软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(11):3588−3602 [doi: 10.13328/j.cnki.jos.005829]

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

基于硬件分支信息的 ROP 攻击检测方法*

李威威[1,2], 马 越[1], 王俊杰[1], 高伟毅[1,2], 杨秋松[1], 李明树[1]

[1](中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

[2](中国科学院大学,北京 100049)

通讯作者: 李威威, E-mail: weiwei@nfs.iscas.ac.cn

摘 要: 控制流完整性保护技术(control flow integrity,简称 CFI)是防御面向返回编程攻击(return-oriented programming,简称 ROP)的一种有效途径.针对现有 CFI 中存在的四大问题:性能开销大、依赖程序代码信息、容易遭受历史刷新攻击以及规避攻击,提出了基于硬件分支信息的 ROP 攻击检测方法——MIBChecker(mispredicted indirect branch checker).该方法实时地利用硬件性能管理单元(performance monitor unit,简称 PMU)的事件触发机制,针对每个预测失败的间接分支进行ROP攻击检测,规避了历史刷新攻击的可能,同时提出基于敏感系统调用参数的新型检测方法来检测短攻击链(称为 gadgets-chain)ROP 攻击.实验结果表明,MIBChecker 能够不受历史刷新攻击的影响进行ROP短指令片段(称为gadget)检测,可有效地检测出常规ROP攻击和规避攻击,并仅引入5.7%的性能开销.

关键词: 面向返回编程;控制流完整性;历史刷新攻击;规避攻击

中图法分类号: TP309

中文引用格式: 李威威,马越,王俊杰,高伟毅,杨秋松,李明树.基于硬件分支信息的 ROP 攻击检测方法.软件学报,2020,31(11):3588−3602. http://www.jos.org.cn/1000-9825/5829.htm

英文引用格式: Li WW, Ma Y, Wang JJ, Gao WY, Yang QS, Li MS. ROP attack detection approach based on hardware branch information. Ruan Jian Xue Bao/Journal of Software, 2020,31(11):3588−3602 (in Chinese). http://www.jos.org.cn/1000-9825/5829.htm

ROP Attack Detection Approach Based on Hardware Branch Information

LI Wei-Wei[1,2], MA Yue[1], WANG Jun-Jie[1], GAO Wei-Yi[1,2], YANG Qiu-Song[1], LI Ming-Shu[1]

[1](National Engineering Research Center of Fundamental Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Control flow integrity (CFI) is an effective method to defend against return-oriented programming (ROP) attack. To address the four drawbacks of current CFI approaches, i.e., high performance overhead, relying on software code information, subject to history flushing attack, and evasion attack, this study proposed an ROP attack detection approach based on hardware branch information—mispredicted indirect branch checker, called MIBChecker. It performs real time ROP detection on every mispredicted indirect branch via events triggered by performance monitor unit, and produces a new critical syscall data detection approach to defend against ROP attacks using short gadgets-chain. Experiments show that MIBChecker can detect gadgets which is not affected by history flushing attack, and can effectively detect common ROP attack and evasion attack with only 5.7% performance overhead.

Key words: return-oriented programming; control flow integrity; history flushing attack; evasion attack

ROP 攻击是当前最主要的控制流劫持攻击方式之一,如何防御 ROP 攻击,是计算机安全领域近年来研究的

* 基金项目: “核高基”国家科技重大专项(2014ZX01029101-002); 中国科学院战略性先导科技专项(XDA-Y01-01)

Foundation item: National Science and Technology Major Program for Core Electronic Device, High-end Chip, and Basic Software Product (2014ZX01029101-002); Strategic Priority Research Program of Chinese Academy of Sciences (XDA-Y01-01)

收稿时间: 2018-07-14; 修改时间: 2018-11-06; 采用时间: 2019-02-28

重点.ROP攻击是一种利用代码复用技术的攻击方法,攻击者通过内存漏洞劫持程序的控制流,将代码中已有的以间接分支指令结尾的短指令片段(称为gadget)串联起来(组成gadgets-chain),以实现预设的攻击目标[1−4].

CFI技术直接针对程序控制流进行保护,是一种防御ROP攻击的有效途径.CFI技术通过监视程序运行过程中的控制流转移过程,使其始终处于原有CFG(control flow graph)所限定的合法范围内.具体的做法是:分析程序的控制流图,重点关注间接分支指令,包括间接jmp、间接call和ret指令;在程序运行过程中,对间接分支指令的跳转目标进行检测,当攻击者劫持控制流导致分支目标偏离控制流图时,可迅速进行阻断,保证系统安全[5].

然而,现有的CFI技术通常面临着如下几大问题.

1) 传统CFI技术为了保证控制流检测的完备性,需要针对所有的间接分支进行检测,检测点多,性能开销较大[5,6].
2) 基于控制流图的粗粒度CFI技术为了在间接分支处插入检测点,需要修改源码或者通过反汇编重写二进制代码插入检测代码,依赖于源码或者调试信息,难以适用于部分传统应用程序,限制了方法的实用性[7−10].
3) 现有的硬件辅助的CFI技术为了降低性能开销,通常在API函数调用或者窗口页切换处利用存储的分支历史信息进行窗口式检测[11,12],无法做到对单个分支的实时检测,检测之前可能遭受历史刷新攻击[13]:攻击者在触发检测之前插入合法的程序流,覆盖用于ROP攻击的分支历史信息.
4) 基于特征检测的CFI技术为了与正常程序流中的可能出现的gadgets-chain区分开,通常仅根据长gadgets-chain特征判定ROP攻击[11,12,14],难以对利用短gadgets-chain的规避攻击进行有效的检测.

针对这些问题,本文从降低检测频率、运行时提取检测所需信息、实时检测以及新型短gadgets-chain ROP攻击检测方法这4个角度出发,提出了MIBChecker(mispredicted indirect branch checker)ROP攻击检测方法.

1) 利用ROP攻击所使用的分支会导致BPU(branch processing unit)预测失败这一特性,仅针对预测失败的间接分支进行检测,在保证控制流检测完备性的同时,降低了性能开销.
2) 检测所需信息可直接从运行时获取,包括:a) PMI(performance monitor interrupt)中断保存的系统状态信息;b) 通过LBR(last branch record)获取的分支地址信息;c) 根据分支地址获取的分支指令信息.这些信息的提取过程不依赖源码和调试信息,同时也能保证所提取信息的有效性.
3) 检测在每个预测失败的间接分支处触发,分支相关LBR历史信息是被立即使用的,因此,历史刷新攻击不影响MIBChecker的gadget检测.
4) 提出了敏感系统调用参数检测方法,在敏感系统调用处通过判断敏感系统调用的参数是否来源于之前的gadget来检测ROP攻击,能够检测已有方法[11,12]检测不到的短gadgets-chain规避攻击.

MIBChecker方法具备以下优点.

1) 安全:在每个可能用于ROP攻击的分支处立即触发检测,仅使用LBR中当前分支的信息,从根源上避免了gadget检测所需历史信息被刷新的可能;同时,以敏感系统调用作为检测点,通过系统调用参数检测方法判断敏感系统调用参数是否来源于短gadgets-chain,进而能够识别出通过短gadgets-chain进行敏感系统调用的规避攻击.
2) 透明:通过硬件PMU机制,能在预测失败的间接分支处自动触发PMI中断进行ROP检测,检测所需信息也直接从LBR和运行时获取,整个过程对用户级程序透明.
3) 性能开销低:仅针对预测失败的间接分支进行ROP攻击检测,避免了对预测正确的间接分支进行检测,能够有效减少检测点,控制了性能开销,同时,硬件触发检测相比软件动态插桩式检测也更为高效.

