软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(8):2124−2137 [doi: 10.13328/j.cnki.jos.004684]

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

基于虚拟机监控器的隐私透明保护*

任建宝, 齐 勇, 戴月华, 王晓光, 宣 宇, 史 樾

(西安交通大学 计算机科学与技术系,陕西 西安 710049)

通讯作者: 史樾, E-mail: shiyi@mail.xjtu.edu.cn

摘 要: 操作系统漏洞经常被攻击者利用,从而以内核权限执行任意代码(返回用户态攻击,ret2user)以及窃取用户隐私数据.使用虚拟机监控器构建了一个对操作系统及应用程序透明的内存访问审查机制,提出了一种低性能开销并且无法被绕过的内存页面使用信息实时跟踪策略;结合安全加载器,保证了动态链接库以及应用程序的代码完整性.能够确保即使操作系统内核被攻击,应用程序的内存隐私数据依然无法被窃取.在 Linux 操作系统上进行了原型实现及验证,实验结果表明,该隐私保护机制对大多数应用只带来 6%~10%的性能负载.

关键词: 隐私保护;虚拟机监控器;嵌套页表;代码完整性

中图法分类号: TP316

中文引用格式: 任建宝,齐勇,戴月华,王晓光,宣宇,史樾.基于虚拟机监控器的隐私透明保护.软件学报,2015,26(8):2124−2137. http://www.jos.org.cn/1000-9825/4684.htm

英文引用格式: Ren JB, Qi Y, Dai YH, Wang XG, Xuan Y, Shi Y. Transparent privacy protection based on virtual machine monitor. Ruan Jian Xue Bao/Journal of Software, 2015,26(8):2124−2137 (in Chinese). http://www.jos.org.cn/1000-9825/4684.htm

Transparent Privacy Protection Based on Virtual Machine Monitor

REN Jian-Bao, QI Yong, DAI Yue-Hua, WANG Xiao-Guang, XUAN Yu, SHI Yi

(Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: The vulnerabilities of OS kernel are usually exploited by attackers to execute arbitrary code with kernel privilege (i.e., return-to-user attacks, ret2user) and to steal other processes' private data. In this paper, a transparent OS kernel memory access mediator based on VMM (virtual machine monitor) is proposed, and a non-bypassable low performance overhead memory page tracker is provided to get the memory usage information in real-time. Combined with a safe loader, the new method guarantees the code integrity of dynamic shared objects during run-time. It also ensure that, even when the OS kernel is compromised, the application's memory private data is still safe. A prototype is implemented on the Linux OS, and the evaluation experiments show that it only incurs about 6%~10% performance overhead for most SPEC benchmark tests.

Key words: privacy protection; virtual machine monitor; nested page table; code integrity

操作系统内核庞大而复杂,使其很容易被攻击[1−3].由于处理器硬件固有架构设计,从高特权级到低特权级(操作系统内核态到用户态)可以进行直接的控制流转换以及存储访问.因此,当操作系统漏洞被成功利用后,攻击者可以轻易地以内核权限执行攻击代码(返回用户态攻击,ret2user),窃取用户内存隐私数据等等.另一方面,操作系统的复杂性导致现有方法均无法全面地保护其不受攻击[4].因此,当操作系统受到攻击后,如何保证用户的隐私数据不被攻击者窃取,是一个亟待解决的问题.

针对用户隐私数据保护,国内外学者做了很多工作.一些学者通过控制流以及数据流完整性保护来保证用

* 基金项目: 国家自然科学基金(60933003); 国家高技术研究发展计划(863)(2012AA0109 04); 教育部高等学校博士学科点专项科研基金(20120201110010)

收稿时间: 2014-02-28; 修改时间: 2014-07-07; 定稿时间: 2014-07-31

户数据的安全性[5,6],然而这类方法带来的高性能负载使得其难以实际应用.此外,还有一些学者采用虚拟机监控器对应用程序进行保护[7−13].这类方法需要对操作系统以及应用程序进行修改、定制或者重编译,无法达到透明保护的目的,使得其不能应用到一些闭源软件上,限制了这类保护机制的实用性.同时,也有学者通过对现有CPU 架构进行修改[14−17]来保护用户隐私,这类保护方法由于需要依赖特殊的硬件架构,难以被普通用户采用.具体的相关工作对比我们将在第 4 节进行介绍.

本文提出一种基于虚拟机监控器对操作系统及应用程序透明的用户内存隐私数据保护方法,该方法能够在应用程序生命周期中,即使操作系统被攻击后,保证其内存隐私数据依然无法被攻击者窃取.该方法主要提供3 个方面的防护:(1) 保证操作系统只能访问应用程序显式或隐式提供的数据,比如系统调用参数等;(2) 保证攻击者注入的恶意代码无法以内核权限执行,同时保证被攻击的操作系统无法向被保护程序注入恶意代码;(3) 保护运行时动态链接库代码的完整性,避免利用动态链接库来窃取用户隐私.本文的主要出发点基于我们的一个观察结论:上述安全问题(ret2user 以及隐私窃取)主要源于从内核空间到用户空间没有任何限制的控制流转换以及内存访问.

本文的主要贡献有:(1) 提出了一种对操作系统以及应用程序透明的隐私保护策略;(2) 引入一种高效的、无法绕过的内存页面使用跟踪机制,结合上下文切换监视器,可以有效地防止 ret2user 攻击并保护应用程序内存隐私数据不被攻击者窃取;(3) 在硬件平台上进行了全功能的系统原型实现,验证了所提出方法的有效性及高效性.目前,我们在 AMD 平台上的原型系统测试显示:对于大多数应用程序,该保护方法所带来的性能开销在6%~ 10%左右.

本文第 1 节对 ret2user 攻击以及内存隐私数据泄露进行概述,同时给出本文的前提及假设条件.第 2 节主要陈述本文所提保护机制的设计及实现.第 3 节对实现的原型系统进行分析评估.第 4 节主要与现有研究工作进行比较.第 5 节总结全文并展望未来的工作.

1 背景概述

处理器的特权级设计提供了一个单向的保护机制,即,高特权级可以直接访问低特权级的资源;反之则不然.操作系统通过利用这种硬件保护机制来隔离操作系统,保证应用程序无法破坏操作系统内核.然而,该保护机制具有先天不足,如果操作系统内核漏洞被利用,攻击者则可以毫无限制地访问应用程序空间,窃取用户隐私.同时,攻击者利用内核特权级对其攻击痕迹进行掩饰,避免被安全软件检测.本节中,我们对这些攻击方式进行概述,并给出本文的攻击模型与假设条件.

1.1 ret2user攻击概述

攻击者通过使用处理器单向访问保护机制,同时结合操作系统内核漏洞来完成该攻击.攻击者通过操作系统内核漏洞传递一个精心准备的参数,从而实现对操作系统内核控制数据的篡改(例如函数返回地址、函数指针等).当操作系统再对这些篡改后的数据进行引用时,控制流就会被转换到攻击者事先设置好的攻击代码片段.因为此时处理器依旧处于内核态,所以攻击代码可以在被攻击计算机上执行任何操作,比如提高攻击者的账户权限、查看其他应用程序隐私数据等.

