM/370 time-sharing system. IBM Journal of Research & Development, 1981,25(5):483−490. [doi: 10.1147/rd.255.0483]

[18] Gum PH. System/370 extended architecture: Facilities for virtual machines. IBM Journal of Research & Development, 1983,27(6):530−544. [doi: 10.1147/rd.276.0530]

[19] Garfinkel T, Ptaff B, Chow J, Rosenblum M, Boneh D. Terra: A virtual machine-based platform for trusted computing. ACM Sigops Operating Systems Review, 2003,37(5):193−206. [doi: 10.1145/1165389.945464]

[20] Huang JB, Ding Y, Fang F. Virtualization and cloud computing. In: Proc. of the Asia-Pacific Conf. on Information Network and Digital Content Security. 2011. 83−86 (in Chinese with English abstract).

[21] Zeng S, Hao Q. Network I/O path analysis in the kernel-based virtual machine environment through tracing. In: Proc. of the Int'l Conf. on Information Science and Engineering. 2009. 2658−2661. [doi: 10.1109/ICISE.2009.776]

[22] Wang J, Niphadkar S, Stavrou A, Ghosh AK. A virtualization architecture for in-depth kernel isolation. In: Proc. of the 43th Hawaii Int'l Conf. on System Sciences. 2010. 1−10. [doi: 10.1109/HICSS.2010.41]

[23] Whitaker A, Shaw M, Gribble SD. Scale and performance in the Denali isolation kernel. ACM SIGOPS Operating Systems Review, 2002,36(SI):195−209. [doi: 10.1145/844128.844147]

[24] Perez R, Doom LV, Sailer R. Virtualization and hardware-based security. IEEE Security & Privacy, 2008,6(5):24−31. [doi: 10.1109/MSP.2008.135]

[25] Wisniewski RW, Inglett T, Keppel P, Murty R, Riesen R. mOS: An architecture for extreme-scale operating systems. In: Proc. of the 4th Int'l Workshop on Runtime and Operating Systems for Supercomputers. 2014. 1−8. [doi: 10.1145/2612262.2612263]

[26] Yu KL, Chen Y, Mao JJ, Zhang L. ARM-MuxOS: A system architecure to support multiple operating systems on single mobile device. Computer Science, 2014,41(10):7−11 (in Chinese with English abstract). [doi: 10.11896/j.issn.1002-137X.2014.10.002]

[27] Nanda S, Chiueh TC. A survey on virtualization technologies. RPE Report, 2005. 1−42.

[28] Smith JE, Nair R. The architecture of virtual machines. Computer, 2005,38(5):32−38. [doi: 10.1109/MC.2005.173]

[29] Rosenblum M, Garfinkel T. Virtual machine monitors: Current technology and future trends. Computer, 2005,38(5):39−47. [doi: 10.1109/MC.2005.176]

[30] Popek GJ, Goldberg RP. Formal requirements for virtualizable third generation architectures. Communications of the ACM, 1974, 17(7):412−421. [doi: 10.1145/361011.361073]

[31] Sites RL, Chernoff A, Kirk MB, Marks MP, Robinson SG. Binary translation. Communications of the ACM, 1993,36(2):69−81. [doi: 10.1145/151220.151227]

[32] Suzuki A, Oikawa S. Implementing a simple trap and emulate VMM for the ARM architecture. In: Proc. of the IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications. 2011. 371−379. [doi: 10.1109/RTCSA.2011.26]

[33] Bellard F. QEMU, a fast and portable dynamic translator. In: Proc. of the Freenix Track: 2005 Usenix Technical Conf. 2005. 41−46.

[34] Bartholomew D. QEMU: A multihost multitarget emulator. Linux Journal, 2006,2006(145):68−71.

[35] Oh SC, Kim KH, Koh KW, Ahn CW. ViMo (virtualization for mobile): A virtual machine monitor supporting full virtualization for ARM mobile systems. In: Proc. of the 1st Int'l Conf. on Cloud Computing, GRIDs, and Virtualization. 2010. 48−53.

[36] Hwang JY, Suh SB, Heo SK, Park CJ, Ryu JM, Park SY, Kim CR. Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones. In: Proc. of the 5th IEEE Conf. on Consumer Communications and Networking Conf. 2008. 257−261. [doi: 10.1109/ccnc08.2007.64]

[37] Dall C, Nieh J. KVM for ARM. In: Proc. of the Ottawa Linux Symp. 2010. 45−56.

[38] Heiser G, Leslie B. The OKL4 microvisor: Convergence point of microkernels and hypervisors. In: Proc. of the ACM SIGCOMM Asia-Pacific Workshop on Systems (APSYS 2010). 2010. 19−24. [doi: 10.1145/1851276.1851282]

[39] Lee SM, Suh SB, Jeong B, Mo S. A multi-layer mandatory access control mechanism for mobile devices based on virtualization. In: Proc. of the Consumer Communications and Networking Conf. 2008. 251−256. [doi: 10.1109/ccnc08.2007.63]

[40] Park M, Yoo SH, Yoo C. Real-Time operating system virtualization for xen-arm. In: Proc. of the 4th Int'l Symp. on Embedded Technology. 2009. 1−2.