本文第1节介绍本文的相关研究背景.第2节对本文提出的MIBChecker检测方法进行详细介绍.第3节详细描述MIBChecker的实现.第4节对MIBChecker进行安全性及性能评估.第5节对本文进行总结.

1 相关研究

CFI作为一种有效的ROP攻击防御技术,最早由Abadi等人提出[5],该方法假设程序在其执行过程中应当遵

循预先定义好的 CFG,以确保程序控制流不被劫持或非法篡改.具体做法是,在分支指令前插入检测代码来判断目标地址的合法性.CFI 技术分为细粒度和粗粒度两种类型.

Abadi 等人[5]提出了最早的细粒度 CFI 技术,该技术预先计算出所有间接分支指令可能的目标地址,并为之赋予唯一的 ID;在每条间接跳转指令执行前插入检测代码,判断 ID 是否合法,一旦发现违反 CFG 的间接分支就会被识别为攻击.Abadi 等人[6]利用 CFI 思想保证控制流的正确性,并结合内存访问控制、影子栈等,在运行时对传统的应用程序提供保护与监控.虽然细粒度的 CFI 技术针对所有分支进行检测,能够提升系统的安全性,但其性能开销超过 15%,难以得到实际部署.

粗粒度的 CFI 技术将一组相同或相近类型的目标归到一起进行检测,可在一定程度上降低性能开销.Zhang 等人[7]提出了 CCFIR 方法,该方法对间接 call 指令和 ret 指令的目标进行区分,阻止未经验证的 ret 指令跳转到敏感函数的行为.Zhang 等人[8]提出了 binCFI,该方法将间接分支指令的操作数分为代码指针、异常处理程序入口和返回地址等类型,通过精细的静态分析,得到不同类型间接分支指令的合法目标集合,并基于此进行攻击检测.Mashtizadeh 等人[9]提出了 CCFI 方法对代码指针进行更细致的划分,该方法将代码指针细分为函数指针类、返回指针类、异常处理函数指针类、虚表指针类,通过对代码指针进行加密来增强 CFI 方法.以上方法都是只实施了控制流不敏感策略,然而上下文信息的缺少会降低方法的有效性.因此,Veen 等人[10]提出了上下文敏感的 CFI 技术,该方法能够监控通往敏感函数(可用于实施控制流劫持攻击)的执行路径,通过使用二进制插桩技术,在被监控路径上强制执行上下文敏感的不变量.

以上这些粗粒度 CFI 技术都需要修改源码或者通过反汇编重写二进制代码(通常需要依赖调试信息以保证其正确性),这给这些技术的使用带来了额外的限制.本文提出的 MIBChecker 方法直接利用硬件获取运行时信息进行检测,不依赖于源码及调试信息,可以直接作用于传统二进制程序.

此外,一些研究通过利用现有硬件机制来降低 CFI 技术的性能开销.Pappas 等人提出了 kBouncer 方法[11],该方法利用 LBR 捕获最近的 16 次分支信息,在系统调用处对捕获的 16 次分支进行安全性检测.Cheng 等人提出了 ROPecker 方法[12],该方法也是利用 LBR 捕获程序控制流信息的方式进行 ROP 攻击检测.该方法在运行时检测过去和未来的执行流中是否存在长 gadgets-chain 来进行攻击检测,还通过滑动窗口的机制来进一步提高准确性和高效性.但是,这两种方法都是一次性针对整个 LBR 进行检测,会面临历史刷新问题,容易遭受历史刷新攻击[13].Xia 等人提出了 CFIMon[14],该方法采用 BTS(branch trace store)机制来捕获程序运行过程中分支指令的信息.虽然 BTS 能够将程序整个执行过程中的所有分支指令的历史信息都记录下来,但相比于 LBR,使用 BTS 会引入更大的性能开销.此外,以上 3 种方法均是针对长 gadgets-chain 的 ROP 攻击进行检测,对于由短 gadgets-chain 进行的规避攻击检测效果不佳.

本文提出的 MIBChecker 方法针对每个预测失败的间接分支立即从 LBR 获取当前分支信息进行检测,能够从根源上避免 LBR 历史刷新攻击问题.进一步的,本方法提出了系统调用参数检测方法,能够有效地检测出通过短 gadgets-chain 进行的规避攻击.

除了利用现有硬件技术以外,一些研究还通过设计自身硬件来进行 CFI 检测,此类方法通常通过硬件影子栈以及扩展新的指令集来实现 CFI 检测策略[15],或者通过修改分支指令的硬件逻辑来限制跳转目标,如限制函数间的跳转[16]、限定跳转目标为基本代码块的开始[17]等.此类方法能够降低性能开销,但都需要新的硬件来支持,短期内难以推广,实用性较差.

除了 CFI 方法,另一个 ROP 攻击防御方面的研究热点是随机化技术[18−22],该类技术通过将函数、内存页、基础块或指令排布等随机化,让攻击者难以准确预测所需 gadget 的位置,进而无法进行 ROP 攻击.该类方法主要提供概率性保护,能够增大攻击的难度,本文的方法可以和这类方法一起作用,用于阻止攻击者对应用程序进行 ROP 攻击.

2 MIBChecker——基于硬件分支信息的 ROP 攻击检测方法

本文提出了 MIBChecker 方法,该方法的基本原理如图 1 所示.

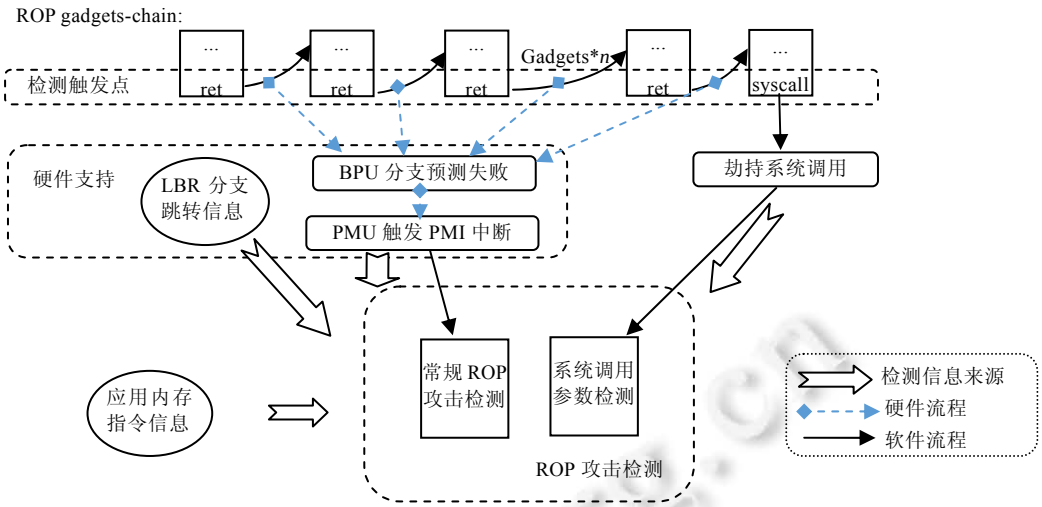


Fig.1 Principle of MIBChecker

图 1 MIBChecker 原理

MIBChecker 以可能用于攻击的所有间接分支以及敏感系统调用作为检测触发点,利用运行时获取的信息进行 ROP 攻击检测.本节将先描述 MIBChecker 针对的威胁模型以及利用的硬件资源,然后对检测信息来源、检测触发点的选择以及检测算法进行详细描述.

