软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(9):2443−2458 [doi: 10.13328/j.cnki.jos.004875]

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

基于内存取证的内核完整性度量方法*

陈志锋[1,2], 李清宝[1,2], 张 平[1,2], 王 炜[1,2]

[1](解放军信息工程大学,河南 郑州 450001)

[2](数学工程与先进计算国家重点实验室,河南 郑州 450001)

通讯作者: 陈志锋, E-mail: xiaohouzi06@163.com

摘 要: 内核级攻击对操作系统的完整性和安全性造成严重威胁.当前,内核完整性度量方法在度量对象选取上存在片面性,且大部分方法采用周期性度量,无法避免 TOC-TOU 攻击.此外,基于硬件的内核完整性度量方法因添加额外的硬件使得系统成本较高;基于 Hypervisor 的内核完整性度量方法,应用复杂的 VMM 带来的系统性能损失较大.针对现有方法存在的不足,提出了基于内存取证的内核完整性度量方法 KIMBMF.该方法采用内存取证分析技术提取静态和动态度量对象,提出时间随机化算法弱化 TOC-TOU 攻击,并采用 Hash 运算和加密运算相结合的算法提高度量过程的安全性.在此基础上,设计实现了基于内存取证的内核完整性度量原型系统,并通过实验评测了 KIMBMF 的有效性和性能.实验结果表明:KIMBMF 能够有效度量内核的完整性,及时发现对内核完整性的攻击和破坏,且度量的性能开销小.

关键词: 内核完整性;完整性度量;TOC-TOU;内存取证;时间随机化

中图法分类号: TP316

中文引用格式: 陈志锋,李清宝,张平,王炜.基于内存取证的内核完整性度量方法.软件学报,2016,27(9):2443−2458. http://www.jos.org.cn/1000-9825/4875.htm

英文引用格式: Chen ZF, Li QB, Zhang P, Wang W. Kernel integrity measurement method based on memory forensic. Ruan Jian Xue Bao/Journal of Software, 2016,27(9):2443−2458 (in Chinese). http://www.jos.org.cn/1000-9825/4875.htm

Kernel Integrity Measurement Method Based on Memory Forensic

CHEN Zhi-Feng[1,2], LI Qing-Bao[1,2], ZHANG Ping[1,2], WANG Wei[1,2]

[1](PLA Information Engineering University, Zhengzhou 450001, China)

[2](State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

Abstract: Kernel-level attacks are serious threat to the integrity and security of operating systems. Existing kernel integrity measurement methods are one-sided when selecting the measurement objects, as most of these methods suffer from periodic detection shortcoming that makes themselves vulnerable to TOC-TOU attacks. Besides, hardware-based kernel integrity measurement methods are usually too expensive, while hypervisor-based kernel integrity measurement methods are always likely to degrade system performance due to the introduction of complex VMMs. To address these problems, this study proposes a kernel integrity measurement approach based on memory forensics technique (KIMBMF). First, the static and dynamic measurement objects are extracted with the memory forensics technique, and a time random algorithm is presented to degrade the impact caused by TOC-TOU attacks. At the same time, a novel algorithm is also introduced by combining the Hash operation with cryptographic operation, thereby ensuring the security of the measurement progress. Next, a kernel integrity measurement prototype is implemented according to the above techniques and algorithms, and its effectiveness and overhead are evaluated. Experimental results show that KIMBMF can measure the integrity of operating system effectively, and has a reasonable time overhead.

* 基金项目: “核高基”国家科技重大专项(2013JH00103); 国家高技术研究发展计划(863)(2009AA01Z434)

Foundation item: National Science and Technology Major Project of China (2013JH00103); National High-Tech R&D Program of China (863) (2009AA01Z434)

收稿时间: 2014-11-04; 修改时间: 2015-05-05; 采用时间: 2015-07-13

Key words: kernel integrity; integrity measurement; TOC-TOU, memory forensic; time randomization

操作系统的安全性是计算机安全不可或缺的一部分,内核是操作系统的重要组成,其安全性直接影响着操作系统的安全性.完整性度量技术通过测量、验证、评估系统的预期与实际的符合情况,提供系统安全状况的全面描述.利用完整性度量技术度量内核的完整性,对内核在内存中的运行状态、资源的完整性进行评估,可以有效保护和增强内核和操作系统的安全.

现有的内核完整性度量方法根据部署层次的不同可分为基于主机的内核完整性度量方法、基于硬件的内核完整性度量方法和基于 Hypervisor 的内核完整性度量方法.其中,基于主机的完整性度量方法[1−3]仍然是目前使用最广泛的检测方法.文献[1,2]主要对内核的静态对象如系统调用表等进行度量,未考虑动态对象,不能够发现内存数据攻击,且采用周期性分析技术,存在 TOC-TOU(time-of-check to time-of-use)问题[4];文献[3]对进程和内核模块的代码进行度量,虽然其强调在任意时刻都能进行度量操作,但是未讨论何时度量才能够有效发现攻击,且其度量对象具有局限性.基于 Hypervisor 的内核完整性度量方法[5−8]将度量工具部署在 Hypervisor 中,一定程度上提高了度量工具的安全性,但由于引入虚拟监控对系统的干预,带来了额外的负载和性能损失,并且由于缺乏充分的上层语义信息,仅关注进程相关的度量对象,未考虑其他对象,如模块、网络连接等,导致度量结果不精确.SBCFI[5]通过验证代码和内核函数指针的完整性来保护内核完整性.Osck[6]借鉴 SBCFI 的思想,通过监控静态的、持续的和动态的控制流转移以及验证非控制流数据的属性,实现内核完整性度量.LKIM[7]则是利用上下文检查技术度量内核代码段、系统调用表、中断描述表、全局描述符表等静态内容和动态数据结构的完整性.基于辅助硬件的检测方法[9−12]将检测程序固化于硬件,与被检测对象隔离开,具有防篡改性和不可旁路性.但由于需要额外的硬件支持且不可避免地引入性能开销,并未得到广泛应用.同时,该方法也存在 TOC-TOU 问题. Copilot[9]通过 PCI 接口连接的协处理器,以 DMA 方式周期性地度量内核代码的完整性,由于其仅支持对具有持久状态的内存状态进行快照,度量对象受限,且采用周期性度量,无法防御 TOC-TOU 攻击.Gibraltar[10]通过 PCI 接口连接的后端安全网络,远程访问目标客户系统的内存,推导内核数据不变量,判断内核的完整性.Vigilare[11] 和 MGUARD[12]分别利用总线嗅探技术和高级内存缓存扩展技术(advanced memory buffer,简称 AMB)实现内核完整性监控,两者都是通过捕获内核存储空间访问的瞬态操作实现瞬态攻击的检测,Vigilare 由于总线带宽受限和商业处理器存储控制器的隐藏性和不可访问性导致其不实用,而 MGUARD 集成到 DRAM DIMM 现成设备中,且对存储控制器和 CPU 是透明的,不存在 Vigilare 面临的问题,但实现难度较大.

针对这些问题,提出了基于内存取证的内核完整性度量方法(kernel integrity measurement based on memory forensic,简称 KIMBMF).该方法属于第 1 类方法,但可以部署在不同层.它基于内存取证技术提取内核空间度量对象,在度量时间点触发内存提取,提取指定范围的内存内容;然后,根据度量需求从内存内容中重构出度量对象,过程中不需复杂硬件以及 Hypervisor 的支持,不增加系统成本,减少了系统性能损失,并且适用于通用平台.采用随机化时间触发度量,提取、分析和度量等过程不再遵循周期性规律,避免攻击者掌握度量规律,弱化 TOC-TOU 攻击.综合考虑静态度量对象和动态度量对象,有效提高了度量的准确性.实验结果表明:KIMBMF 能够有效度量内核的完整性,且比现有的周期性完整性度量工具度量准确率更高,对系统的负担和时间开销更小,负担仅为 0.27%,度量时间保持在 ms 级.