图 1 为一个利用 Linux 操作系统内核漏洞从而获得系统根权限的攻击代码片段,其总共利用了 3 个内核漏洞,对 Linux 2.6.36.2 之前的所有内核版本均有效.在进行‘clone’系统调用时传进一个经过特殊准备的内存地址(代码片段第 7 行中的 *target* 参数),如果线程创建时使用了‘CLONE_CHILD_CLEARTID’标志,当子线程结束后,操作系统调用 *put_user*()函数擦除该地址所指向的内存单元.操作系统在使用 *put_user*()函数清除该内存单元前会首先调用‘*access_ok*()’函数对该内存地址进行校验,确保其是一个用户空间地址.

该攻击实例中,在操作系统通过‘*set_fs*(*get_ds*())’重置用户空间地址限制后,如果线程运行突然产生错误,那么‘*set_fs*(*old_fs*)’将不会被执行.这时,当操作系统强制终止该线程并且调用‘*put_user*()’进行相关清理工作时,由于用户空间地址限制值没有被改回,操作系统判断任何地址值都是用户空间,从而导致之前‘*target*’参数所指向的内存单元被清零.而在该攻击中,‘*target*’参数指向的内存单元存放的是内核的一个函数指针,这样,该函数指

针便被间接地修改.一般来说,修改后的函数指针会指向攻击者事先准备好的攻击代码地址,从而当操作系统通过该函数指针进行函数调用时,内核控制流就会转换到攻击代码中.

```
1
  /******* full-nelson.c **********/
3 target = econet_ops + 10 * sizeof(void *) - OFFSET;

5 clone((int (*)(void *))trigger,
    (void *)((unsigned long)newstack + 65536),
7   CLONE_VM | CLONE_CHILD_CLEARTID | SIGCHLD,
    &fildes, NULL, NULL, target);
9
  /******* linux/fs/splice.c **********/
11 static ssize_t kernel_write(struct file *file, ....)
  {
13    old_fs = get_fs();
      set_fs(get_ds()); // set the address limite to
           KERNEL_DS -1
15    res = vfs_write(...);  // a NULL pointer execption
           will happen
      set_fs(old_fs);     // will not be executed
17 }
```

Fig.1 Econet local root exploit

图 1 Econet 本地根攻击

1.2 隐私泄露概述

操作系统提供多种接口来实现对物理内存页面的访问,比如/dev/mem 设备文件以及 fmem 内核模块等.攻击者一旦通过某种手段提升权限(例如第 1.1 节所述的 ret2user 攻击)后,就可以通过这些接口随意地访问整个系统的物理内存页面.通过对物理内存页面内容的提取与分析,即可获得应用程序驻留在内存中的所有敏感数据.

除了通过操作系统提供的接口访问物理内存页面之外,恶意软件也可以通过修改页表从而将其他应用程序的物理页面映射到自己的虚拟地址空间中,这样,即使其之后失去内核权限也可以正常访问其他应用程序的内存页面.令人不安的是,“面向返回类型编程(return-oriented programming,简称 ROP)”[18]这种新类型的攻击手段使得这种隐私窃取方式更加容易实现.

此外,一些服务程序以及数据库程序往往被赋予过高的权限,进而导致攻击者利用这些程序的漏洞获得系统特权,使得攻击者可以轻易地窃取或篡改其他应用程序的内存隐私数据.

综上所述,无论是 ret2user 攻击还是内存隐私数据泄露,其根本原因在于,操作系统可以直接或间接的访问其他应用程序的内存页面.因此,如果能够有效地对操作系统的内存页面访问进行监控,即可成功地预防 ret2user 攻击以及保护应用程序的内存隐私数据.

1.3 攻击模型与假设

在本文中,我们使用一个很强的攻击模型,即,攻击者拥有系统内核权限.这意味着攻击者可以肆意的读取、篡改任意内存单元的数据,可以以内核权限执行任意的内存代码,能够随意地将物理内存页面映射到内核空间或是用户空间等.

本文所提出的隐私保护方法是直接的,即,验证操作系统对被保护程序内存页面的所有访问.为此,我们提出以下几点前提假设:(1) 整个计算机系统硬件以及固件是安全的,即,处理器、内存总线不被监听,BIOS 未被篡改等;(2) 被保护的应用程序自身不存在导致隐私泄露的漏洞;(3) 由于操作系统被攻击所导致的系统服务暂停以及系统宕机不在本文考虑范围.

2 系统设计与实现

从第 1 节的描述中可以看出,无论是 ret2user 攻击还是利用操作系统内核窃取用户隐私数据,其根本原因都在于操作系统具有较高的特权级,其可以无任何限制地访问任意的物理内存页面.因此,预防 ret2user 类型攻击

以及保护应用程序内存隐私数据的一个有效手段就是对操作系统的内存页面访问进行监控验证.与传统计算环境相比,在虚拟化环境下,整个计算机系统的管理由虚拟机监控器负责,操作系统不再会无限制地访问任意物理内存页面.因而,本文通过借助轻量级安全虚拟机监控器 OSV[19]将被保护应用程序所使用的物理内存页面与操作系统进行隔离,在每次操作系统访问被保护应用程序内存时,虚拟机监控器中的隐私保护组件对操作系统访问的合法性进行验证,保证每次操作系统对应用程序内存的访问与应用程序本身意图相符.即,应用程序通过系统调用等方式授权操作系统对其内存内容进行访问.

本节首先对基于虚拟机监控器的隐私保护系统进行概述;然后,分别从被保护应用程序安全启动环境的建立、运行时内存隐私数据保护、操作系统内存访问验证等方面对整个系统的设计及实现关键细节进行描述;最后,对实现过程中的一些其他细节问题进行描述,并提供相应的解决方案.

2.1 系统概述

如图 2 所示,虚线框中的 APP 表示需要被保护的应用程序.该隐私保护系统主要由 3 个组件组成:位于虚拟机监控器中的隐私保护器核心组件、安全加载器以及启动壳子.启动壳子(start shell)通过与位于虚拟机监控器中的隐私保护器进行交互来启动被保护应用程序,从而避免了对应用程序进行修改、重编译.在应用程序(ELF文件)启动时,操作系统均会调用加载器对应用程序所需要的动态链接库进行加载.通过将系统默认的加载器替换为安全加载器,进而获得应用程序使用的动态链接库信息,隐私保护器通过这些信息保证应用程序运行时所使用的动态链接库的代码完整性.在被保护应用程序运行阶段,隐私保护器实时地获取被保护程序的物理内存页面信息,当操作系统访问应用程序的内存页面时,隐私保护器对这些访问操作进行检查,如果该内存访问是被应用程序允许的(例如正常系统调用所使用的内存内容等),隐私保护器则将对应的内存内容复制至操作系统内核;如果被访问的内存内容不是应用程序所允许的,隐私保护器则首先将对应的内存内容进行加密,然后再将密文复制至操作系统内核.为了避免操作系统通过修改系统调用返回地址从而构造面向返回类型攻击,隐私保护器对每次应用程序与操作系统内核的空间切换进行监控,保证返回地址的正确性.整个系统的设计及实现难点在于如何透明地对应用程序所使用的内存页面进行实时的跟踪、监控操作系统以及应用程序之间交互以及有效的保护应用程序所使用的动态链接库.具体的设计及实现细节见下文.