2.1 威胁模型

MIBChecker 面向用户态程序进行 ROP 攻击检测,基于的威胁模型如下.

攻击者具备如下能力:能够在远程或本地正常终端以普通用户权限访问目标应用;能够利用目标应用程序中的内存漏洞读取或修改任意用户态内存内容;能够通过应用内存漏洞获取所有用户态内存空间排布信息.

但是假定硬件、操作系统、动态链接器是可信的,目标操作系统支持 DEP(data execution prevention)技术.因此,攻击者不具备以下能力:不能直接控制硬件及操作系统运行;不能直接利用内存漏洞插入恶意代码(代码注入攻击)或直接修改应用代码.

总的来说,在该模型下,攻击者仅能通过应用本身的漏洞复用应用本身代码进行攻击.

2.2 硬件分支信息

MIBChecker 方法基于硬件分支信息进行检测,需要借助如下硬件资源获取分支相关信息.

- BPU:是现代处理器(如 Intel、AMD 等各类产品)进行分支指令预测的单元.它主要用于记录分支跳转历史信息,并根据历史信息对后续分支指令的跳转方向及目标地址进行预测.MIBChecker 利用 BPU 的分支预测特性,将预测正确的间接分支(即历史上已经发生过的合法分支)排除在检测范围之外.
- PMU:是现代处理器中用于记录处理器事件的功能单元.现代 CPU 支持大量的性能监控事件,包含了处理器在运行过程中可能遇到的各种情形,例如指令计数、浮点运算指令计数、L2 缓存未命中的时钟周期、分支预测失败事件等.用户可以根据具体需求设置事件选择寄存器(IA32_PERFEVTSEL)选取需要采样的事件类型.PMU 会根据用户所选的事件类型对处理器上发生的该事件进行计数,当计数器溢出时,可以触发 PMI 中断,用户可以在中断处理过程中进行相关处理.MIBChecker 利用 PMU 采样预测失败的间接分支事件触发 PMI 中断作为检测点进行 ROP 攻击检测.
- LBR:是 x86 架构中提供的一组用于记录和追踪程序最近的若干次分支跳转信息的循环寄存器组,它们会记录跳转分支指令的地址信息{from_ip,to_ip},即分别对应于跳转源地址和目标地址.这些寄存器的数量与处理器的微架构相关.另外,LBR 寄存器具备过滤功能,能够选择性地滤掉一些不需采样的分

支指令.MIBChecker 利用 LBR 的过滤功能,让 LBR 只针对间接分支指令信息进行采样,并在运行时从 LBR 中提取这些分支信息用于 ROP 攻击检测.

2.3 检测触发点选择

MIBChecker 采用实时检测的方法来避免遭受历史刷新攻击,需要针对所有可能用于 ROP 攻击的分支指令进行检测.为了达成这个目标,MIBChecker 引入了 PMU 的事件触发机制,让 PMU 在这些分支处自动触发事件进入事件处理程序进行 ROP 攻击检测.相比于软件动态插桩触发的方式,PMU 的事件触发机制更加简便、高效,而且触发过程可对应用程序透明.

然而,如果针对所有间接分支事件进行检测,会导致检测频率过高,引入较大的性能开销.因此,本文需要在分支相关事件中进行筛选,在降低事件触发频率的同时,不遗漏任何用于 ROP 攻击的分支.

本文结合 ROP 攻击特征及硬件分支处理相关特性进行了待检测分支的进一步筛选.

1) 用户态分支和内核态分支:本文针对应用态程序进行保护,假定操作系统是可信的,因此在提权之前,用于 ROP 攻击的分支均为用户态分支.
2) 预测正确分支和预测失败分支:现代CPU的分支预测机制主要基于历史信息来对分支进行预测,由于 ROP 攻击会让分支偏离原有的跳转目标,导致其与正常历史信息不一致,使得分支预测失败,因此仅需对预测失败的分支进行 ROP 攻击检测.
3) 跳转分支和不跳转分支:ROP 攻击链衔接所需要的分支应该均为跳转的分支,如果不跳转,那么直接执行下一条指令,攻击者不能利用该分支跳转到预期的目标.因此,能用于 ROP 攻击中衔接的分支均为跳转的分支.
4) 直接分支和间接分支:直接分支其目标已确定,在不能直接修改程序代码的前提下,攻击者无法对控制流进行有效控制.因此,用于 ROP 攻击的 gadget 是以间接分支结尾的指令片段,针对 ROP 攻击的检测仅需对间接分支进行.
5) 执行分支和提交分支:现代 CPU 为了提高效率基本都支持乱序执行技术,部分分支会被投机执行,但可能会被抛弃,最终不会提交,所以分支可分为执行的分支和提交的分支.ROP 攻击过程中的分支是程序真正运行时用到的分支,都属于会提交的分支(只是攻击者破坏了控制流完整性,使其偏离了程序开发人员的本意),因此仅需对提交的分支进行 ROP 攻击检测.

综合以上特征分析,我们所需检测的最理想分支为满足用户态、预测失败、跳转、间接以及提交这 5 大特性的分支.

除了分支以外,为了识别短 gadgets-chain ROP 攻击,MIBChecker 还将敏感系统调用作为检测触发点,通过劫持系统调用进行 ROP 攻击检测.这类敏感系统调用包括 mprotect、execve、mmap、sendmsg、remap_file 等,它们可以关闭 DEP(data execution prevention)、直接执行非预期命令、写本地文件或发送信息到网络上,是攻击者常选择的 ROP 攻击途径.

2.4 检测信息来源

MIBChecker 进行 ROP 攻击检测所需信息包括 4 个来源.1) LBR 分支信息:硬件在执行跳转分支指令时,会记录最近分支的信息.2) PMI 中断处理保存的系统状态信息:操作系统处理 PMI 中断前,会自动保存当前的系统状态.3) 系统调用参数信息:劫持敏感系统调用时,直接通过系统调用本身获取调用参数信息.4) 程序指令信息:内核中的检测模块可直接根据地址获取应用程序内存信息.

这些信息均可在运行时获取,不依赖于源码及调试信息,因此信息提取过程可对应用程序透明.

2.5 ROP攻击检测算法

MIBChecker 的 ROP 攻击检测算法主要分为两个部分:常规 ROP 攻击检测以及系统调用参数检测两类.

2.5.1 常规 ROP 攻击检测

MIBChecker 采用了与其他基于特征的 ROP 攻击检测方法类似的常规 ROP 攻击检测方法:基于 ROP 攻击

的“短”或“违规”gadget、“长”gadgets-chain 特征进行检测.设定两个阈值:gadget 指令片段最长长度(*maxGadgetLength*)以及 ROP gadgets-chain 最短长度(*minChainLength*).