本文第 1 节在分析内存取证技术的基础上讨论基于内存取证的完整性度量方法面临的难题和需要突破的关键技术.第 2 节详细介绍基于内存取证的完整性度量方法及其实现.第 3 节介绍本文实现的基于内存取证的内核完整性度量原型系统.第 4 节给出实验结果并对本文的方法进行性能分析.第 5 节给出结论及下一步工作.

1 问题描述

完整性度量的根本问题是准确、及时、全面地获取度量对象,现有的内核完整性度量方法在获取度量对象上存在片面性,且借助硬件或者虚拟监控带来了额外的代价和开销.解决这些问题,是提高完整性度量有效性的关键.

内存取证技术是当前流行的数字取证分析技术,着眼于从计算机内存中获得信息,如内存中存在的进程信息、文件的使用情况或网络连接状态等,可以提供系统遭受入侵发生时和发生后系统运行状态的大量信息,引起了研究者的广泛关注,相关工具不断呈现,如基于 Windows 的 DOSKEY[13],CMAT[14]等、基于 Linux 系统的 Foriana[15],SecondLook[16]等.内存取证主要包括两个关键步骤:内存提取和内存分析.内存提取提供物理内存内容,内存分析通过分析物理内存内容获取重要的入侵证据等资源.

完整性度量对象存在于内存中,因此,内存取证技术的基本思想可用于度量对象的获取,且该技术获取度量对象不需要硬件辅助,不需要 Hypervisor 的支持,可克服现有完整性度量方法存在的代价、性能和语义问题.但是内存取证技术并没有涉及度量对象的相关讨论.此外,内存取证技术只是提供了一套机制,没有讨论何时进行度量克服周期度量缺陷(度量点设置问题)、如何进行度量等问题的解决方法.

因此,将内存取证技术应用于内核完整性度量,还需要解决以下问题:第一,内存取证提供了提取度量对象的方法,但从内存中获取哪些度量对象、如何获取度量对象,是完整性度量面临的首要问题;第二,周期性度量存在 TOC-TOU 问题,攻击者可绕过完整性度量实现攻击,因此在不采用实时监控的情况下,如何合理分布度量点降低攻击者逃逸度量的可能性,是基于内存取证的完整性度量需要解决的另一个难点.

2 基于内存取证的内核完整性度量

2.1 内核内存取证

2.1.1 随机内存提取

目前,内存取证获取物理内存内容的方法主要包括基于硬件的方法和基于软件的方法[17].

基于硬件的方法所依赖的硬件有 Tribble 设备和 FireWire 设备.Tribble 设备获取内存时不需要在内存中引入额外的代码,对内存影响小,但该设备需要跟随机器一起提前装入.而 FireWire 设备已集成到主板中,不存在普适性问题,使用方便,但是早期实现的产品[17,18]容易导致系统死机或者提取的内存内容不完整.为此,Gladyshev 等人[19]开发的 goodfish、山东大学的王连海[20]提出的内存数据获取方法克服了该缺点.

基于软件的方法[14,17,21]不需要硬件辅助,快捷有效,但是性能上低于基于硬件的方法.DD 工具[14]因新版本内核限制/dev/mem,/dev/kmem 使得其获取内存能力受限.LiME[21]通过内存映射实现内存提取,支持内存提取到外部设备文件系统或者通过网络传递提取的内存.但是该工具目前仅支持全部物理内存内容的提取.虚拟环境下客户机的运行状态能够暂时被挂起,其虚拟的内存可以保存在相应的主机系统中[20].

基于硬件的方法能够有效获取物理内存,但需要硬件辅助,成本较高.本文选择在现有硬件的基础上,采用软件方式实现内存提取.由于现有基于软件的方法将整个物理内存内容复制到存储设备中,随着物理内存容量的增大,时间开销也大大增加.事实上,对于一个运行的操作系统,内核占用的物理内存空间有限,同时,内核内存分析时也是针对性地提取度量对象,将内存内容全部转存到外部存储设备中是不必要的.

因度量对象离散分布在内存中,要克服现有内存提取方法的不足、实现按需提取的解决方法就是要设计一种能够随机提取物理内存内容的方法.随机提取的实现,需要解决程序任意访问内存的限制问题,本文通过内核模块或驱动实现内核内存访问,通过地址映射实现对内核任意物理内存地址的访问,从而可提取任意地址的物理内存内容.与 LiME 不同的是,这里是按需映射需要访问的内核内存空间,并没有一次性映射所有内核内存空间.

按照上述解决思路,设计实现了可选择的物理内存内容提取算法 PMCE(physical memory contents extractor),其中,可选择是指可提取任意位置任意范围的物理内存内容.该算法实现物理内存的随机访问和内容提取,解决了物理地址和逻辑地址的相互映射问题,算法的第 4 步~第 7 步对应地址映射过程,将物理地址映射为逻辑地址后,程序便可获得访问物理内存的能力.

算法 1. PMCE 算法.

输入:地址范围[*address*1,*address*2];

输出:物理内存内容 *MC*.

```
1.  if (address1<0|address2>maxphysical)
2.     exit();
3.  end if
4.  page1= address1/PAGE_SIZE; offset1=address1%PAGE_SIZE;      //地址映射
5.  page2=address2/PAGE_SIZE; offset2=address2%PAGE_SIZE;
6.  void*from=kmap(page1)+offset1;
7.  void*to=kmap(page2)+offset2;
8.  copy_to_user(buf,from,to);                                  //资源拷贝实现内存访问
9.  MC=copy(buf);
10. return MC;
```

在算法 1 的基础上,内存提取方法的基本步骤如下:

1. 加载物理内存字符设备驱动程序(算法 1 的具体实现).该驱动程序实现了对物理内存区域的访问,可对物理内存内容的读取和存储位置的随机访问;
2. 构建物理内存设备文件.设备文件与驱动程序通过设备号统一实现访问,通过该设备文件,用户程序可自由读取物理内存内容;
3. 在步骤 1、步骤 2 的基础上,用户程序可对设备文件任何地址或地址范围的内容进行读取,即,实现对度量对象的获取.

PMCE 算法实现了物理内存的随机访问,能够获取物理内存任意地址范围的内容.由于完整性度量所需要的度量对象并不是连续分布和存储在内存中,基于硬件的内存提取方法和基于软件的内存提取方法都是将整块内存提取出来,再从中提取资源,一方面对系统的影响较大,另一方面带来较大的时间开销.而 PMCE 不需要读取整块内存内容,只需根据度量对象在内存中的位置提取资源,一方面实现了按需访问,一步提取,对系统的负担较小;另一方面符合度量对象分布规律,极大地加快度量对象的提取.

2.1.2 内核内存分析

通过 PMCE 访问物理内存提取的是二进制格式的内容,下一步工作是从这些内容中重构度量对象,而首先必须确定哪些对象和资源在系统运行过程中会影响内核的完整性.

1. 度量对象的确定

从操作系统内核在内存中的基本组成、运行机制和被攻击对象等几个方面综合考虑来确定度量对象.