[41] Liu CL, Layland JW. Scheduling algorithms for muliprogramming in a hard-real-time environment. Journal of the ACM, 2002, 20(1):46−61. [doi: 10.1145/321738.321743]

[42] Li DJ. Multi-Platform extension of lightweight virtual machines [MS. Thesis]. Wuhan: Huazhong University of Science and Technology, 2011 (in Chinese with English abstract). [doi: 10.7666/d.d188307]

[43] Rossier D. EmbeddedXEN: A revisited architecture of the Xen hypervisor to support ARM-based embedded virtualization. White Paper, 2012.

[44] Zhong MZ. Study of embedded system security assurance based on virtualization technology [MS. Thesis]. Tianjin: Nankai University, 2013 (in Chinese with English abstract).

[45] Yang YJ, Gao YW. Study of direct access mapping of image files in Xen virtualized environment. Chines High Technology Letters, 2012,22(5):483−489 (in Chinese with English abstract). [doi: 10.3772/j.issn.1002-0470.2012.05.006]

[46] Zhao YH. Study of Linux kernel virtual machine technology implemented on ARM platform [MS. Thesis]. Wuhan: Huazhong University of Science and Technology, 2011 (in Chinese with English abstract). [doi: 10.7666/d.d187115]

[47] Ding JH, Lin CJ, Chang PH, Tsang CH, Hsu WC, Chung YC. ARMvisor: System virtualization for ARM. In: Proc. of the Ottawa Linux Symp. 2012. 95−109.

[48] Dall C, Nieh J. KVM/ARM: The design and implementation of the Linux ARM hypervisor. In: Proc. of the 19th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2014. 333−348. [doi: 10.1145/2541940.2541946]

[49] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A. Xen and the art of virtualization. ACM Sigops Operating System Review, 2003,37(5):164−177. [doi: 10.1145/1165389.945462]

[50] Lengyel TK, Kittel T, Pfoh J, Eckert C. Multi-Tiered security architecture for ARM via the virtualization and security extensions. In: Proc. of the Security in Highly Connected IT Systems. 2014. 308−312. [doi: 10.1109/DEXA.2014.68]

[51] Kivity A, Kamay Y, Laor D, Lublin U, Liguori A. KVM: The Linux virtual machine monitor. In: Proc. of the Linux Symp. 2007. 225−230.

[52] Russel R. Virtio: Towards a de-facto standard for virtual I/O devices. ACM SIGOPS Operating System Review, 2008,42(5): 95−103. [doi: 10.1145/1400097.1400108]

[53] Paolino M, Rigo A, Spyridakis A, Fanguède J, Lalov P, Raho D. T-KVM: A trusted architecture for KVM ARM v7 and v8 virtual machines securing virtual machines by means of KVM, TrustZone, TEE and SELinux. In: Proc. of the 6th Int'l Conf. on Cloud Computing, GRIDs, and Virtualization. 2015. 39−45.

[54] Liedtke J. On μ-kernel construction. In: Proc. of the 15th ACM Symp. on Operating System Principles. 1995. 237−250. [doi: 10.1145/224056.224075]

[55] Varanasi P. Implementing hardware-supported virtualization in OKL4 on ARM [Bachelor Thesis]. Sydney: University of New South Wales, 2010. [doi: 10.1145/2103799.2103813]

[56] Yang Y, Qian ZJ, Huang H. A lightweight monitor for Android kernel protection. Computer Engineering, 2014,40(4):48−52 (in Chinese with English abstract). [doi: 10.3969/j.issn.1000-3428.2014.04.009]

[57] Horsch J, Wessel S. Transparent page-based kernel and user space execution tracing from a custom minimal ARM hypervisor. In: Proc. of the IEEE Int'l Conf. on Trust, Security and Privacy in Computing and Communications. 2015. 408−417. [doi: 10.1109/Trustcom.2015.401]

[58] Nordholz J, Vetter J, Peter M, Junker-Petschick M, Danisevskis J. XNPro: Low-Impact hypervisor-based execution prevention on ARM. In: Proc. of the Int'l Workshop on Trustworthy Embedded Devices. 2015. 55−64. [doi: 10.1145/2808414.2808415]

[59] Chen PM, Noble BD. When virtual is better than real. In: Proc. of the 8th Workshop on Hot Topics in Operating Systems. 2001. 133−138.