- Gadget 的“短”特性

如果从上一个检测分支入口到当前分支的指令的长度低于 *maxGadgetLength* 阈值,则将该指令片段识别为 gadget,即

$$isGadget=LBR(cur).from\_ip-LBR(last).to\_ip<maxGadgetLength \tag{1}$$

其中,*LBR*(*cur*)和 *LBR*(*last*)分别表示当前检测分支的 LBR 信息和上一次进行检测时保存的 LBR 信息.

- gadget 识别方法

从 LBR 中获取当前分支的 from_ip 信息,和保存的上一次分支 to_ip 信息进行比较,如果满足上述条件,那么将当前指令片段识别为 gadget;在 gadget 检测后,从 LBR 中获取当前检测分支的 to_ip 信息进行保存(用于下一次检测).

- Gadget 的“违规”特性

正常情况下,ret 指令会返回到 call 指令的下一条指令处,而 ROP 攻击过程中使用的 gadget 通常会破坏这个特性.

对于所有的 ret 指令,如果其分支目标的上一条指令不是 call 指令,则将该指令片段识别为 gadget,即

$$isGadget=Cur\_instr(LBR(cur).from\_ip)==\text{“}RET\text{”}\ \&\&\ Last\_instr(LBR(cur).to\_ip))!=\text{“}CALL\text{”} \tag{2}$$

其中,*Cur_instr*(*addr*)和 *Last_instr*(*addr*)分别表示地址 *addr* 对应的指令和它的上一条指令.

- gadget 识别方法

利用硬件 LBR 中的分支地址(from_ip 和 to_ip)访问应用内存,分别获取当前分支指令信息以及目标地址上一条指令信息,然后对它们进行动态反汇编来判断指令类型,如果满足上述条件,那么将当前指令片段识别为 gadget.

- gadgets-chain 的“长”特性

当 gadgets-chain 长度超过 *minChainLength* 阈值时,那么该 gadgets-chain 判定为用于 ROP 攻击的 gadgets-chain,受保护程序遭受了 ROP 攻击,即

$$isROPAttack=gadgetsChainLength>minChainLength \tag{3}$$

- ROP 攻击判定方法

在每次检测到 gadget 时,累计连续的 gadgets(gadgets-chain)长度 *gadgetsChainLength*,如果 *gadgetsChainLength* 大于 *minChainLength*,那么认为程序遭受了 ROP 攻击.

2.5.2 Gadgets-chain 长度累计优化

gadgetsChainLength 记录连续的 gadget 数目.常规 ROP 攻击中采用的每个 gadget 都具备一定功能,gadgets-chain 长度统计实际上就是对连续的各种子功能片段进行统计.

然而在深度递归调用返回时,单个 ret 指令片段如果较短,那么就有可能被 MIBChecker 识别成 gadget,那么重复的递归返回就可能构成超长的 gadgets-chain,导致误报.

本文将这种连续重复使用的 gadget 称为递归 gadget.由于连续重复的 gadget 实际上只能实现单一功能,因此,MIBChecker 对常规的 gadgets-chain 累计方法进行了优化,在 gadgets-chain 长度统计中仅需要将第一个 gadget 纳入统计,即不对递归 gadget 进行统计,这样既符合我们统计 gadgets-chain 长度的初衷——统计连续的子功能片段的个数,又可以避免深度递归调用中重复递归返回引起的误报问题.

如果当前 gadget 和上一 gadget 一致,那么将当前 gadget 识别为递归 gadget,即

$$\left.\begin{aligned}&isRecusiveGadget=isGadget(cur)\ \&\&\ isGadget(last)\ \&\&\\&LBR(cur).from\_ip=LBR(last).from\_ip\ \&\&\ LBR(cur).to\_ip=LBR(last).to\_ip\end{aligned}\right\} \tag{4}$$

其中,*isGadget*(*cur*)和 *isGadget*(*last*)表示当前指令片段和上一指令片段均为 gadget,*LBR*(*cur*),*LBR*(last)和特征 1 描述中具备一样的含义.

- 递归 gadget 识别方法

首先,利用特征 1 和特征 2 判断当前指令片段是否为 gadget,如果是,且之前的 gadget-chain 长度不为 0,那么从 LBR 中获取当前分支的 from_ip 和 to_ip 信息,和保存的上一个分支的 from_ip 和 to_ip 信息进行比较:如果一致,则将该片段识别为递归 gadget.在 gadget 检测后,从 LBR 中获取当前检测分支的 from_ip 和 to_ip 信息进行保存(用于下一次检测).

MIBChecker 采用如下算法来统计 gadgets-chain 的长度(*gadgetsChainLength*).

1) 初始 gadgets-chain 长度为 0.
2) 如果当前指令片段被识别为递归 gadget,那么 gadgets-chain 长度不变.
3) 如果当前指令片段被识别为 gadget(非递归 gadget),那么将 gadgets-chain 长度加 1.
4) 非情形 2)和情形 3)的情形,则将 gadgets-chain 长度清 0.

2.5.3 系统调用参数检测方法

常规的 ROP 攻击检测方法可能被利用短 gadgets-chain 进行的规避攻击绕过.而通过短 gadgets-chain 进行的规避攻击只能通过较少的步骤达到攻击目的,通常会采用最直接的办法——快速构造系统调用参数,执行敏感系统调用.因此针对此类攻击,MIBChecker 提出了系统调用参数检测方法,利用如下特征进行检测.

- 短 gadgets-chain ROP 攻击特征:利用短 gadgets-chain 进行的 ROP 攻击中,攻击者需要利用短 gadgets-chain 来构造系统调用参数进行敏感系统调用,以达到攻击目的.
- ROP 攻击判定方法:在每次检测到 gadget 时,记录当时的系统状态;劫持敏感系统调用,在进行实际系统调用之前,检测系统调用的参数是否与上一次检测到 gadget 时保存的系统状态(以系统架构寄存器的值来表征系统状态)一致,如果一致(即敏感系统调用受到了攻击者控制),则认定当前遭受了 ROP 攻击.

2.5.4 检测算法小结

结合上述小结的检测算法,ROP 攻击检测的总体算法如图 2 所示.

last_lbr:保存上一分支 LBR 信息
last_arch_state:上一 gadget 的系统架构信息
cur_arch_state:当前的系统架构信息
cur_lbr:包含当前分支的 LBR 信息
syscall_state:包含系统调用所需参数架构信息
chain_length:保存当前的 gadgets-chain 长度,初始值为 0
cur_instr(*addr*):解析 *addr* 对应内存地址的指令类型
last_instr(*addr*):解析 *addr* 对应内存地址的上一指令类型
match(*state*1,*state*2):比较 *state*1 中的架构信息是否包含 *state*2 中的所需的架构信息
ROP_detect 算法如下:

```
//gadget 检测
potential_garget=0;
if (cur_lbr.from_ip|last_lbr.to_ip<maxGadgetLength)
   potential_garget=1;
if (Cur_instr(cur_lbr.from_ip)=="RET" && Last_instr(cur_lbr.to_ip))!="CALL")
   potential_garget=1;
//gadgets-chain 累计
if (potential_garget && chain_length>0 && cur_lbr.from_ip==last_lbr.from_ip &&cur_lbr.to_ip==last_lbr.to_ip)
   recursive_gadget=1;
chain_length=recursive_gadget? chain_length:
   potential_gadget? chain_length+1:0;
//ROP 攻击判定
if (chain_length>minChainLength|match(last_arch_state,syscall_state))
   signal ROP-attack (kill the detected application).
last_lbr=cur_lbr;
if (potential_garget)
   last_arch_state=cur_arch_state;
```

Fig.2 ROP detection algorithm

图 2 ROP 检测算法

3 实 现

本文实现的 MIBChecker 系统运行实验环境为:Intel i5-3320M CPU 2.6GHz,4G RAM;操作系统为 Ubuntu 15.04 x86_64(内核版本 3.19.0.93).系统结构如图 3 所示.