通过分析操作系统内核加载到内存中的各种资源可知:内核的代码段、只读数据段、中断描述符表、系统调用表、全局描述符表等在操作系统运行过程中是保持不变的,即,资源的地址、结构定义和偏移等不会发生变化.一旦这部分内容被破坏,系统的预期行为就会发生变化,即,完整性遭受破坏.因此,完整性度量需要重点关注它们.由于它们在系统运行过程中保持不变,称它们为静态度量对象.此外,在系统运行过程中存在动态变化的资源,如进程、模块、网络连接、熵池资源等数据,称为动态度量对象.尽管它们在系统生命周期内不断变化,但是它们具备一定的特性.例如,Linux 系统中的进程满足 $run\_list \subseteq all\text{-}tasks$ 或者 $all\text{-}tasks == ps\text{-}tasks$,其中,*run_list* 表示调度运行的进程,*all-tasks* 表示系统所有的进程,*ps-tasks* 表示系统工具可看到的所有进程,那么进程资源在系统运行过程中是完整的,否则是不完整的;Linux 内核使用伪随机数生成器 PRNG 生成随机数,为了确保随机数的随机性,内核使用多项式更新熵池内容,多项式的系数在 *poolinfo* 数据结构的整形字段中声明,它们的取值满足 $poolinfo.tap1 \in \{32,128\}$,$poolinfo.tap2 \in \{26,103\}$,$poolinfo.tap3 \in \{20,76\}$,$poolinfo.tap4 \in \{14,51\}$,$poolinfo.tap5 \in \{7,25\}$,$poolinfo.tap6==1$ 等;其余的动态对象与进程类似具备其他的特性.根据动态对象的特征,它们可分为 5 类:子集关系类、从属关系类、约束关系类、固定长度关系类和固定取值关系类.如进程、模块、网络连接等属于子集关系类,熵池结构属于从属关系类,支持的可执行文件类型数 *formats* 属于固定长度类等.

此外,通过分析 rootkit 攻击行为发现:rootkit 对内核在内存中的各种资源进行篡改,从而破坏内核的完整性.例如,存在直接篡改中断处理函数和系统调用处理函数的 kbeast,enyelkm 等 rootkit[22].又如:adore-ng[22]通过系统

拦截实现进程隐藏、文件隐藏等破坏进程、文件的完整性;attack_tasklist[23]将要隐藏的进程从进程链表中移除;wipemod[22]将自身从模块链表中移除,实现恶意模块自隐藏等.此外,现有的内核完整性度量方法如 Copilot[9], Osck[6]等也是针对 rootkit 的攻击对象提出相对应的完整性度量策略和方法.

根据上述分析,本文的度量对象包括静态度量对象和动态度量对象两大类,它们分布在内存的不同位置,下面讨论如何从内存中分析重构出这些度量对象.

2. 度量对象的分析

操作系统内核符号表是内存分析的切入点.内核符号表中存储的是内核关键数据结构、关键数据的入口地址和存储地址.Linux 内核符号表为 *System.map*,该文件中存储了成千上万个符号地址,我们只关注与完整性度量相关的内容,主要包括内核代码段起始地址和终止地址、内核初始化数据段起始地址和终止地址、中断向量表存储地址、系统调用表存储地址、0 号进程入口地址等.此外,内核关键数据结构的定义及各个字段的偏移,也是内存分析的重要条件.

下面以 Linux 系统进程分析为例说明内存分析的基本思想.进程在内存中的信息存储在进程结构体 *task_struct* 中,所有的进程通过双链表链接,如图 1 所示.分析过程中,首先需要某一个进程的 *task_struct* 结构在内存中的位置,0 号进程是系统所有进程的父进程,其永驻内存,通过 *System.map* 获取其逻辑地址后转换为物理地址,即可提取该进程在内存中的内容,包括 PID、运行状态、进程名、虚拟内存等基本信息.之后,以该进程为出发点,通过进程结构体的链式组织结构从内存中分析其余进程.

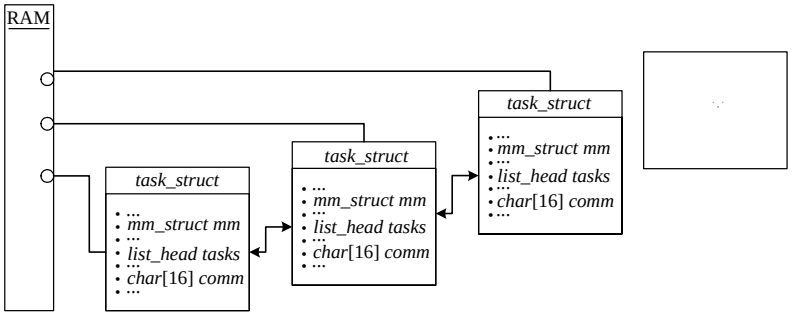


Fig.1 Process structure and its organization on memory

图 1 内存中进程结构及其组织

综上,在分析物理内存时,以内核符号表为基础,提取度量对象内存地址,根据地址关系和内核关键数据结构获取度量对象.根据该思想,设计实现了内存分析算法 PMA(physical memory analysis).算法的第 3 步~第 8 步实现静态度量对象的分析,第 9 步~第 13 步实现了动态度量对象的提取.

算法 2. PMA 算法.

输入:物理内存内容 *MC*、结构定义 *SC*[]、字段偏移 *FS*[]、*System.map*;

输出:静态度量对象或动态度量对象 *MSAD*[].

1. *address*[]=*getmeasuresoure*(*System.map*)
2. **while** (*address*[*i*]!=0)
3. **if** (*address*[*i*]∈*codesection*|*datasection*|*IDTEntry*|*System_call_entry*) //静态度量对象分析
4. *MSAD*[]=*getcode*(*MC*); //代码
5. *MSAD*[]=*getdata*(*MC*); //数据
6. *MSAD*[]=*getIDT*(*MC*); //中断向量表
7. *MSAD*[]=*getSys_call*(*MC*); //系统调用表
8. return *MSAD*[];

```
9.    else if (address[i]∈init_task|module_set|others)
10.     MSAD[]=analyzeTask(init_task,SC[],FS());   //进程
11.     MSAD[]=analyzeMoudle(module_set,SC[],FS());   //模块
12.     MSAD[]=analyzeOther(others,SC[],FS[]);   //文件、网络连接、不变量取值等其他内容
13.     return MSAD[];
14.   end if
15. end while
```

由于代码、数据、中断向量表和系统调用表等静态度量对象在内存中是连续分布的,故分析过程相对单一,本文不再细讲,重点论述动态度量对象的分析方法.下面以进程为例,分析函数 *analyzeTask*(*init_task*,*SC*[],*FS*())的具体实现过程:

1. 根据 *init_task* 的地址调用 PMCE 算法,获取 0 号进程的进程结构体内容,存入 *SC*[];
2. 根据 *task_struct* 结构体的字段定义和偏移信息 *FS*[](*pid*,*comm*,*tasks* 等)重构 *SC*[],获取进程的操作系统级语义,存入 *p*1[];
3. 利用进程结构体链接关系重复步骤 1 和步骤 2 获取所有进程信息,分别存入 *p*1[];
4. 采用与分析 *task_struct* 结构体一样的方法分析 *runqueues* 结构体(图 2),将结果存入 *p*2[];
5. 采用去重法合并 *p*1[]和 *p*2[]得到最终的进程关系存入 *MSAD*[].