Fig.2 Architecture of privacy protection

图 2 隐私保护系统框架

2.2 嵌套页表机制

本文所提出的隐私保护机制主要依赖于虚拟化技术中的嵌套页表机制.现代操作系统主要采用分页机制对系统内存进行管理.如图 3 中左图所示,操作系统使用页表将应用程序线性地址转换为实际的物理地址,这样,即使不同的应用程序使用相同的虚拟地址,也能保证它们实际使用的物理页面相互独立.在虚拟化环境下,应用程序线性地址到实际的物理地址之间又多了一层页表转换,这层页表被称为嵌套页表(nested page table,简称NPT).如图 3 中右图所示,应用程序的线性地址经过操作系统所维护的第 1 层页表转换为操作系统认为的物理地址(guest physical address),然后再经过虚拟机监控器维护的第 2 层页表转换为实际的机器地址(machine physical address).这种嵌套页表的地址转换机制保证了操作系统可见的同一物理地址实际对应不同的机器地

址.因此,对被保护应用程序与操作系统使用不同的 NPT 页表映射,即可保证当操作系统被攻击后,攻击者依然无法访问应用程序内存页面.

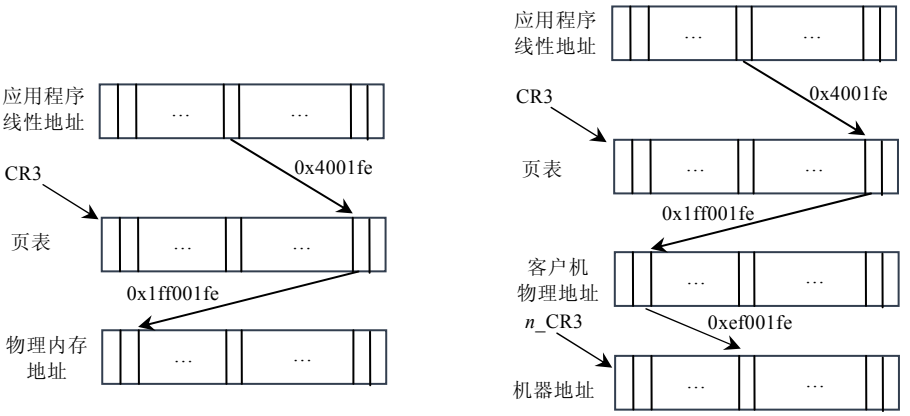


Fig.3 Address translation

图 3 地址转换

2.3 保护环境初始化

2.3.1 启动壳子

为了避免对被保护程序进行修改,我们通过启动壳子来启动被保护应用程序.启动壳子是一个自包含程序,其通过虚拟机调用与位于虚拟机监控器中的隐私保护核心组件进行直接交互.当隐私保护器接收到来自启动壳子的虚拟机调用时,其首先会对启动壳子的代码完整性进行校验,避免操作系统以及其他应用程序对启动壳子代码进行篡改,代码完整性检验的结果会通过直接写显存的方式显示在屏幕上.如果启动壳子代码未被修改,隐私保护器会在虚拟机监控器中为启动壳子创建一个单独的页表映射,使其与操作系统以及其他应用程序隔离.此外,等被保护应用程序结束运行后,启动壳子通过虚拟机调用通知隐私保护器进行相关的清理工作.

2.3.2 安全加载器

为了实现代码复用以及节省内存使用,应用程序经常使用动态链接库来实现程序内部的某些功能.操作系统启动应用程序时,首先会使用加载器对应用程序所使用的动态链接库进行加载,动态链接库每次加载到应用程序地址空间中的虚拟地址是变化的.因此,如何获得这些动态链接库加载信息并对动态链接库的代码完整性进行验证,是保护用户内存隐私数据的一个重要方面.现有研究工作为了回避动态链接库带来的问题,通常要求将被保护应用程序进行静态编译,这使得一些无法进行重新编译的闭源软件无法应用相应的隐私保护机制;同时,静态编译导致应用程序规模变得异常庞大,比如,一个简单的“hello world”程序动态编译时只有 6 889 字节,而静态编译时则为 784 679 字节.

为此,我们对 glibc[20]的 elf 加载器进行修改,设计并实现了一个安全加载器,使用安全加载器替代系统默认的加载器(64 位系统下默认加载器为/lib/x86_64-linux-gnu/ld-x.x.so).安全加载器在每次加载动态链接库时,记录加载的链接库名称以及对应加载到的虚拟地址和加载的长度.在加载器完成应用程序启动准备工作,将程序控制流跳转到应用程序之前,安全加载器通过获得的动态链接库加载信息对应用程序所使用的动态链接库的代码完整性进行验证.如果验证通过,加载器则通过虚拟机调用通知底层的隐私保护器对这些动态链接库所使用的页面进行标识.隐私保护器对底层的 NPT 页表项进行设置,将这些页面标识为只读页面,从而防止在应用程序运行阶段,动态链接库被非法篡改.

2.3.3 自保护机制

虚拟机监控器、隐私保护器、启动壳子以及安全加载器组成了整个系统的可信计算基(trust computing base,简称 TCB),这几个组件的安全完整性是保护应用程序隐私的前提.目前,我们采用轻量级安全虚拟机监控器

OSV[21,22]作为整个系统的基础,TPM[23]芯片保证了在系统启动时所加载的虚拟机监控器是安全、可靠的.当OSV 虚拟机监控器启动后,其对加载的操作系统内核代码进行校验,保证操作系统代码没有被恶意篡改.在每次启动壳子与安全加载器启动时,隐私保护器均会对这两个组件的代码完整性进行校验,防止隐私保护功能失效.每次代码校验的结构通过类似“超级盒子(powerbox)”的方式呈现给用户,防止被攻击的操作系统修改校验结果.在系统运行时,虚拟机监控器将其自身及隐私保护器所使用的物理页面从操作系统所使用的嵌套页表中进行屏蔽,保证虚拟机监控器以及隐私保护器不会被篡改.此外,我们已对 OSV 虚拟机监控器的安全性进行验证[21],从而保证整个系统计算基的可信性.

2.4 运行时保护

2.4.1 内存页面监控

(a) 内存隔离

隐私保护器为上层操作系统以及被保护的应用程序提供不同的 NPT 页面映射.如图 4 所示,被保护应用程序的 NPT 页面映射包含其自身所使用的内存页面,而对于操作系统内核以及其他应用程序所使用的内存页面则标记为不存在,对于动态链接库则表示为只读页面.相反地,操作系统内核所使用的NPT页面映射则将被保护应用程序所使用的内存页面标识为不存在.由于 NPT 页表的维护是由位于虚拟机监控器中的隐私保护组件负责的,所以上层操作系统无法感知底层物理页面是如何映射的.因此,操作系统依然以原始方式管理系统内存页面,而对于如何根据操作系统动态地为应用程序分配及回收内存页面进而调整相应的 NPT 映射,我们将在下一节进行介绍.通过使用NPT页表进行内存隔离,除了实现对操作系统透明以外,其亦可保证即使操作系统被攻击后,被保护应用程序与操作系统之间依然互相隔离.此外,从图中我们也可以看到,被保护应用程序与其他应用程序之间的内存空间也是相互隔离的,所以即使操作系统将被保护应用程序所使用的物理页面映射到其他恶意进程的地址空间中,其他进程依然无法访问被保护应用程序的内存页面.