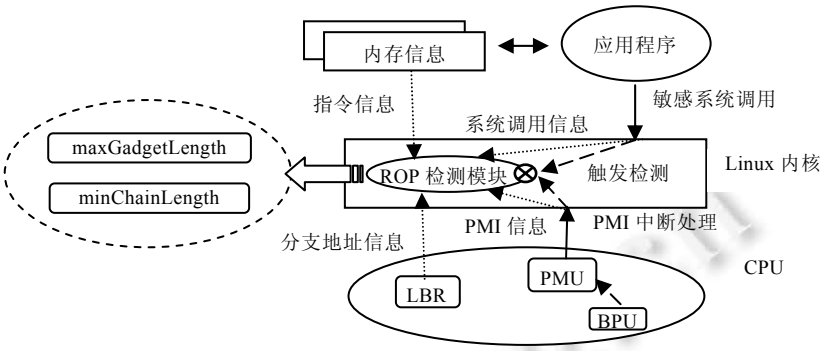


Fig.3 MIBChecker system achitecture

图 3 MIBChecker 系统结构

ROP 检测模块在 Linux 内核上实现,检测算法如上一节所述.在具体实现过程中,MIBChecker 检测算法还有两个问题需要确定.

1) 实际 CPU 平台中不一定支持我们理想的分支事件,那么实际的分支检测触发点如何选择?

2) gadget 检测以及 ROP 攻击判定相关的阈值如何设定?

3.1 分支检测触发点选择

本文在 Ivy Bridge 平台下实现了 MIBChecker 系统.然而,Ivy Bridge 的 PMU 功能并不支持我们理想的分支检测触发点,那么我们需要从该平台中寻找最接近理想分支事件的事件超集,即选择的事件必须包含所有理想的分支事件(这一原则适用于所有其他平台).

经过分析,Ivy Bridge 平台下满足条件的最接近理想分支事件的两个分支事件为:用户态预测失败的跳转分支指令提交事件(事件编码 0xc5,掩码为 0x20)和用户态预测失败的跳转间接分支指令执行事件(事件编码 0x89,掩码为 0xac).

针对这两种事件类型,我们利用 perf 性能采样工具对随机选取的 10 个 Linux 常用程序进行了采样,命令为

perf stat –e rac89:u,r20c5:*u* ⟨app⟩ ⟨options⟩

其中,perf stat 为性能采样命令;–e 用于指示待采样的事件;r⟨encode⟩用于表示事件编码和掩码,例如 rac89 表示事件编码为 0x89,掩码为 0xac;*u* 表示事件为用户态事件;app 为待采样的程序名;options 为程序运行所需携带的参数.重复 10 次得到的采样数据平均值见表 1.

Table 1 Sample data for two branch PMU events on common Linux applications

表 1 常用 Linux 应用上两种分支 PMU 事件的采样数据

程序	参数	用户态预测失败的跳转间接分支指令执行事件(rac89:*u*)	用户态预测失败的跳转分支指令提交事件(r20c5:*u*)
ls	–*a*	480	3 505
dmesg	–	42 984	61 317
ps	–*a*	14 278	21 202
cat	test	195	1 787
mkdir	newdir	456	4 048
ping	www.baidu.com –c 10	0	0
stat	test	970	6 229
passwd	–	0	0
gdb	a.out	273 562	710 585
tar	–cj test.tar test	1 084	7 346
vi	test	2 894	12 011

在这些程序上,用户态预测失败的跳转间接分支指令执行事件相比于用户态预测失败的跳转分支指令提交事件更少,因此本文选择用户态预测失败的跳转间接分支执行事件作为采样事件,并通过设置 PMU 计数器初始值为 0*xFFFFFFFFFFFF*,让其在每次发生该事件时计数器溢出,进而触发 PMI 中断进行 ROP 攻击检测.

3.2 阈值选择

与其他常规 ROP 攻击检测方法[11,12]一样,MIBChecker 的常规 ROP 攻击检测需要设置两个阈值:gadget 指令片段最长长度(*maxGadgetLength*)以及 ROP gadgets-chain 最短长度(*minChainLength*).

3.2.1 maxGadgetLength 阈值选择

maxGadgetLength 阈值用于识别 gadget:当指令片段长度低于该阈值时,认为该指令片段为 gadget.该阈值的选择需要满足以下条件.

1) 大于绝大部分理想的(副作用可控,具备一定攻击辅助功能[23]的)gadget 长度;

2) 应尽量小,以减少误报(降低将正常指令片段识别成 gadget 的可能).

我们通过 ROPgadget 工具从 Linux 常用软件及库(/bin,/usr/bin,/lib,/usr/lib)中抓取可能用于 ROP 攻击的 gadget,对它们进行了筛选,选出其中副作用可控,具备一定攻击辅助功能的 gadget.目前找到的满足条件的最长 gadget 如图 4 所示,长度为 27.为了保证对其他程序中的 gadget 也能进行有效的检测,本文将 *maxGadgetLength* 设为 30,略大于 27.

```
409f17:   66 0f 1f 84 00 00 00   nopw   0x0(%rax,%rax,1)
409f1e:   00 00
409f20:   48 83 c4 28            add    $0x28,%rsp
409f24:   48 89 d8               mov    %rbx,%rax
409f27:   5b                     pop    %rbx
409f28:   5d                     pop    %rbp
409f29:   41 5c                  pop    %r12
409f2b:   41 5d                  pop    %r13
409f2d:   41 5e                  pop    %r14
409f2f:   41 5f                  pop    %r15
409f31:   c3                     retq
```

Fig.4 Longest ideal gadget in testing

图 4 测试中最长的“理想”gadget

3.2.2 minChainLength 阈值选择

minChainLength 阈值用于判定 ROP 攻击,当累计到 gadgets-chain 长度大于该值时,认为程序遭受到了 ROP 攻击,因此该值的选择需要满足以下条件:1) 大于正常指令执行流中出现的最长合法 gadgets-chain 长度;2) 该阈值需要尽量小,以减少漏报(即将 ROP 攻击使用的 gadgets-chain 识别成了正常指令流).

本文针对表 1 中的 Linux 程序及参数进行了采样,识别的最大 gadgets-chain 长度见表 2.