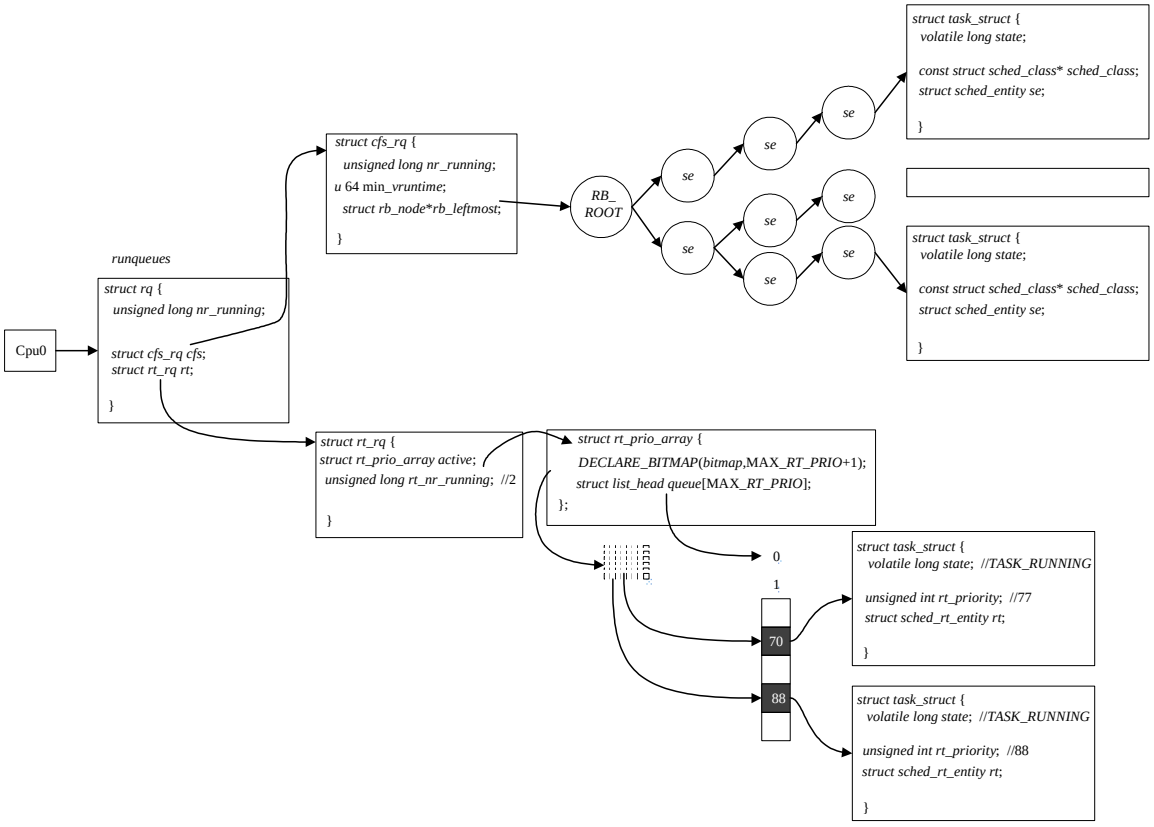


Fig.2 Related data structs of runqueues analysis

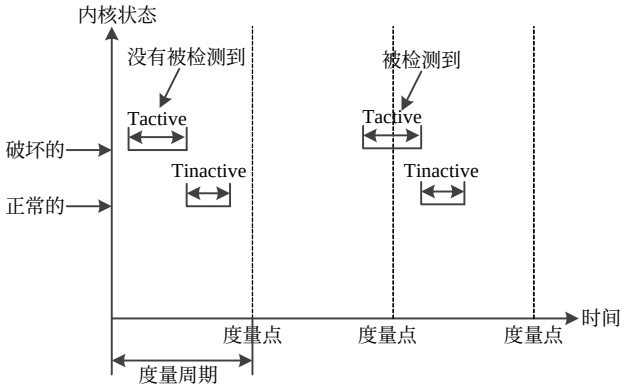
图 2 runqueues 分析相关数据结构关系图

图 2 中给出了 CPU 某个核上的就绪进程队列相关数据结构的关系.根据相关的数据结构定义和相关字段

的偏移关系,可以得到就绪进程的 *task_stuct* 结构,从而可以得到进程信息.分析过程与分析 *task_struct* 链表过程类似.其他动态度量对象分析方法类似,不再赘述.

2.2 内核内存取证

由于周期性度量存在 TOC-TOU 攻击漏洞,攻击者利用周期内的时间发动 rootkit 攻击即可躲避完整性度量,攻击如图 3 所示.



注:Tactive 表示攻击激活期,Tinactive 表示攻击者去除攻击踪迹以免被检测出

Fig.3 Vulnerability of periodic measurement

图 3 周期性度量漏洞

从图 3 中可以看出:度量时间点与度量时间点之间的时间是固定的,不妨设为 *T*.攻击者一旦发现这一规律,便可打破这种度量.因此,为了避免攻击者掌握完整性度量规律,改进了周期性时间算法,提出时间随机化算法 TR.该算法将度量操作随机地分布在时间域上,使得攻击者无法掌握度量时间点的分布情况.

然而,某些随机化时间算法可能导致度量点过于密集或者度量点过于稀疏,过于密集将带来系统负载增加,过于稀疏则会导致度量准确率下降,故度量点的分布不能任意随机化.

为了确定合理的时间范围,本文首先选取了若干时间范围,测试这些时间范围内的度量对系统的负载,其中,度量准确率是在假设攻击者发现了度量规律的情况下给出的评估,结果见表 1.

Table 1 Relation of measurement time range and system load

表 1 度量时间范围与系统负载关系

时间范围	系统负载	度量准确率(相对)
[*T*/3,*T*/2]	2.3	较高
[*T*/4,*T*]	1.8	较高
[*T*/4,6*T*/5]	1.3	较高
[*T*/3,6*T*/5]	1.2	较高
[*T*,6*T*/5]	0.9	一般
[*T*,2*T*]	0.6	一般
T	1.0	较低

根据表 3 所示不同时间范围的度量时间序列对系统负载的影响,同时考虑度量准确率,本文将随机种子限定在[*T*/3,6**T*/5]范围内,即:度量间隔时间在该范围内,既保证度量点的分布满足要求,又确保了度量时间是随机的.在时间序列产生之后,依次将它们累加到时间域,即可构成新的度量时间点分布,具体过程如下.

1. 在给定的时间范围[*T*/3,6**T*/5]内生成种子 *s_seed*;
2. 根据要生成的时间序列个数 *n*(不妨设为 65 535)选择一个小于 *n* 的最大素数 *multiplier* 和一个附加数 *adder*,计算式(((*multiplier***s_seed*+*adder*)>>16)%(6**T*/5))的值 *s_time*:若 *s_time* 小于 *T*/3,那么将 *s_time* 值加上 *T*/3 得到最终的一个时间点 *RS*;否则,*s_time* 作为最终的一个时间点;

3. 重复步骤 2,直至时间点个数为 *n*,结束时间序列生成;
4. 将该时间序列同系统时间同步,即:以目前系统时间为第 1 个开始度量点,将时间序列中的每一个时间点依次累加该系统时间,保证在每个时间点都会触发完整性度量.

3 基于内存取证的内核完整性度量原型系统

本文设计实现了基于内存取证的内核完整性度量原型系统 KIMBMF,其基本结构如图 4 所示.

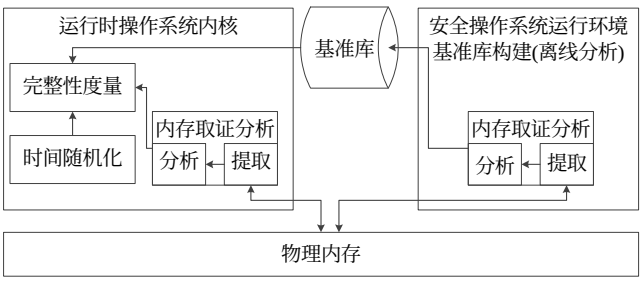


Fig.4 Integrity measurement property system based on memory forensic

图 4 基于内存取证的完整性度量原型系统

基准库是完整性度量的依据.基准库的内容主要为内核运行过程中在内存中保持不变的静态度量对象对应的 Hash 值和除子集关系类对象外的其他动态对象的值或者关系.基准库的构建是在安全环境下进行的,首先确保基准库构建时对内核没有额外的影响;其次采用离线构建,即,系统不联网,避免遭受网络攻击.同时,为了保证完整性度量时基准值的安全性,对度量内容进行 Hash 计算取得基准值后对其进行加密.基准库构建采用的 Hash 和加密算法,与度量过程中采用的算法是一致的.

内存取证分析包括内存提取和内存分析两部分,分别实现了度量对象的提取和分析,为完整性度量过程提供资源.

时间随机化模块主要实现度量时间的生成,将度量操作随机分布在时间域上,一方面产生动态度量的效果,另一方面弱化 TOC-TOU 攻击.