Fig.4 Protected APP and OS kernel nested page table comparison

图 4 被保护程序与操作系统内核的嵌套页表对比

隐私保护器使用唯一的标识号对操作系统中的进程进行识别,目前,所有被保护进程使用同一个 NPT 页面映射表,操作系统以及其他非保护应用程序使用另一个 NPT 页面映射表.这意味着被保护进程之间可以顺利地进行进程间通信以及内存共享,而被保护进程与其他进程以及操作系统之间则是互相隔离的.

虽然内存隔离可以很好地防止在应用程序运行时操作系统以及其他应用程序窃取其隐私数据,然而仅仅进行内存隔离无法确保用户隐私不被泄露.例如,操作系统可以为被保护的应用程序分配一个包含恶意代码的页面,然后将程序的控制流跳转到该页面.因为包含恶意代码的页面与被保护应用程序使用同一个地址空间,因此底层隐私保护器无法阻止其对被保护应用程序内存敏感数据的访问.再如,操作系统通过构造面向返回类型攻击,从而利用被保护程序原有的代码窃取用户隐私数据.这些窃取用户隐私数据的攻击方式是上述内存隔离无法防护的,针对这些类型攻击的防御方法,我们将在第 2.4.2 节中进行介绍.

(b) 页面跟踪

上述内存隔离方法的前提是能够实时、准确地获得被保护应用程序所使用的物理内存页面,而应用程序在运行时会频繁地进行内存申请以及释放操作,因此隐私保护器必须实时精确并且不被绕过地获得应用程序运

行时动态的内存使用信息.对操作系统的内存分配及回收函数进行插桩是最直接的办法,当每次操作系统进行内存分配及释放操作时,插桩代码通过虚拟机调用显式地通知隐私保护器更新相应的 NPT 页表项.这种方法虽然能够实时地获得应用程序内存使用情况,但是插桩代码很容易被攻击后的操作系统绕过,进而使得隐私保护机制失效.另一方面,每次进行内存分配及释放操作都进行虚拟机调用,使得其他非保护应用程序在执行过程中也不可避免地陷入虚拟机监控器,从而带来不必要的性能损失.因此,为了避免插桩代码被绕过以及给其他应用程序带来不必要的性能损失,我们选择将页面跟踪机制在虚拟机监控器中进行实现.

从图 4 中可以看出,在被保护应用程序运行时,不属于该应用程序的内存页面在对应的 NPT 页表项中都标识为不存在.当操作系统在缺页处理中断中为应用程序分配内存页面后,虽然操作系统维护的页表映射中标识该内存页面已存在,但在底层 NPT 页表中对应的页表项仍然标识其对应的物理页面不存在.因此,当应用程序第一次访问操作系统为其分配的物理页面时,就会产生一个 NPT 页表错误.在 NPT 页表错误处理函数中,隐私保护器将对应的物理页面在应用程序所使用的 NPT 页表中改为已存在.相应地,在操作系统所使用的 NPT 页表中则标识为不存在.在更改完 NPT 页表后,隐私保护器将 TLB 缓存刷新,避免由于 TLB 中已存在相应的页面映射,使得操作系统依然可以访问已经为应用程序分配的物理页面.

由于应用程序每次访问一个新分配内存页面的时候均会产生一个 NPT 页表错误,因此上述方法能够确保实时获得应用程序内存使用信息并且不被绕过.然而当应用程序释放内存时,由于虚拟机监控器与上层操作系统之间的语义鸿沟,导致隐私保护器无法获得那些被应用程序释放的内存页面信息.如果操作系统将这些被保护应用程序释放的内存页面分配给其他进程使用而隐私保护器并未对相应的 NPT 页表进行调整,则会导致整个系统崩溃.虽然通过对操作系统进行插桩,从而显式地通知隐私保护器进行相应嵌套页表项调整,但是这样需要对现有操作系统内核进行修改,同时影响其他应用程序的性能.本文采用基于系统调用信息分析的内存释放策略,从而保证当被保护应用程序释放内存页面时,隐私保护器能够及时对相应的嵌套页表项进行调整,避免因页表项更改不及时而导致的系统崩溃问题.具体方法细节见第 2.4.2 节.

2.4.2 上下文切换

(a) NPT 页表选择

如何根据当前处理器所处的运行状态选择相应的 NPT 页表,是实现上述内存隐私保护方法的前提.目前,我们通过嵌套页表错误处理函数捕捉应用程序与操作系统之间的上下文切换.如图 4 所示,当处理器从被保护应用程序用户空间切换到内核空间时,由于应用程序所使用的 NPT 页表中内核代码均标识为不存在,因此处理器会产生一个嵌套页表错误,在 NPT 页表错误处理函数中,隐私保护器将当前处理器使用的 NPT 页表切换为操作系统内核所对应的页表.相应地,当从操作系统内核返回用户空间时,由于被保护程序代码在操作系统所使用的嵌套页表中标识为不存在,所以同样会产生一个嵌套页表错误.因此,通过对嵌套页表错误原因进行分析,即可成功捕捉应用程序与操作系统之间的上下文切换,进而选择合适的嵌套页表.

(b) 返回地址验证

在第 2.4.1 节中,我们提到被攻击的操作系统可以通过构造 ret2user 类型攻击以及向被保护应用程序注入恶意代码,从而达到窃取应用程序内存隐私数据的目的.一般来说,从操作系统返回到应用程序的返回地址为系统调用或是中断发生时指令的下一条指令地址.还有一种情况为应用程序通过“signal”系统调用向操作系统注册信号处理函数,当进程接受到注册过的信号时,操作系统就会将程序控制流转换到相应信号处理函数中.对于前一种情况,我们记录每次从应用程序进入操作系统时下条指令的虚拟地址,当从操作系统返回到应用程序空间时,对返回地址进行验证.对于后一种情况,我们记录每次“signal”系统调用所注册函数的虚拟地址,当操作系统将控制流从内核空间想用户空间转换时,如果进入用户空间的地址与之前记录的地址不一致,则与保存的信号处理函数地址进行对比:如果存在相应的地址,则将控制流进行正常的切换;如果不存在相应的地址,则说明被攻击后的操作系统试图窃取应用程序隐私.此时,隐私保护器向用户发出提示信息,并强制将进入用户空间的指令地址改为之前记录的地址.

因此,即使被攻击的操作系统向被保护应用程序注入恶意代码,隐私保护器也能保证程序控制流不会跳转

到恶意代码中.而对于 ret2user 类型攻击,由于每次从内核空间向用户空间进行切换时都会进行返回地址检查,因此攻击者也无法使用该类攻击获取用户隐私.

(c) 内存页面释放

上文中我们提到,由于虚拟机监控器与上层操作系统之间存在语义鸿沟,使得隐私保护器无法及时获得应用程序释放的内存信息,从而导致整个系统崩溃的问题.针对这个问题,我们通过对系统调用进行分析,从而获得被释放内存页面的信息.应用程序对内存的申请释放是通过库函数实现的,而底层库函数对应用程序申请释放内存的相应操作主要有两种方式:当应用程序申请的内存空间较小时,库函数通过调整应用程序堆大小来实现;当应用程序一次性申请大量内存空间时,库函数通过虚拟地址空间映射来实现.具体来说,库函数通过调用“brk”系统调用,对应用程序的堆大小进行调整.当应用程序申请内存时,“brk”系统调用向操作系统内核传进相应的参数,使得应用程序的堆增大;当应用程序需要释放内存空间时,“brk”系统调用向操作系统内核传进相应的参数,使得应用程序的堆减小.当操作系统发现应用程序减小堆大小时,操作系统就将介于新、旧堆大小之间的所有内存页面回收.当应用程序申请一大段内存空间时,底层库函数进行“mmap”系统调用,为应用程序分配新的虚拟地址区;而当应用程序释放这一大段内存空间时,底层库函数则进行“munmap”系统调用,操作系统据此回收相应的物理内存页面,并根据实际情况将内存修改的页面写入硬盘.