Table 2 Longest gadgets-chain length on common Linux application

表 2 常用 Linux 应用中的最大 gadgets-chain 长度

程序	参数	最大 gadgets-chain 长度
ls	−*a*	3
dmesg	–	1
ps	−*a*	3
cat	test	1
mkdir	newdir	3
ping	www.baidu.com −c 10	2
stat	test	3
passwd	–	2
gdb	a.out	4
tar	−cj test.tar test	5
vi	test	4

这些程序中包含的最大 gadgets-chain 长度不超过 5.已有研究中,对于最大 gadgets-chain 的长度限定一般在 7~10[11,12].为了保证对其他程序的 gadgets-chain 也能够进行有效的检测,MIBChecker 将该阈值设置成 10.

4 评 估

本节将针对 MIBChecker 的安全性及性能进行评估.

4.1 安全性评估

本文构建的攻击测试集主要包括两个测试子集.

1) 常规 ROP 攻击测试子集:该测试子集主要用于评估 MIBChecker 在常规 ROP 攻击上的安全性,包括两种方式获取的攻击实例.
   a) 直接获取的已有攻击实例:本文从 exploit-db 攻击库中复现了针对现实漏洞程序的 7 个 ROP 攻击实例.
   b) 利用自动化攻击工具生成的攻击实例:本文基于 gadgets-chain 自动化生成工具 ropper 针对现实漏洞程序重新生成了 14 个攻击实例.
2) 短 gadgets-chain 攻击测试子集:该测试子集用于评估 MIBChecker 在短 gadgets-chain 攻击上的安全性,包括如下两种攻击实例.
   a) 基于人为构造的漏洞程序实现的攻击实例:本文基于人为构造的漏洞程序人工实现了两种 ROP 攻击变种——历史刷新攻击与规避攻击实例各 1 个.
   b) 现有标准测试集中的短 gadgets-chain 攻击实例:本文复用了 RIPE 缓冲区溢出测试集[24]中的 10 个 ROP 攻击实例.

4.1.1 常规 ROP 攻击

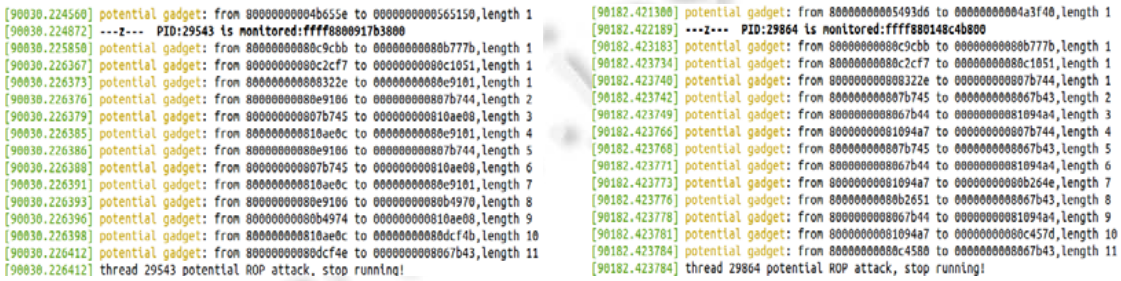
本文复现了 exploit-db 库中的 7 个现实 ROP 攻击实例,并基于 gadgets-chain 自动化生成工具 ropper 对这些现实中的漏洞程序重新生成了新的 ROP 攻击实例(每个漏洞程序生成了 2 个新的案例).测试结果见表 3.

Table 3 Detection result of MIBChecker on real-world ROP attack cases

表 3 MIBChecker 在现实 ROP 攻击实例上的检测结果

程序	EDB-ID	现实攻击实例检测结果	Ropper 生成攻击实例检测结果
SIPP 3.3	45 288	✓	✓
PMS 0.42	44 426	✓	✓
Crashmail 1.6	44 331	✓	✓
Bochs 2.6-5	43 979	✓	✓
MAWK 1.3.3-17	42 357	✓	✓
Flat Assembler 1.7.21	42 265	✓	✓
JAD 1.5.8e	42 255	✓	✓

无论是原始攻击实例还是基于 ropper 工具生成的攻击实例,它们使用的 gadget 都较短,而且 gadgets-chain 长度都大于 10.针对这些攻击实例,MIBChecker 能够检测出每个 gadgets-chain 从“length 1”到“length 11”整个 gadgets-chain 的跳转过程,并在 length 到达 11 时检测出 ROP 攻击,完整复现了这两个 ROP 攻击实例的整个攻击路径.以 jad overflow 为例,实验结果如图 5 所示.



(a) 原始攻击 (b) 基于 ropper 构建的攻击

Fig.5 ROP attack detection result on jad

图 5 针对 jad 的 ROP 攻击检测结果

实验结果表明,MIBChecker 能够有效地检测出现实中利用短 gadget(<30)和长 gadgets-chain(>10)进行的 ROP 攻击,并复现完整的攻击路径.

4.1.2 短 gadgets-chain 攻击

短 gadgets-chain 攻击测试集主要包括 3 种类型:历史刷新攻击以及规避攻击、RIPE 测试集[24].下面分别对它们进行评估测试,然后将检测结果与其他方法进行对比.

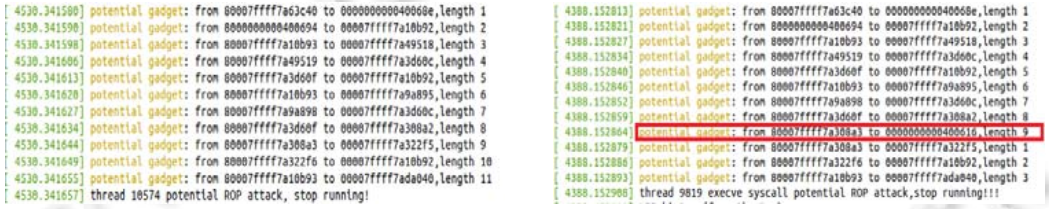
(一) 历史刷新攻击

历史刷新攻击[13]主要是通过在检测触发之前插入合法的程序流刷新 gadget 检测所需的 LBR 历史信息,进而达到掩盖 ROP 攻击的目的.已有研究中[11,12]采用窗口式 ROP 攻击检测,LBR 历史信息的产生到使用具有较长的时间间隔,容易遭受历史刷新攻击.MIBChecker 避免了这一点,在每个预测失败的间接分支处立即使用该分支的 LBR 信息进行 gadget 检测.

为了进一步验证 MIBChecker 能够有效地防范历史刷新攻击,本文人为地构造了一个简单溢出漏洞程序.

```
int victim(·){
   char name[64];
   gets(name);
   return 0;
}
```

利用该程序,我们首先进行了常规 ROP 攻击,然后对常规 gadgets-chain 进行了改造,添加了一个 gadget,该 gadget 会构造参数进行一个正常递归调用 20 次用于刷新 LBR 信息.插入前后 MIBChecker 的检测结果如图 6 所示.



(a) 常规 ROP 攻击 (b) 历史刷新攻击

Fig.6 Gadget detection result on history flush attack

图 6 针对历史刷新攻击的 gadget 检测结果

由图 6 可以看出,在历史刷新攻击存在的情况下,MIBChecker 的检测链确实被插入的 gadget(见图 6 中方框)打断了,但不影响后续 gadget 的检测,该攻击依旧会被 MIBChecker 检测出来.实验结果表明,MIBChecker 能够有效地检测出历史刷新攻击.