完整性度量模块按照完整性度量算法对内核进行完整性度量,将度量结果报告给用户并按照一定的策略做出响应操作.

基于内存取证的内核完整性度量的基本过程如图 5 所示.从图 5 可以看到:每当度量时间点到来,触发一次完整性度量.首先,内存取证分析提取度量对象;然后,对静态对象和动态对象进行完整性度量分析.静态对象的度量只需与基准库的基准值进行匹配分析,而动态对象的度量需要根据动态对象的类型进行区分.若动态对象为进程、模块等子集关系类对象,则需要同时构建两个视图,然后通过视图差异判断该对象的完整性;若为其余类型的动态对象,则只需分析内存中该对象的相关内容,再与基准库中存储的值或者关系进行对比分析即可.

动态完整性度量过程中的关键是确保基准值和度量对象的安全性,安全性取决于 Hash 散列函数和其他增强方法.文中 Hash 散列函数采用的是 SHA1 算法,但由于 Hash 算法存在安全性问题,不可信的系统可能生成一个与某个度量内容相同散列值的基准值.故,为了提高基准值的安全性,在度量对象进行 Hash 运算之后对 Hash 值进行加密处理,采用的加密算法为 AES.为了防止 AES 密钥泄露,提高安全性,密钥的生成和存储均在 TPM 芯片内完成.

整个度量过程不需要硬件辅助,不需要 Hypervisor 监控提供度量对象;度量时间点的设置采用时间随机化算法,可避免攻击者掌握度量规律;内存取证负责分析度量点时刻的内存状态并提取度量对象,为度量过程提供基础保证;完整性度量分别度量静态对象和动态对象的完整性,全面度量内核的完整性.

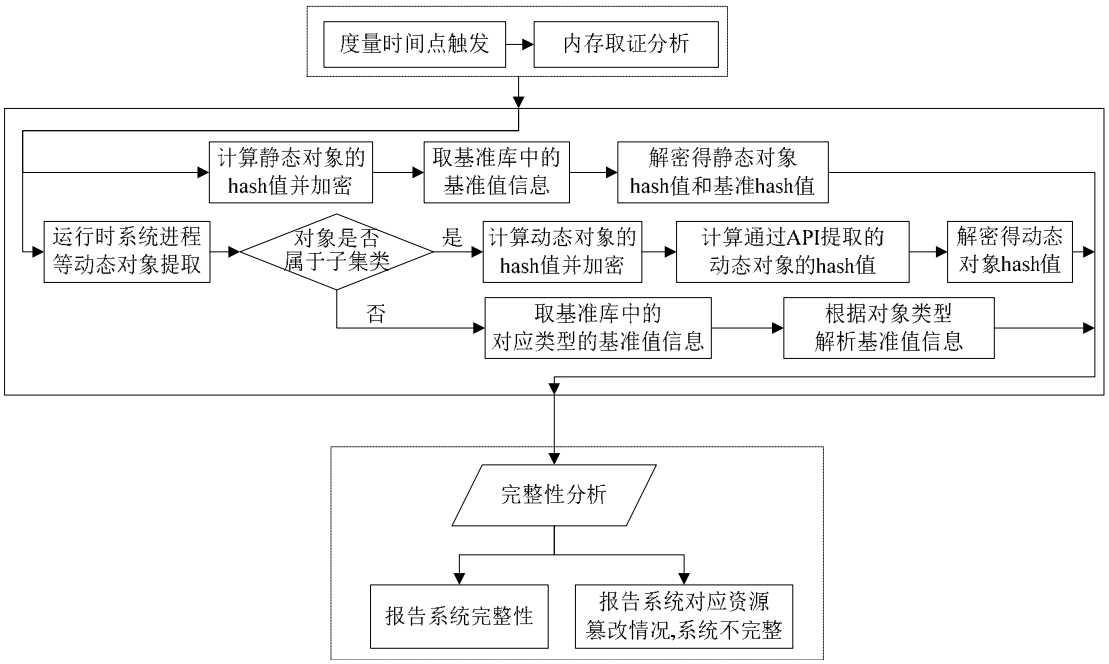


Fig.5 A process of kernel integrity measurement

图 5 一次内核完整性度量过程

4 实验及结果分析

本节从功能和性能两个方面来对 KIMBMF 进行评测,功能测试用于验证 KIMBMF 的有效性,即,能否度量发现系统的异常和攻击;性能测试用于验证 KIMBMF 的效率和开销.实验环境如下:

操作系统为 Ubuntu 10.04,内核版本 2.6.32-52-686,CPU 为 Intel(R) Core(TM) i3-3217U@1.80GHz,内存大小 4G,硬盘容量 500G.选择 kbeast,enyelkm,ddrk,adore-ng,mood-nt 和 suckit v2.0 等 rootkit[15]作为测试程序.

4.1 功能测试

本节首先选取了 kbeast,enyelkm 和 adore-ng 这 3 种 rootkit 来验证 KIMBMF 的有效性.

1. kbeast

Kbeast 通过修改内核系统调用表的若干项,实现了隐藏自身、隐藏文件目录、隐藏进程、Root 提权、隐藏后门等功能.在实验机器上安装该款 rootkit 之后,启动 KIMBMF 度量内核的完整性,度量结果如图 6 所示.

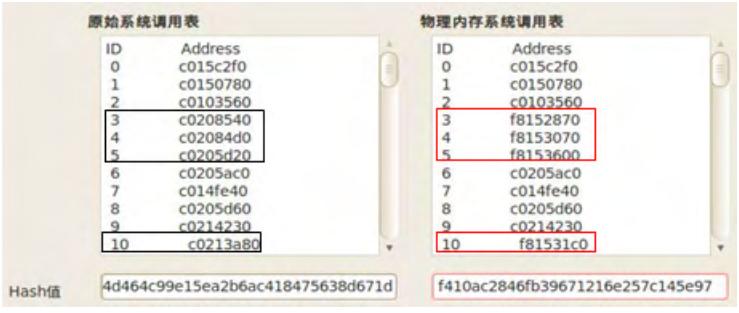


Fig.6 Integrity measurement of system call table

图 6 系统调用表完整性度量

图 6 显示了内核系统调用表遭受 kbeast 修改的情况,从图中可以看到:系统未被篡改前,系统调用表基准 Hahs 值为 4d464c99e15ea2b6ac418475638d671d;篡改后,Hash 值变为了 9c1defbdb629f87261627a8251a97339.通过 Hash 值对比可知,内核系统调用表的完整性遭受破坏.进一步分析可知:系统调用表的第 3 号~第 5 号、第 10 号系统调用入口地址被篡改,替换成了不同的地址,此外还有第 37 号、第 38 号、第 40 号、第 129 号、第 220 号、第 301 号系统调用入口地址遭受篡改.

2. enyelkm

enyelkm 是一款内核级的 rootkit,其替换了 IDT 的 0*x*80 处的 *system_call* 子程序的若干字节和 *sysenter_entry* 入口函数的若干字节为自定义的处理函数,使用户请求的某些系统调用转向攻击者预先设定的处理函数,从而实现攻击.在实验环境中加载 *enyelkm* 后,启动 KIMBMF,读取物理内存中代码段内容,计算其 Hash 值,该值为 8cc11fdd4b5ea5a1d553dfb8cc00697ae0f8e0a1.访问基准库提取代码段的基准值,其值为 11b1624052f5a57c715f95976c5f92a064cc1ea0.对比这两个值即可知道:内核代码段的内容遭受了篡改,内核是不完整的.