通过以上分析可知,只要能够对“brk”以及“munmap”系统调用进行捕获,就能获知应用程序对内存页面的释放操作.前文中我们已经介绍过,隐私保护器可以捕获每次应用程序与操作系统的上下文切换,因此我们只需要对上下文切换发生的具体原因进行分析,即可识别出系统调用.更进一步地,通过识别系统调用号,就可以顺利地捕获“brk”和“munmap”系统调用,进而根据系统调用的参数获得哪些内存页面被应用程序释放.

(d) 系统调用验证

从之前的描述可以看出,当应用程序运行时,操作系统对应用程序内存页面的访问会引起嵌套页表错误中断.如果在中断处理程序中不进行特殊处理直接拒绝操作系统进行的内存访问,则会导致应用程序正常的功能无法实现,比如 sys_write,sys_rename 等系统调用.另一方面,如果不对操作系统的内存访问进行分析,直接让操作系统访问应用程序内存,则会导致上述内存隔离变得毫无意义.因此,在嵌套页表中断处理函数中对操作系统的内存访问进行分析,是实现隐私保护的关键所在.

为了区分操作系统内存正常访问与非法访问,隐私保护器在每次系统调用时对系统调用的参数进行分析,记录每次系统调用所使用的内存地址范围.当操作系统访问应用程序内存页面而产生嵌套页表错误时,隐私保护其将产生错误的内存地址与之前记录的系统调用所应该访问的内存地址范围进行对比:如果两者一致,隐私保护器则将应用程序内存内容复制到操作系统内核空间;如果两者不一致,隐私保护器则将应用程序内存内容加密后,再将密文复制到操作系统内核空间.

2.5 其他问题

2.5.1 中断堆栈

在 64 位操作系统中,内核为每一个外部中断提供一个独立的中断栈.由于中断栈属于内核空间所有,所以在应用程序的嵌套页表中,中断栈页面所对应的页表项设置为不存在.当在被保护应用程序执行过程中,处理器接收到一个外部中断将相应的信息压栈时,就会产生一个嵌套页表错误异常.此时,处理器已经通过向 PIC 或 APIC 发送中断回复信号(ACK)获取中断类型及中断向量等信息,从而导致该中断在处理器中的状态已不是挂起状态.此时,如果从嵌套页表中断处理函数中直接返回上层操作系统,就会导致中断丢失问题.因此,在嵌套页表中断处理函数中,隐私保护器需要判断当嵌套页表错误异常发生时,上层操作系统是否处于中断处理过程中.如果处于中断处理过程中时,那么隐私保护器获得相应的中断处理向量以及相关标识信息,再通过中断注入的方式向上层操作系统注入相同的中断.当处理器从嵌套页表中断处理函数返回上层操作系统时,操作系统就会发现新的中断,从而避免了原始中断丢失.

2.5.2 页表页面

上文提到,当被保护应用程序第 1 次访问操作系统为其分配的物理页面时,就会产生一个嵌套页表错误,因

此保护器据此认为该物理页面属于被保护应用程序所有,每次操作系统访问该页面时,隐私保护器均会判断应用程序是否显示或隐式地允许操作系统进行访问.当被保护应用程序访问一个之前未被访问过的虚拟地址时,其使用的操作系统维护的地址转换页表中对应的某一级页表页面在嵌套页表中标识为不存在,此时就会在产生一个嵌套页表错误异常.根据之前的描述,隐私保护器判定该页面为应用程序所有,操作系统不能直接访问.而当应用程序进行系统调用时,操作系统内核需要使用相应的页表页面进行地址转换.由于之前隐私保护器已经判定该页面属于应用程序所有,所以操作系统就无法完成正常的地址转换操作以及相应页表页面中页表项的修改、填充操作.因此在嵌套页表错误处理函数中,不能简单地将发生在用户空间的嵌套页表错误的页面标识为应用程序所有.隐私保护器需要判断发生错误的物理页面是否被用来存储应用程序页表.而这种判断对于早期 AMD 处理器来说并未提供相应的标识信息,需要通过软件逐级遍历操作系统维护的页表,这种页表遍历对于内存敏感型应用程序来说是很耗时的.而对于现在普遍应用的 AMD 处理器来说,在发生嵌套页表错误时,相应的错误上下文信息均会被记录,隐私保护器通过记录的信息(AMD 处理器 VMCB 结构中的 exitinfo1[33:32])来确定发生错误的页面是否为页表页面.

3 实验与分析

3.1 实验配置

本文所提出的应用程序内存数据隐私保护方法已在曙光 A620r-G 服务器上进行了原型实现.该服务器配置两个 8 核 AMD 皓龙 6320 处理器,16GB 内存.服务器使用 Debian wheezy 操作系统,Linux 3.10 操作系统内核.本文用原始 Linux 操作系统作为测试基线,所有测试结果均与原始 Linux 操作系统的性能测试结果进行对比.

3.2 安全性分析

3.2.1 ret2user 攻击预防

本节通过一个具体的攻击实例说明本文所提保护机制如何预防 ret2user 攻击.虽然整个分析基于特定的攻击实例,然而实际 ret2user 攻击多采用与之相似的方法实现,因此该节的攻击预防分析具有一定的普适性.

图 5 是前文第 1.1 节所述 ret2user 攻击的具体实现流程.攻击者通过结合“*clone*()”函数与操作系统内核的一个堆栈溢出漏洞,从而实现对内核函数指针“*econet_ioctl*()”地址的修改,如图 5 中虚线①所示.当操作系统内核函数指针值被篡改后(图 5 中虚线②所示),攻击者调用 ioctl 系统调用,从而使操作系统引用被修改后的函数指针(图中虚线③所示),进而将操作系统控制流转向预先设定好的攻击代码(图 5 中虚线④所示).攻击代码以内核权限进行运行,从而实现权限提升或是用户隐私数据窃取攻击.

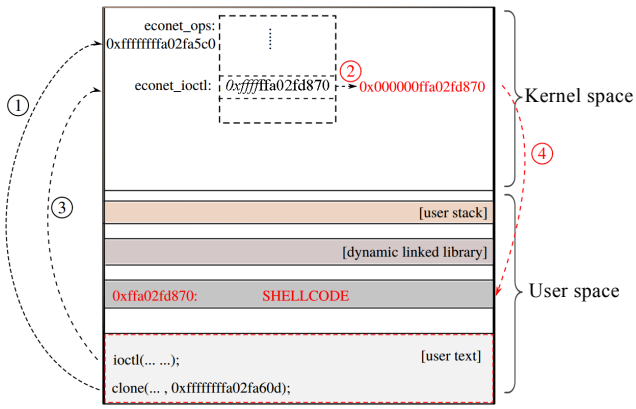


Fig.5 Exploitation of Econet vulnerability

图 5 Econet 漏洞攻击实例

在这类攻击方式中,必须存在一段能够以内核权限执行的用户态代码.根据前文所述,本文所提到的保护机制建立了较强的用户空间与内核空间隔离机制,因此当以内核权限执行用户态代码时,就会产生 NPT 错误中断.在 NPT 错误中断处理函数中,处理器特权级被强制性地降低为用户态特权级.因此,即使操作系统控制流被攻击者转向用户态的攻击代码,这些攻击代码也无法以内核态权限进行运行.由此可以看出,本文所提保护机制可以有效地防护 ret2user 类型攻击.