(二) 规避攻击

规避攻击[13]由 Carlini 等人提出,原指仅使用“合法”的 gadget 进行的 ROP 攻击.本文将利用“合法”gadgets-chain(其中可以包含不合法 gadget)进行的 ROP 攻击也归入该范畴.已有方法如 kBouncer[11]、ROPecker[12]均不能有效地检测规避攻击.规避攻击可能的情况如下.

1) 全部由“合法”的 gadget(对于 MIBChecker,是指长度大于 30 且满足 CALL-RET 特性的指令片段)构成的攻击.
2) 仅由一个“合法”gadgets-chain(对于 MIBChecker,是指长度小于 10 的 gadgets-chain)构成的攻击.
3) 由一系列“合法”gadgets-chain,中间插入“合法”gadget 构成的攻击(上一节的例子属于该类).

对于第 1 种情形,长度大于 30 的长指令片段副作用较多,攻击者难以有效控制其达到攻击目的,可行性不高.对于第 3 种情形,其本质上和第 2 种情形是一致的,即最终都是通过短 gadgets-chain 进行敏感系统调用完成攻击.为了进一步验证 MIBChecker 是否能够检测这种短 gadgets-chain 构成的规避攻击.本文构造了一个包含 5

个 gadget 的 ROP 攻击实例,检测结果如图 7 所示.

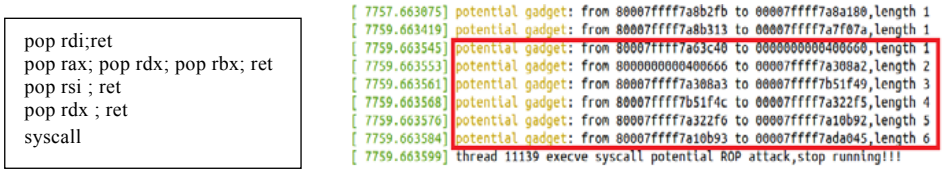


Fig.7 ROP attack detection result on short gadgets-chain attack

图 7 短 Gadgets-chain 攻击的 ROP 攻击检测结果

实验结果表明,MIBChecker 能够有效地检测出通过较短的 gadgets-chain 进行敏感系统调用的规避攻击.

(三) RIPE 测试集

RIPE 测试集[24]是一个专门用于包含评估缓冲区溢出漏洞防御技术安全性的测试集,它总共包含 850 个不同的攻击实例,其中 10 个为 ROP 攻击实例.这些 ROP 攻击实例基于不同的函数(如 memcpy、strcpy、sscanf 等)来实现缓冲区溢出.MIBChecker 在这些攻击实例上的检测结果见表 4.

Table 4 ROP attack detection result on RIPE test suite

表 4 针对 RIPE 测试集的 ROP 攻击检测结果

缓冲区溢出函数	检测结果
memcpy	✓
strcpy	✓
strncpy	✓
sprintf	✓
strcat	✓
strncat	✓
sscanf	✓
fscanf	✓
homebrew	✓

这些攻击案例所利用的 ROP 攻击代码通过相同的短 gadgets-chain 来实现,MIBChecker 不关注于缓冲区溢出的具体函数,直接针对该短 gadgets-chain 进行检测(由于版本兼容问题,原 gadgets-chain 无法成功进行攻击,本文在不改变 gadgets-chain 长度的前提下,对原 gadgets-chain 进行了简单改造).改造后的 gadgets-chain 构成与检测结果如图 8 所示.

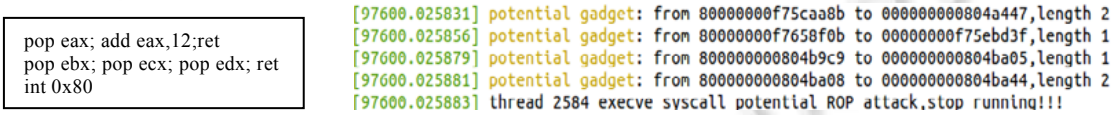


Fig.8 Gadgets-chain construction and detection result on RIPE test suite

图 8 RIPE 测试集中的 gadgets-chain 构成与检测结果

实验结果表明,MIBChecker 能够有效地检测出 RIPE 测试集中的短 gadgets-chian 攻击.

(四) 与其他防御方法的对比

本文将这些短 gadgets-chain 攻击在仅开启常规 ROP 检测功能的 MIBChecker 中进行了测试,并针对其他防御方法如 kBouncer 和 ROPecker 进行了评估,其对比结果见表 5.

Table 5 Comparison of detection result on different defense approaches

表 5 不同防御方法检测结果对比

攻击实例	kBouncer/ROPecker	MIBChecker(仅常规 ROP 检测)	MIBChecker
历史刷新攻击	×	×	✓
规避攻击	×	×	✓
RIPE ROP 攻击	×	×	✓

这些攻击实例在检测点“可见”的 gadgets-chain 长度都小于 kBouncer、ROPecker 以及 MIBChecker 常规 ROP 检测的检测阈值,因此会绕过这些防御机制的检测.对比结果表明,相对于这些仅依靠长 gadgets-chain 特征判定 ROP 攻击的检测方法,引入系统调用检测方法的 MIBChecker 在短 gadgets-chain 攻击防御方面具备较为明显的优势.

4.2 性能评估

我们采用 SPEC2006 作为 Benchmark 对 MIBChecker 的性能开销进行了评估.SPEC2006 测试集包含两个子测试集:SPECint2006 整型测试集和 SPECfp2006 浮点测试集.SPECfp2006 浮点测试集由于版本兼容问题,部分程序编译失败,而且与整型测试集相比,浮点测试集中分支指令比重较少,受 MIBChecker 影响较小,因此该评估主要集中在 SPECint2006 测试集上,仅对当前编译通过的部分浮点测试集程序进行评估.

首先,针对 SPEC2006 的间接分支事件进行了采样,见表 6.

Table 6 Branch related PMU event counter on SPEC2006

表 6 SPEC2006 上分支相关 PMU 事件计数

程序	预测失败的间接分支执行事件数	执行的间接分支数	所需检测的分支减少比例(%)
400.perlbench	2 130 356 154	47 210 407 260	95.49
401.bzip	296 416 065	8 112 479 919	96.35
403.gcc	459 151 686	11 809 447 072	96.11
429.mcf	1 254 075	8 556 327	85.34
445.gobmk	33 486 205	808 133 481	95.85
456.hmmer	1 263 206	8 658 138	85.41
458.sjeng	3 458 946 719	34 930 617 758	90.10
462.libquantum	695 361	5 009 294	86.11
464.h264ref	3 860 320	26 420 321	85.38
471.omnetpp	3 884 031	28 671 677	86.45
473.astar	1 278 374	8 642 528	85.20
483.xalancbmk	319 495 152	18 514 692 642	98.27
433.milc	653 106	4 457 463	85.34
444.namd	1 185 799	8 169 297	85.48
450.soplex	2 529 271	17 239 736	85.32
470.lbm	665 000	4 557 532	85.40
482.sphinx3	10 352 577	75 785 775	86.33

从该表可看出,MIBChecker 利用硬件 BPU 的特性,能够将所需检测的间接分支平均减少 89.06%(最多减少了 98%,最少减少了 85.27%).