3. adore-ng

adore-ng 也是内核级的 rootkit,其利用 inline hook 技术截获 *proc_lookup* 函数,使其执行自定义函数 *adore_lookup* 实现攻击.实验中首先加载 adore-ng,然后利用内置的客户端软件 ava 隐藏系统中某个进程,然后启动完整性度量系统 KIMBMF,分析物理内存中存在的进程,计算 Hash 值 *h*1;同时分析系统自带工具 ps 提供的进程,计算 Hash 值 *h*2.实验结果如图 7 所示,从图 7 中可以看出:*h*1 为 659dad4e7d177b9ff71f7916845a10f75a1b855,*h*2 为 c9b682c0c7502d1f96683031e0956620b34480fd,*h*1≠*h*2,可知系统的完整性遭受破坏.进一步分析发现:系统中隐藏的进程是 gnome-pty-helpe,进程号为 6994.

Fig.7 Integrity measurement of process

图 7 进程完整性度量

从上面 3 个实验可以看出:KIMBMF 实现了内核完整性的度量,能够度量发现系统中存在的攻击.

接下来通过多款不同的 rootkit 来进一步验证 KIMBMF 的有效性,其中涉及到采用不同手段实现攻击的 rootkit,例如:ddrk 既替换系统工具,又修改了系统调用表;wnps 与 enyelkm 类似,对代码段内容进行了修改;allroot 则是实现了提权功能,mood-nt 通过/dev/kmem 重定向系统调用表实现攻击.表 2 给出了 KIMBMF 和其他度量工具在多款 rootkit 攻击系统后度量系统完整性的结果,其中,“√”表示成功度量发现系统遭受攻击,“×”表示未能度量发现系统的完整性遭受破坏,“—”表示该系统未对此款 rootkit 进行测试,其他工具的测试结果来源于各自的参考文献.由表 2 可知:KIMBMF 能够成功地度量发现系统中采用不同攻击方式的 rootkit,能够发现系统遭受篡改的地方,而其他 3 款工具未能度量发现用户态 rootkit 的攻击现象.因此,KIMBMF 较现有的度量工具度量准确率高.

文中提出了随机化时间度量,为了验证采用该思想的 KIMBMF 相对于采用周期性时间进行度量的工具的有效性,将 KIMBMF 和当前几款采用周期性度量的工具进行了对比.周期性度量工具的周期时间设定为 *T*=15s,同时将 rootkit 的活动周期与 *T* 关联,令 rootkit 在[3s,9s]处于活跃期,在[9s,11s]期间消除痕迹,活动次数设定为 100.经过多次运行,统计结果见表 3,其他工具的测试结果是根据它们的工作原理分析得到的.其中,Gibraltar 和

Copilot 都是基于 PCI 周期性地度量内核的完整性,需要硬件的辅助;而 KIMBMF 不需要,硬件开销相对较小;Osck 是基于 KVM 的周期完整性度量系统,通过 VMM,周期性提取内存映像分析度量对象.然而在度量发现 rootkit 的攻击和系统的完整性上,在上述实验条件下,经过多次实验统计,KIMBMF 度量发现 91%的 rootkit 攻击,而 Gibraltar,Copilot 和 Osck 均没有发现 rootkit 攻击.这是由于实验过程中,rootkit 活动周期与这 3 款系统度量周期相关,在它们进行完整性度量时,rootkit 已经不在活跃期且清除了攻击痕迹,所以它们均不能够度量发现系统的不完整性.从对比分析中得知:KIMBMF 发现了 91%的攻击,仅漏报 9%的攻击.漏报的原因是由于度量时间点刚好位于 rootkit 未发生或者消除痕迹后.但上述实验环境是在 rootkit 不间断的攻击情况下进行的,在实际环境中,攻击者一般会选择不规则的间隔周期,所以在现实环境中,漏报率会低于 9%.因此,时间随机化完整性度量方法较周期性完整性度量方法具有更强的度量能力,度量准确率更高.

Table 2 KIMBMF and other tools' measurement for kinds of rootkit attacks

表 2 KIMBMF 和其他工具对系统遭受多种 rootkit 攻击后的度量

rootkit	攻击级别	攻击方式	KIMBMF 度量结果	Copilot 度量结果	Osck 度量结果	Gibraltar 度量结果
ddrk	用户态、内核态	替换系统工具、修改系统调用实现攻击	√	–	–	–
wnps v0.26	内核态	与 enyelkm 类似,修改系统调用子程序若干字节	√	√	√	√
allroot	内核态	修改系统调用表的若干项实现提权	√	√	√	√
mood-nt	内核态	攻击/dev/kmem 重定向系统调用表	√	–	–	–
Knark-2.4.3	内核态	修改系统调用表实现进程文件隐藏	√	√	–	√
suckit v2.0	内核态	攻击/dev/kmem 拦截和修改系统调用实现后门和隐藏功能	√	√	–	–
lrk v5	用户态	替换系统工具实现提权、进程文件隐藏等	√	×	×	×

Table 3 Comparison of KIMBMF and other measurement tools

表 3 KIMBMF 与其他度量工具的对比

度量工具	硬件或者虚拟机	完整性度量结果(百分比)	
		发现次数	漏报次数
Gibraltar	PCI network card	0	100
Copilot	PCI	0	100
Osck	KVM	0	100
KIMBMF	–	91	9

4.2 性能测试

本节采用度量效率测试 KIMBMF 的性能,度量效率是指进行内核完整性度量时的时间开销.测试中,为了提高测试的精确性和科学性,实验重复执行 10 次,实验结果是 10 次实验的平均值.

采用完整性度量时间 T_M 作为性能指标评价 KIMBMF 的度量效率.度量时间是指从内存取证分析到获得完整性度量结果所经历的的时间.一般而言,度量时间越短,度量准确率越高,攻击者躲避度量检测的可能性越小,并且度量时间对周期时间 T 的确定具有指导作用.

对于 KIMBMF,度量时间主要由 3 部分组成.

(1) 内存分析时间 T_{pma}:分析物理内存中的度量内容所需要的时间;

(2) Hash 和加密运算时间 T_{he}:对度量内容进行 Hash 和加密运算所需要的时间;

(3) 完整性度量时间 T_{im}:对内核的完整性进行度量所需要的时间.

综上,KIMBMF 的度量时间 T_M 可表示为

$$T_M=T_{pma}+T_{he}+T_{im} \tag{1}$$

4.2.1 PMCE 测试

欲进行内存分析,首先需要对物理内存内容的访问和提取,故本节在测试 T_M 前先测试 PMCE 的功能和性能,测试过程中选取了不同的物理内存内容提取工具进行了对比分析,其中,fmem[14]和 mem(Linux 系统自带)提

供内存映像文件,提取过程用 dd 工具[14]实现.功能对比测试是测试这些设备随机可选范围内存内容提取的可行性,性能测试是测试读取不同大小内存的时间开销.功能测试结果见表 4.

Table 4 Comparison of PMCE and other tools

表 4 PMCE 与其他工具的功能对比

	PMCE	fmem	mem
内存读取	√	√	√
随机读取	√	×	×
可选范围读取	√	×	×

由表 4 可知:fmem 和 mem 不支持随机和可选范围读取,故测试内存提取耗时均从地址 0 开始读取不同大小的内存内容,分别为 1KB,1MB,100MB 和 1GB,测试结果见表 5,时间单位 s.