3.2.2 数据泄露预防

攻击者对其他应用程序内存隐私数据的窃取可以分为两种方式:第 1 种是通过获得内核权限,从而直接访问其他应用程序内存数据;第 2 种是通过利用操作系统内核漏洞将其他应用程序的物理页面映射到攻击进程的地址空间中,从而以普通权限窃取其他应用程序敏感数据.

本文为被保护应用程序和操作系统以及其他应用程序提供不同的 NPT 页表映射.在内核使用的 NPT 页表中,被保护应用程序的内存页面均被标识为不存在,因此当以内核权限访问被保护应用程序内存数据时就会产生 NPT 错误.针对第 2 种类型攻击,由于攻击者进程与被保护进程使用不同的 NPT 页表映射,因此即使被攻击的操作系统将被保护应用程序内存页面映射到攻击者进程地址空间,攻击者进程在访问被保护应用程序内存数据时依然会产生 NPT 页表错误.因此,本文提出的隐私保护机制可以有效地防止被保护进程的内存敏感数据被操作系统以及其他应用程序窃取.

3.3 实验结果与分析

图 6 为本文所述隐私保护方法在 SPEC INT 2006 上所带来的性能负载测试结果.图中所有测试结果均转换为相对于原生 Linux 操作系统性能的百分比,Y 轴表示所带来的性能负载,1 表示原生 Linux 性能,柱状条越高,说明带来的性能负载越大.

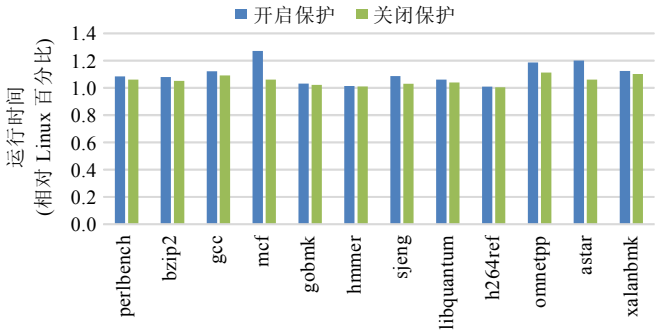


Fig.6 SPEC INT 2006 performance comparison among Linux, protection on and off

图 6 Linux, 保护机制开启及关闭情况下 SPEC INT2006 性能测试对比

从图中可以看出,对于 h264ref,hmmer,gobmk 等计算密集型应用,隐私保护所带来的性能开销只有 2%~4%之间,基本可以忽略不计;对于 perlbench,bzip2,libquantum 等多数应用,隐私保护所带来的性能开销大概在 6%~10%之间;而对于 mcf 这类内存敏感性应用,内存隐私保护带来了较大的性能负载,大约 27%左右.

在计算密集型应用运行过程中,处理器基本处于应用程序用户空间,用户空间与操作系统内核之间交互较少.因此在其运行过程中,隐私保护器很少介入应用程序运行,避免了因为从上层操作系统陷入虚拟机监控器所带来的性能开销.此外,该类应用具有很好的内存局部性,因此在应用程序运行过程中很少产生 TLB 刷新操作,从而减少了底层嵌套页表所带来的多级页表遍历开销.相反地,对于 mcf 这类内存敏感型应用,在其运行过程中内存分别局部性不好,带来较大的 TLB 刷新开销.原生操作系统采用 4 级页表,1 次 TLB 缺失操作最多需要访问 4 次内存页面访问获得相应的页表映射;而在虚拟化环境下,由于加入第 2 层地址转换机制,1 次 TLB 缺失需要最多进行 4×4 次内存页面访问才能完成相应的页面映射.所以相对于其他应用程序,mcf这类应用程序在隐私保护开启时带来了较大的性能开销.此外,从图中还可以看出,对于非保护应用程序,本文提出的系统所带来的性

能开销大约在 2%~6%之间.这说明其他非隐私敏感应用在该系统上运行时,其性能损失基本可以忽略不计.

表 1 为 4 个微型测试,分别为空系统调用、进程创建、内存复制以及缺页异常处理.在空系统调用中,测试结果单位为 CPU 周期数;进程创建测试结果单位为μs;内存复制为从用户空间向内核空间复制 1GB 数据所用的时间,单位为 s;缺页异常处理测试结果单位为μs.从表中可以看出,当隐私保护功能关闭时,相对于原生 Linux 操作系统,隐私保护系统所带来的性能开销可以忽略不计.当隐私保护功能打开时,对于空系统调用、进程创建以及缺页异常处理这 3 个测试,隐私保护系统带来大概 2~5 倍的性能开销,这主要是由每次从用户空间进入内核空间以及从内核空间返回用户空间均需要陷入虚拟机监控器所带来的.而对于 Fork 系统调用,隐私保护系统带来了极大的性能开销.这主要是由于每次创建新进程时,隐私保护系统都需要对被创建进程的地址空间进行遍历,查找进程所使用的物理页面,因而带来了较大的性能开销.

Table 1 Execution times of some micro experiments

表 1 微测试集执行时间

	Null system call	Fork	Memory copy	Page fault
Native Linux	133	76	0.92	0.136 4
Protection on	368	76	0.92	0.153 2
Protection off	135	2 508	2.08	0.512 8

4 相关工作对比

针对应用程序内存隐私数据的保护,国内外学者主要从两个不同方面进行研究:一种是通过加强传统操作系统安全性,当操作系统内核漏洞无法被攻击者利用后,本文所提到的攻击手段也会无法实施;另一种是通过虚拟化技术对操作系统监控,从而保证应用程序的内存隐私数据不被窃取.

4.1 加强操作系统安全性

为了提供高可靠性及高安全性,seL4 通过建立形式化模型对操作系统内核进行验证,保证操作系统编程接口的安全性[24].也有学者通过保证操作系统控制流以及数据流完整性,从而加强操作系统的安全性[5,6,25].虽然这些方法从一定程度上保证了操作系统的安全性,然而这些方法所带来的较高的性能负载使得其难以实际应用.形式化验证的方法虽然保证了高可靠性,然而其需要付出极大的人力、物力代价进行实现;此外,形式化验证的方法对系统代码量极其敏感,现有操作系统庞大的代码量使得形式化方法难以实际运用.

此外,还有一些学者通过防止以内核权限执行用户空间代码来保护操作系统内核.Secvisor[26]以及 Nickle[27]使用小型虚拟机监控器保证每次从操作系统内核空间切换到用户空间时,处理器的特权级强制地从 0 级设置为 3 级,从而防止以内核权限执行用户态代码.Intel 前期提供了一种名为 SMEP[21]的新处理器特性,SMEP 保证当处理器特权级处于 0 级时,用户态的内存页面是不可执行的.虽然这些方法能够有效地防护 ret2user 类型攻击,然而新的面向返回类型攻击使得攻击者依然可以通过操作系统来窃取用户隐私数据.