[60] Enck W, Gilbert P, Han S, Tendulkar V, Chun BG, Cox LP, Jung J, Mcdaniel P, Sheth AN. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. on Computer Systems, 2014,32(2):5:1−5:29. [doi: 10.1145/2619091]

[61] Lin JH, Ren W, Jia LL. The design and implementation of a system for privacy protection in Android. Netinfo Security, 2013,(7): 16−19 (in Chinese with English abstract). [doi: 10.3969/j.issn.1671-1122.2013.07.004]

[62] Jia P, He X, Liu L, Gu B, Fang Y. A framework for privacy information protection on Android. In: Proc. of the Int'l Workshop on Sensor, Peer-to-peer and Social Networks. 2015. 1127−1131. [doi: 10.1109/ICCNC.2015.7069508]

[63] Shen D, Zhang ZK, Ding XH, Li ZJ, Deng RH. H-Binder: A hardened binder framework on android systems. In: Proc. of the 12th EAI Int'l Conf. on Security and Privacy in Communication Networks. 2016. 24−43.

[64] Cheng Y, Ding X. Virtualization based password protection against malware in untrusted operating systems. In: Proc. of the Int'l Conf. on Trust and Trustworthy Computing. 2012. 201−218. [doi: 10.1007/978-3-642-30921-2_12]

[65] ARM. ARM security technology—Building a secure system using trustzone technology. ARM Technical White Paper, 2009.

[66] Santos N, Raj H, Saroiu S, Wolman A. Using ARM trustzone to build a trusted language runtime for mobile applications. In: Proc. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2014. 67−80. [doi: 10.1145/2541940.2541949]

[67] Azab AM, Ning P, Shah J, Chen Q, Bhutkar R, Ganesh G, Ma J, Shen W. Hypervision across worlds: real-time kernel protection from the ARM trustzone secure world. In: Proc. of the ACM Sigsac Conf. on Computer and Communications Security. 2014. 90−102. [doi: 10.1145/2660267.2660350]

[68] Zhao S, Zhang Q, Hu G, Qin Y, Feng D. Providing root of trust for ARM trustzone using on-chip SRAM. In: Proc. of the 4th Int'l Workshop on Trustworthy Embedded Devices. 2014. 25−36. [doi: 10.1145/2666141.2666145]

[69] Pinto S, Oliveira D, Pereira J, Cardoso N, Ekpanyapong M, Cabral J, Tavares A. Towards a lightweight embedded virtualization architecture exploiting ARM TrustZone. In: Proc. of the IEEE Int'l Conf. on Emerging Technologies and Factory Automation. 2014. 1−4. [doi: 10.1109/ETFA.2014.7005255]

[70] Sun H, Sun K, Wang Y, Jing J. TrustOTP: Transforming smartphones into secure one-time password tokens. In: Proc. of the ACM Sigsac Conf. on Computer and Communications Security. 2015. 976−988. [doi: 10.1145/2810103.2813692]

附中文参考文献:

[5] 李文新,王姜博,慕德俊,袁源.Android 系统 Rootkit 技术综述.微处理机,2011,32(2):68−72. [doi: 10.3969/j.issn.1002-2279.2011.02.020]

[20] 黄建波,丁扬,方芳.虚拟化与云计算.见:亚太信息网络与数字安全会议论文集.2011.83−86.

[26] 余宽隆,陈瑜,茅俊杰,张磊.ARM-MuxOS:一台手机,多个世界.计算机科学,2014,41(10):7−11. [doi: 10.11896/j.issn.1002-137X.2014.10.002]

[42] 李大江.轻量级虚拟机的多平台扩展[硕士学位论文].武汉:华中科技大学,2011. [doi: 10.7666/d.d188307]

[44] 钟木忠.基于虚拟化技术的嵌入式系统安全保证研究[硕士学位论文].天津:南开大学,2013.

[45] 杨亚军,高云伟.Xen 虚拟化环境中镜像文件的访问直接映射研究.高技术通讯,2012,22(5):483−489. [doi: 10.3772/j.issn.1002-0470.2012.05.006]

[46] 赵亚辉.ARM 平台上实现 Linux 内核虚拟机技术研究[硕士学位论文].武汉:华中科技大学,2011. [doi: 10.7666/d.d187115]

[56] 杨永,钱振江,黄皓.一种轻量级的 Android 内核保护监控器.计算机工程,2014,40(4):48−52. [doi: 10.3969/j.issn.1000-3428.2014.04.009]

[61] 林佳华,任伟,贾磊雷.Android 手机隐私保护系统的设计与实现.信息网络安全,2013(7):16−19. [doi: 10.3969/j.issn.1671-1122.2013.07.004]

李舟军(1963－),男,湖南湘乡人,博士,教授,博士生导师,主要研究领域为网络与信息安全,数据挖掘,智能信息处理.

苏晓菁(1989－),女,助理研究员,主要研究领域为高可靠性器件与集成技术,先进计算光刻技术.

沈东(1989－),男,博士生,CCF 学生会员,主要研究领域为移动安全,虚拟化安全.

马金鑫(1986－),男,博士,助理研究员,主要研究领域为软件安全,程序分析.