Table 5 Time cost comparison of PMCE and other tools

表 5 PMCE 与其他工具读取内存的时间开销对比

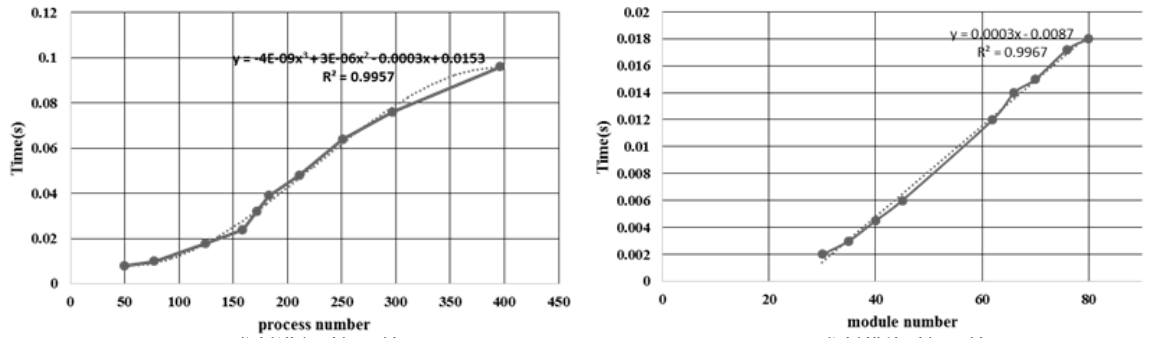
	PMCE	fmem	mem
1KB	0.000 176 959	0.000 148 262	0.000 214 648
1MB	0.119 464	0.116 594 5	0.116 088 5
100MB	0.393 643 6	0.386 751 4	×
1GB	8.901 331	8.891 135	×

从表 5 中可以看出:PMCE,fmem 和 mem 读取 1KB,1MB 内存内容的时间开销相差不多;在读取 100MB,1GB 内存内容上,设备 PMCE 和 fmem 也是相差不大,但是 mem 则不支持 100MB 和 1GB 内容的提取.因此,PMCE 在读取内存内容的时间开销上符合需求.

综上两个方面的测试,PMCE 不仅支持内存内容的提取,而且可读取随机和任意范围的内存内容,时间开销较小.由于内核完整性度量时所要提取的内存内容是离散分布的,fmem 和 mem 无法支持,只有改进的 PMCE 算法满足要求.

4.2.2 T_{pma} 测试

下面测试 PMA 算法分析物理内存中度量内容的时间开销 T_{pma},测试选择的度量内容包括中断向量表、系统调用表、中断处理函数、系统调用处理函数、内核代码段、进程和模块,其中,进程和模块是动态变化的内容.测试结果分两部分:一是分析静态内容的时间开销,见表 6.二是分析进程和模块列表的时间开销,如图 8 所示.



(a) 分析进程时间开销 (b) 分析模块时间开销

Fig.8 Time cost of PMA algorithm for dynamic measure objects analysis

图 8 PMA 算法分析动态度量对象的时间开销

Table 6 Time cost of PMA algorithm for static measure objects analysis

表 6 PMA 算法分析静态度量对象的时间开销

	SYSCALL	IDT	IDTFUNC	SYSCALLFUNC	KERNELCODE	总数
比特数	1 372	2 048	1 577 098	2 034 403	6 724 012	10 338 933
Time (ms)	0.008 21	0.012 53	9.4373	12.173 8	40.253 1	61.884 94

表 6 中的数据表明:PMA 算法分析静态内容的时间与内容的大小相关,随着内容的增多,分析时间越长,总体时间开销在 61.884 94ms 左右.图 8 给出了不同的进程数和模块数情况下 PMA 算法分析所需要的时间,分析时间与进程数呈多项式递增趋势,与模块数呈线性递增趋势.这是由于分析过程是通过遍历链表实现的,链表越复杂,时间越长.当进程数为 400 时,分析时间为 100ms,模块数为 55,分析时间为 10ms.

4.2.3 T_{he} 和 T_{im} 测试

Hash 和加密运算是完整性度量过程的重要步骤,图 9 给出了不同时刻对不同的度量内容进行运算所需要的时间开销 T_{he},其中,Hash 运算时间为黑色线,Hash 和加密运算总时间为灰色线.

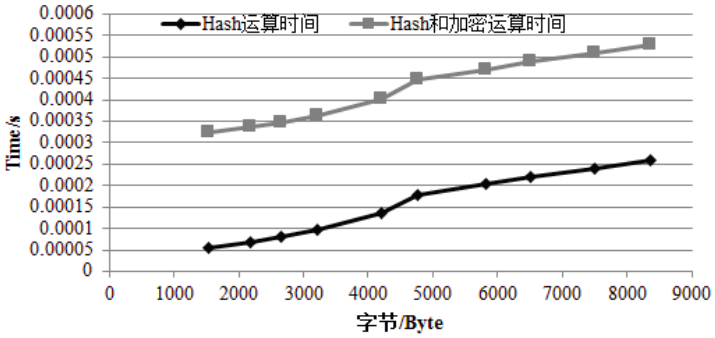


Fig.9 Time cost of Hash operation and cryptographic operation

图 9 Hash 运算和加密运算时间开销

从图 9 中可以看出,Hash 运算时间和总时间与内容大小呈线性正比关系.这与预期想法是一致的,Hash 和加密运算都是按字节进行运算的,由于 Hash 运算结果是固定大小的 160bit 值,加密所需要的时间是固定的,故总时间的变化关系只依赖于 Hash 运算时间.

完整性度量时间主要包括取基准值、解密和完整性匹配的时间,取基准值是从基准库中提取对应度量对象的基准值,解密是负责将经过加密的基准值解密得到 Hash 值,完整性匹配完成对运行过程中的度量对象的 Hash 值与基准值的匹配分析.表 7 中给出了各个部分所需要的时间和总时间,其中,取基准值所需要时间最长,最终完整性度量时间 T_{im} 约为 1.6ms.

Table 7 Time cost of part of measure process

表 7 度量过程各部分时间开销

	取基准值	解密	完整性匹配	完整性度量
Time (ms)	0.993 7	0.492	0.131	1.616 7

从上面分别测试可以得出:当系统进程数为 200、模块数为 50 时,公式(1)的值约为 349.636 6ms.其中,对度量效率影响最大的是 Hash 和加密运算阶段,其次是分析内存提取度量对象阶段,各环节占总时间的百分比如图 10 所示,总时间为 ms 数量级,对系统的造成的负担很小,并且由于 rootkit 活跃期和周期时间 T 一般都在 10s 数量级以上,所以只要度量点处于 rootkit 活跃期,均能度量发现系统中 rootkit 的存在.

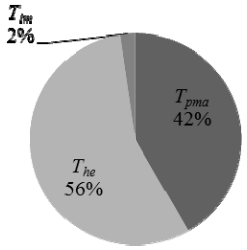


Fig.10 Breakdown of measure time

图 10 度量时间分解

Gibraltar 一次检测时间约为 20s,因为其需要对整个内存进行扫描;Osck 针对某个 rootkit 的检测时间约为 0.5s,即 500ms,与基于内存取证的完整性度量方法时间相近,但是基于内存取证的完整性度量方法所需要的时间并不是检测某一个 rootkit 的时间,有若干个 rootkit 的情况其也可以在 ms 数量级的时间内实现,并且 Osck 基于 KVM 带来了一定的系统负载;Copilot 中并没有明确指出一次度量所需要的时间,只给出了其度量周期.此外,由于 KIMBMF 通过分析内存实现完整性度量,故选择了 STREAM benchmark 测试系统负载.经测试, KIMBMF 在 T 为 15s 的情况下对系统的负载仅为 0.27%,相对于 Osck 的 2%~6%、SBCFI 的 3.6%~48.3%而言, KIMBMF 对系统的负载可以忽略不计;与 Gibraltar 的 0.49%、Copilot 的 0.84%相比,KIMBMF 不仅负载较小,而且不需要硬件辅助.

此外,为了测试 KIMBMF 占用 CPU 对系统的性能影响,我们通过 top 命令监控 KIMBMF 的 CPU 占用率.经过 1 个小时的监控,KIMBMF 的 CPU 最高占用率仅为 6%,平均占用率为 2%.这是因为 KIMBMF 采用的随机化时间间隔运行机制,只有在某给定时间内才会运行,从而占用 CPU.故,可认为 KIMBMF 占用 CPU 对系统的性能影响可忽略不计.