KGuard[4]通过编译器插件对操作系统内核自动加入相关检查点,在系统运行时,相应的检查点代码监测是否存在 ret2user 类型攻击.“return-less”[22]以及 G-Free[28]类型的防护机制通过剔除程序中所有的 ret 类型指令,从而防止 ret2user 类型攻击的发生.上述方法要么需要对操作系统内核进行重新编译,失去保护透明性;要么无法建立强健的用户空间与内核空间隔离机制,使得攻击者依然可以通过内存映射等方式进行 ret2user 类型攻击.此外,JOP(jump-oriented-programming)也使这些保护方法显得不够完善.与之相比,本文所提出的保护方法不需要对操作系统内核进行修改,同时建立了强的内核与用户空间隔离机制,能够有效地防止 ret2user 类型攻击发生.此外,较强的内核用户空间隔离机制也保证了应用程序内存数据的安全性.

4.2 基于虚拟机的隐私保护

XOM[29],AEGIS[16]以及 Cryptopage[15]通过设计新的处理器体系结构,保证在系统内存等硬件设备不可靠的环境下,应用程序内存隐私数据依然不会被攻击者窃取.与之不同的是,本文提出的隐私保护方法主要面对的是软件层面操作系统的不可靠性.因此,这些方法与本文所提出的方法相互正交,可以通过借鉴这些隐私保护方法

从而进一步完善本文所提出的隐私保护手段.

TrustVisor[11]通过将应用程序敏感的自包含片段(pieces of application logic,简称 PALs)与操作系统以及 DMA 设备完全隔离执行,从而保证应用程序隐私敏感部分的代码与数据不被攻击者篡改或窃取.与之类似,Intel 最新提出了一套新的指令集 SGX[14],该指令集通过为应用程序提供一个与操作系统以及其他应用程序完全隔离的独立区域(enclave),从而保证该区域代码与数据的安全可靠性.目前,该指令集仅在 Intel 的原型处理器上进行了实现,并未有相应的商业产品发布.SICE[30]利用处理器系统管理模式(system management mode)的天然强隔离性,将应用程序的隐私敏感代码及数据在 SMM 模式中进行执行与处理.Terra[31]使用虚拟化技术,通过限制被保护应用程序部分功能后,将其放在一个黑盒操作系统中执行,从而提供安全可靠的隔离环境.Fides[10]使用经过定制后的编译器对被保护应用程序进行编译,被保护应用程序代码模块被编译后,将被划分成公共部分以及隐私部分,在应用程序运行阶段,所有对应用程序内部的访问只能通过公共部分进行.该方法与传统面向对象语言定义的共有变量以及私有变量机制相似.上述隐私隔离保护方法从一定程度上保证了用户隐私数据的安全可靠,然而其需要对被保护应用程序架构进行重新设计(TrustVisor,Intel SGX,SICE),或是限制应用程序功能(SICE,Terra),或是对应用程序进行重编译(TrustVisor,Intel SGX,SICE,Fides),使得这些方法在实际应用中有很大难度.与之不同的是,本文所提出的隐私保护机制对操作系统以及应用程序透明,无需对被保护应用程序进行修改、重编译的特殊操作.

Chaos[32],Overshadow[33]以及 SP3[12]与本文类似,通过使用虚拟化技术,为被保护应用程序以及操作系统提供不同的物理地址映射,从而达到保护应用程序内存隐私数据的目的.这些方法基于 Xen,VMware 等商业化的虚拟机监控器,增加了系统 TCB,为系统带来了较大的攻击面.本文所提出的隐私保护方法使用轻量级安全虚拟机监控器 OSV,增加了系统可靠性,同时,与上述方法相比带来了一定的性能提升.另一方面,上述方法需要对操作系统或被保护应用程序进行静态编译,使得其无法应用到一些闭源软件上.本文提出的隐私保护方法对操作系统以及应用程序透明,通过使用安全加载器保护了动态链接库的完整性,从而无需对应用程序进行重新编译.

5 总结与未来的研究

本文以轻量级安全虚拟机监控器 OSV 为基础,提出一种对操作系统及应用程序透明的隐私数据保护方法.该方法在操作系统不可信环境下,能够在应用程序生命周期中保证其内存隐私数据无法被窃取.该方法主要提供 3 个方面的保护:(1) 保证操作系统只能访问应用程序显式或隐式提供的数据,比如系统调用参数等;(2) 保证攻击者注入的恶意代码无法以内核权限执行,同时保证被攻击的操作系统无法向被保护程序注入恶意代码;(3) 保护运行时动态链接库代码的完整性,避免利用动态链接库来窃取用户隐私.通过对原型系统进行性能测试,该保护方法对于大多数被保护应用程序仅带来 6%~10%的性能负载.

目前,该隐私保护方法还存在一些不完善性.比如,该方法未对 DMA 设备的内存操作进行监控,无法防止恶意 DMA 设备获取应用内存隐私数据.此外,在实际应用中,用户与计算机之间的输入、输出交互通道也是隐私数据泄露的一个重要渠道,文中所提出的方法并未涉及这些问题,而这些问题将作为本文未来的工作进行进一步研究.

References:

[1] CVE-2008-0600. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0600

[2] CVE-2010-4258. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4258

[3] On exploiting null ptr derefs, disabling SELinux, and silently fixed Linux vulns. http://seclists.org/dailydave/2007/q1/224

[4] Kemerlis VP, Portokalidis G, Keromytis AD. kGuard: Lightweight kernel protection against return-to-user attacks. In: Proc. of the 21st USENIX Conf. on Security Symp. 2012.

[5] Jr Petroni NL, Hicks M. Automated detection of persistent kernel control-flow attacks. In: Proc. of the 14th ACM Conf. on Computer and Communications Security. ACM Press, 2007. 103−115. [doi: 10.1145/1315245.1315260]

[6] Rhee J, Riley R, Xu D, Jiang X. Defeating dynamic data kernel rootkit attacks via VMM-based guest-transparent monitoring. In: Proc. of the Int'l Conf. on Availability, Reliability and Security (ARES 2009). IEEE, 2009. 74−81. [doi: 10.1109/ARES.2009.116]

[7] Jin H, *et al*. Computer System Virtualization Theory and Application. Beijing: Tsinghua University Press, 2008. 1−26 (in Chinese).