因此,通过与这些工具度量或检测所需要的时间和系统负载对比,基于内存取证的完整性度量方法在时间开销和系统负载上具有一定的优势.

5 结论与下一步工作

本文针对现有的完整性度量方法中存在硬件开销、TOC-TOU 问题和 VMM 带来的性能损失,提出了基于内存取证分析的内核完整性度量方法 KIMBMF.它采用内存取证技术,首先解决如何获取内存内容的问题,通过 PMCE 算法实现对物理内存的访问和内容的提取;然后解决了如何分析内存内容获取度量对象的问题.通过关注 rootkit 攻击内核的方法,总结 rootkit 攻击的内核对象,以内核符号表和内核关键数据结构为入口点分析提取度量对象;同时提出时间随机化算法弱化 TOC-TOU 问题,提高度量的准确性;在度量过程中,将 hash 运算和加密运算相结合加强度量安全性,避免攻击者伪造基准值躲避完整性度量.通过实验研究与分析:KIMBMF 能够有效的度量内核的完整性,时间开销保持在 ms 数量级,对系统的负担影响很小,且与周期性度量工具相比,其度量能力及准确率更高.

实验虽然验证了时间随机化度量比周期性度量具有较高的度量准确率,但该方法能够弱化而不是完全避免 TOC-TOU 攻击,从实验数据可知,该系统未能发现 9%的 rootkit 攻击.此外,由于有些攻击方法并不对内核的任何地方进行修改实现提权、控制流劫持等,进而破坏内核的完整性,如 ROP 攻击[24],而现有的完整性度量方法(包括本文的方法)需要发现修改内核或者破坏内核的某些特性才能够发现内核的不完整性,所以它们目前无法度量发现这类攻击,这也涵盖在 9%内.下一步的工作将在现有工作的基础上着重研究访问控制、轻量级虚拟监控、内核级恶意软件数据特征等技术,进一步提高完整性度量的有效性和准确率.

致谢 在此,我们向对本研究工作提供帮助的老师和同学表示感谢.同时,我们也向对本文提出宝贵意见的评审

专家表示感谢.

References:

[1] Wang YM, Beck D, Vo B, Roussev R, Verbowski C. Detecting stealth software with strider ghostbuster. In: Proc. of the Int'l Conf. on Dependable Systems and Networks. Washington: IEEE CS Press, 2005. 368−377. [doi: 10.1109/DSN.2005.39]

[2] Joy J, John A. A host based kernel level rootkit detection mechanism using clustering technique. In: Proc. of the Int'l Conf. on Computer Science, Engineering and Information Technology. Berlin, Heidelberg: Springer-Verlag, 2011. 564−570. [doi: 10.1007/978-3-642-24043-0_57]

[3] Liu ZW, Feng DG. TPM-Based dynamic integrity measurement architecture. Journal of Electronics & Information Technology, 2010,32(4):875−879 (in Chinese with English abstract). [doi: 10.3724/SP.J.1146.2009.00408]

[4] Bratus S, D'Cunha N, Sparks E, Smith SW. TOCTOU, traps, and trusted computing. In: Proc. of the 1st Int'l Conf. on Trusted Computing and Trust in Information Technologies. Berlin, Heidelberg: Springer-Verlag, 2008. 14−32. [doi: 10.1007/978-3-540-68979-9_2]

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

[6] Hofmann OS, Dunn AM, Kim S, Roy I, Witchel E. Ensuring operating system kernel integrity with osck. In: Proc. of the 6th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. ACM Press, 2011. 279−290. [doi: 10.1145/1950365.1950398]

[7] Peter AL, Mcgill KN. LKIM: Linux kernel integrity measurer. Johns Hopkins APL Technical Digest, 2013,32(2):509−516.

[8] Rhee J, Riley R, Lin ZQ, Jiang XX, Xu DY. Data-Centric OS kernel malware characterization. IEEE Trans. on Information Forensics and Security, 2014,9(1):72−87. [doi: 10.1109/TIFS.2013.2291964]

[9] Petroni NL, Fraser T, Molina J, Arbaugh WA. Copilot—A coprocessor-based kernel runtime integrity monitor. In: Proc. of the 13th Conf. on USENIX Security Symp. Berkeley: Usenix, 2004. 179−194.

[10] Arati B, Vinod G, Liviu I. Detecting kernel-level rootkits using data structure invariants. IEEE Trans. on Dependable and Secure Computing, 2011,8(5):670−684. [doi: 10.1109/TDSC.2010.38]

[11] Moon H, Lee H, Lee J, Kim K, Paek Y, Kang BB. Vigilare: Toward snoop-based kernel integrity monitor. In: Proc. of the 19th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2012. 28−37. [doi: 10.1145/2382196.2382202]

[12] Liu Z, Lee JH, Zeng J, Wen YF, Lin ZQ, Shi WD. CPU transparent protection of OS kernel and hypervisor integrity with programmable DRAM. In: Proc. of the 40th Annual Int'l Symp. on Computer Architecture. ACM Press, 2013. 392−403. [doi: 10.1145/2508148.2485956]

[13] Stevens RM, Casey E. Extracting Windows command line details from physical memory. Digital Investigation, 2010,7(8):57−63. [doi: 10.1016/j.diin.2010.05.008]

[14] James SO, Gilbert LP. Windows driver memory analysis: A reverse engineering methodology. Computers and Security, 2011,30(8):770−779. [doi: 10.1016/j.cose.2011.08.001]

[15] Kollar I. Forensic RAM dump image analyzer [MS. Thesis]. Department of Software Engineering, Charles University of Prague, 2010.

[16] Sylvea J, Caseb A, Marzialeb L, Richarda GG. Acquisition and analysis of volatile memory from android devices. Digital Investigation, 2012,8(3-4):175−184. [doi: 10.1016/j.diin.2011.10.003]

[17] Carvey H. Windows Forensic Analysis. 2nd ed., Elsevier: Syngress, 2009. 59−63.

[18] Vidstrom A. Memory dumping over firewire-uma issues. 2006. http://ntsecurity.nu/onmymind/2006/2006-09-02.html

[19] Gladyshev P, Almansoori A. Reliable acquisition of RAM dumps from Intel-based Apple Mac computers over FireWire. In: Proc. of the 2nd Int'l Conf. on Digital Forensics and Cyber Crime. Berlin, Heidelberg: Springer-Verlasg, 2010. 55−64. [doi: 10.1007/978-3-642-19513-6_5]

[20] Wang LH. Model and methods research on computer live forensics based on physical memory analysis [Ph.D. Thesis]. Ji'nan: Shandong Univerisity, 2014 (in Chinese with English abstract).

[21] LiME. 2012. http://code.google.com/p/lime-forensics

[22] Petroni NL. Property based integrity monitoring of operating system kernels [Ph.D. Thesis]. University of Maryland, 2008.

[23] Riley R. A framework for prototyping and testing data-only rootkit attacks. Computers and Security, 2013,37:62−71. [doi: 10.1016/j.cose.2013.04.006]

[24] Pappas V, Polychronakis M, Keromytis AD. Transparent ROP exploit mitigation using indirect branch tracing. In: Proc. of the 22nd USENIX Security Symp. Washington: Usenix, 2013. 447−462.

附中文参考文献:

[3] 刘孜文,冯登国.基于可信计算的动态完整性度量架构.电子与信息学报,2010,32(4):875−879.

[20] 王连海.基于物理内存分析的在线取证模型与方法的研究[博士学位论文].济南:山东大学,2014.

陈志锋(1986－),男,福建漳州人,博士,讲师,主要研究领域为信息安全,可信计算.

张平(1969－),女,博士,副教授,主要研究领域为并行识别,并行编译,信息安全.

李清宝(1967－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信息安全,可信计算.

王炜(1975－),男,博士,副教授,CCF 会员,主要研究领域为计算机系统结构,信息安全.