[8] Feng DG, Zhang M, Zhang Y, Xu Z. Study on cloud computing security. Ruan Jian Xue Bao/Journal of Software, 2011,22(1): 71−83 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3958.htm [doi: 10.3724/SP.J.1001.2011.03958]

[9] Hofmann OS, Kim S, Dunn AM, Lee MZ, Witchel E. InkTag: Secure applications on an untrusted operating system. In: Proc. of the 18th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM Press, 2013. 265−278. [doi: 10.1145/2451116.2451146]

[10] Strackx R, Piessens F. Fides: Selectively hardening software application components against kernel-level or process-level malware. In: Proc. of the 19th ACM Conf. on Computer and Communications Security (CCS 2012). ACM Press, 2012. 2−13. [doi: 10.1145/2382196.2382200]

[11] McCune JM, Li Y, Qu N, Zhou Z, Datta A, Gligor V, Perrig A. TrustVisor: Efficient TCB reduction and attestation. In: Proc. of the IEEE Symp. on Security and Privacy (SP). IEEE, 2010. 143−158. [doi: 10.1109/SP.2010.17]

[12] Yang J, Shin K. Using hypervisor to provide data secrecy for user applications on a per-page basis. In: Proc. of the 4th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. ACM Press, 2008. 71−80. [doi: 10.1145/1346256.1346267]

[13] Xiang GF, Jin H, Zou DQ, Chen XG. Virtualization-Based security monitoring. Ruan Jian Xue Bao/Journal of Software, 2012,23(8): 2173−2187 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4219.htm [doi: 10.3724/SP.J.1001.2012.04219]

[14] McKeen F, Alexandrovich I, Berenzon A, Rozas CV, Shafi H, Shanbhogue V, Savagaonkar UR. Innovative instructions and software model for isolated execution. In: Proc. of the 2nd Int'l Workshop on Hardware and Architectural Support for Security and Privacy. ACM Press, 2013. 10−19. [doi: 10.1145/2487726.2488368]

[15] Duc G, Keryell R. Cryptopage: An efficient secure architecture with memory encryption, integrity and information leakage protection. In: Proc. of the 22nd Annual Computer Security Applications Conf. (ACSAC 2006). IEEE, 2006. 483−492. [doi: 10.1109/ACSAC.2006.21]

[16] Suh GE, Clarke D, Gassend B, Van Dijk M, Devadas S. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In: Proc. of the 17th Annual Int'l Conf. on Supercomputing. ACM Press, 2003. 160−171. [doi: 10.1145/782814.782838]

[17] George V, Piazza T, Jiang H. Technology Insight: Intel Next Generation Microarchitecture Codename Ivy Bridge. 2011.

[18] Prandini M, Ramilli M. Return-Oriented programming. Security & Privacy, IEEE, 2012,10(6):84−87. [doi: 10.1109/MSP.2012.152]

[19] Dai YH, Qi Y, Ren JB, Shi Y, Wang XG, Yu X. A lightweight VMM on many core for high performance computing. In: Proc. of the 9th ACM SIGPLAN/SIGOPS Int'l Conf. on Virtual Execution Environments. ACM Press, 2013. 111−120. [doi: 10.1145/2451512.2451535]

[20] The GNU C library (glibc). 2014. http://www.gnu.org/software/libc/libc.html

[21] Dai YH, Shi Y, Qi Y, Ren JB, Wang PJ. Design and verification of a lightweight reliable virtual machine monitor for a many- core architecture. Frontiers of Computer Science, 2013,7(1):34−43. [doi: 10.1007/s11704-012-2084-0]

[22] Li JK, Wang Z, Jiang XX, Grace M, Bahram S. Defeating return-oriented rootkits with "return-less" kernels. In: Proc. of the 5th European Conf. on Computer Systems (EuroSys). ACM Press, 2010. 195−208. [doi: 10.1145/1755913.1755934]

[23] Trusted Computing Group. Trusted platform module (TPM) summary. 2014. http://www.trustedcomputinggroup.org/resources/trusted_platform_module_tpm_summary

[24] Klein G, Elphinstone K, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engelhardt K, Kolanski R, Norrish M, Sewell T, Tuch H, Winwood S. seL4: Formal verification of an OS kernel. In: Proc. of the ACM SIGOPS 22nd Symp. on Operating Systems Principles. ACM Press, 2009. 207−220. [doi: 10.1145/1629575.1629596]

[25] Wang Z, Jiang XX, Cui WD, Ning P. Countering kernel rootkits with lightweight hook protection. In: Proc. of the 16th ACM Conf. on Computer and Communications Security. ACM Press, 2009. 545−554. [doi: 10.1145/1653662.1653728]

[26] Seshadri A, Luk M, Qu N, Perrig A. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity Oses. ACM SIGOPS Operating Systems Review, 2007,41(6):335−350. [doi: 10.1145/1323293.1294294]

[27] Riley R, Jiang X, Xu D. Guest-Transparent prevention of kernel rootkits with VMM-based memory shadowing. In: Proc. of the Recent Advances in Intrusion Detection. Springer-Verlag, 2008. 1−20. [doi: 10.1007/978-3-540-87403-4_1]

[28] Onarlioglu K, Bilge L, Lanzi A, Balzarotti D, Kirda E. G-Free: Defeating return-oriented programming through gadget-less binaries. In: Proc. of the 26th Annual Computer Security Applications Conf. (ACSAC). IEEE, 2010. 49−58. [doi: 10.1145/1920261.1920269]

[29] Lie D, Thekkath C, Mitchell M, Lincoln P, Boneh D, Mitchell J, Horowitz M. Architectural support for copy and tamper resistant software. ACM SIGPLAN Notices, 2000,35(11):168−177. [doi: 10.1145/356989.357005]

[30] Azab A, Ning P, Zhang XL. SICE: A hardware-level strongly isolated computing environment for x86 multi-core platforms. In: Proc. of the 18th ACM Conf. on Computer and Communications Security. ACM Press, 2011. 375−388. [doi: 10.1145/2046707.2046752]

[31] Garfinkel T, Pfaff 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]

[32] Chen HB, Zhang FZ, Chen C, Yang ZY, Chen R, Zang B, Yew PC, Mao WB. Tamper-Resistant execution in an untrusted operating system using a virtual machine monitor. 2007. http://ppi.fudan.edu.cn/system/publications/paper/chaos-ppi-tr.pdf

[33] Chen XX, Garfinkel T, Lewis EC, Subrahmanyam P, Waldspurger CA, Boneh D, Dwoskin J, Ports DRK. Overshadow: A virtualizationbased approach to retrofitting protection in commodity operating systems. ACM SIGPLAN Notices, 2008,43(3):2−13. [doi: 10.1145/1353536.1346284]

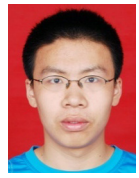
附中文参考文献:

[7] 金海,等.计算系统虚拟化——原理与应用.北京:清华大学出版社,2008.1−26.

[8] 冯登国,张敏,张妍,徐震.云计算安全研究.软件学报,2011,22(1):71−83. http://www.jos.org.cn/1000-9825/3958.htm [doi: 10.3724/SP.J.1001.2011.03958]

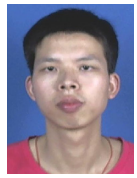
[13] 项国富,金海,邹德清,陈学广.基于虚拟化的安全监控.软件学报,2012,23(8):2173−2187. http://www.jos.org.cn/1000-9825/4219.htm[doi: 10.3724/ SP.J.1001.2012.04219]

任建宝(1986－),男,陕西西安人,博士生,主要研究领域为操作系统,虚拟化技术,系统安全.

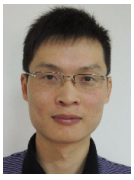


王晓光(1987－),男,博士生,主要研究领域为操作系统,虚拟化技术,系统安全.

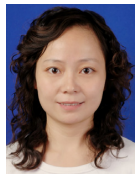
齐勇(1957－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为操作系统,分布式系统,云计算虚拟化技术,系统安全与应用.



宣宇(1988－),男,硕士,主要研究领域为操作系统,虚拟化技术.



戴月华(1981－),男,博士,讲师,主要研究领域为操作系统,虚拟化技术.



史楠(1975－),女,博士,讲师,CCF 会员,主要研究领域为操作系统,虚拟化技术,系统安全.