为了进一步确认 MIBChecker 真实引入的性能开销,我们对开启和未开启 MIBChecker 情况下的 SPEC2006 程序分别进行了性能采样(每个程序运行 10 遍取平均值),性能评估结果对比如图 9 所示.

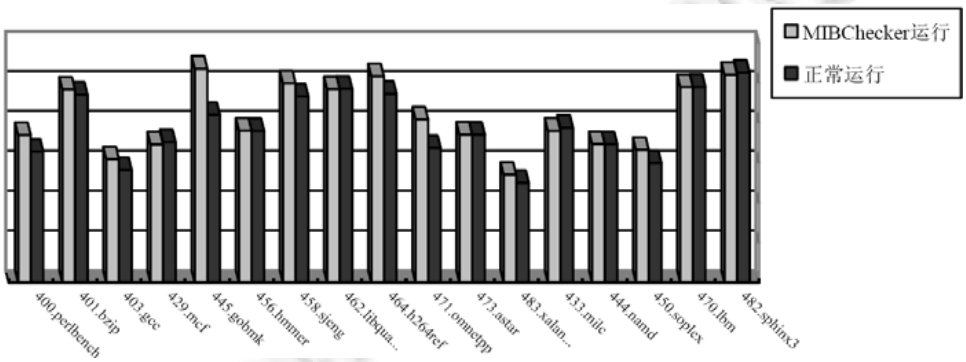


Fig.9 Performance overhead on SPEC2006

图 9 SPEC2006 性能开销

MIBChecker 平均引入 5.7%的性能开销,对于大部分 SPEC2006 程序性能开销均在 3%以内,只有少数程序

性能开销超过了 10%.实验结果表明,MIBChecker 引入的性能开销在可接受范围之内.

5 结 论

ROP 攻击是当今软件安全领域面临的主要安全威胁之一.本文针对 ROP 攻击展开研究,提出了一种基于硬件分支信息的 ROP 攻击检测方法——MIBChecker.该方法具备如下优点.

1) 安全:首次利用硬件 PMI 机制实时地针对每个可能用于 ROP 攻击的间接分支进行 ROP 攻击检测,规避了历史刷新攻击的可能,同时提出了敏感系统调用参数检测方法,能够有效地检测出短 gadgets-chain ROP 攻击.
2) 透明:检测直接由 PMU 事件触发机制以及敏感系统劫持来触发,而且检测所需信息完全从运行时通过 LBR 等提取,不依赖于程序源码或调试信息,整个检测过程对用户程序透明.
3) 性能开销低:结合 BPU、PMU 等硬件机制以及 ROP 攻击特性,对所需检测的间接分支进行了大规模的筛减(减少约 89%的间接分支),大大降低了检测频度,进而降低了攻击检测本身引入的性能开销;同时,高效的硬件触发机制也降低了触发攻击检测所引入的性能开销.

实验结果表明,该方法能够不受历史刷新攻击影响进行 gadget 检测,能够有效地检测出常规 ROP 攻击和规避攻击,并仅引入 5.7%的性能开销.

References:

[1] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: Proc. of the 14th ACM Conf. on Computer and Communications Security (CCS 2007). Alexandria: ACM, 2007. 552−561. [doi: 10.1145/1315245.1315313]

[2] Bletsch T, Jiang X, Freeh VW, Liang ZK. Jump-oriented programming: A new class of code-reuse attack. In: Proc. of the 6th ACM Symp. on Information, Computer and Communications Security (ASIACCS 2011). Hong Kong: ACM, 2011. 30−40. [doi: 10.1145/1966913.1966919]

[3] Schuster F, Tendyck T, Liebchen C, Davi L, Sadeghi AR, Holz T. Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications. In: Proc. of the 2015 IEEE Symp. on Security and Privacy (SP). San Jose: IEEE, 2015. 745−62. [doi: 10.1109/SP.2015.51]

[4] Veen VVD, Andriesse D, Stamatogiannakis M, Chen X, Bos H, Giuffrida C. The dynamics of innocent flesh on the bone: Code reuse ten years later. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2017). Dallas: ACM, 2017. 1675−1689.

[5] Abadi M, Budiu M, Erlingsson Ú, Ligatti J. Control-Flow integrity. In: Proc. of the 12th ACM Conf. on Computer and Communications Security (CCS 2005). Alexandria: ACM, 2005. 340−353. [doi: 10.1145/1102120.1102165]

[6] Abadi M, Budiu M, Erlingsson Ú, Ligatti J. Control-flow integrity principles, implementations, and applications. ACM Trans. on Information and System Security, 2009,13(1):4:1−4:40. [doi: 10.1145/1609956.1609960]

[7] Zhang C, Wei T, Chen ZF, Duan L, Szekeres L, McCamant S, Song D, Zou W. Practical control flow integrity and randomization for binary executables. In: Proc. of the 2013 IEEE Symp. on Security and Privacy (SP). Berkeley: IEEE, 2013. 559−573. [doi: 10.1109/SP.2013.44]

[8] Zhang M, Sekar R. Control flow integrity for COTS binaries. In: Proc. of the 22nd USENIX Security Symp. (USENIX Security 2013). Washington: USENIX Association, 2013. 337−352.

[9] Mashtizadeh AJ, Bittau A, Boneh D. Cryptographically enforced control flow integrity. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security (CCS 2015). Denver: ACM, 2015. 941−951. [doi: 10.1145/2810103.2813676]

[10] Veen VVD, Andriesse D, Göktas E, Gras B, Sambuc L, Slowinska A, Bos H, Giuffrida C. Practical context-sensitive CFI. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security (CCS 2015). Denver: ACM, 2015. 927−940. [doi: 10.1145/2810103.2813673]

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

[12] Cheng YQ, Zhou ZW, Yu M, Ding XH, Deng RH. ROPecker: A generic and practical approach for defending against ROP attacks. In: Proc. of the 2014 Network and Distributed System Security Symp. (NDSS 2014). San Diego: 2014. [doi: 10.14722/ndss.2014.23156]

[13] Carlini N, Wagner D. ROP is still dangerous: Breaking modern defenses. In: Proc. of the 23rd USENIX Security Symp. (USENIX Security 2014). San Diego: USENIX Association, 2014. 385−399.

[14] Xia Y, Liu Y, Chen H, Zang B. CFIMon: Detecting violation of control flow integrity using performance counters. In: Proc. of the IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN 2012). Boston: IEEE, 2012. 1−12. [doi: 10.1109/DSN.2012.6263958]

[15] Christoulakis N, Christou G, Athanasopoulos E, Christoulakis N, Ioannidis S. HCFI: Hardware-enforced control-flow integrity. In: Proc. of the 6th ACM on Conf. on Data and Application Security and Privacy (CODASPY 2016). New Orleans: ACM, 2016. 38−49. [doi: 10.1145/2857705.2857722]

[16] Das S, Zhang W, Liu Y. A fine-grained control flow integrity approach against runtime memory attacks for embedded systems. IEEE Trans. on Very Large Scale Integration Systems, 2016,24(11):3193−3207. [doi: 10.1109/TVLSI.2016.2548561]

[17] He W, Das S, Zhang W, Liu Y. No-jump-into-basic-block: Enforce basic block CFI on the fly for real-world binaries. In: Proc. of the 54th Annual Design Automation Conf. (DAC). Austin: ACM, 2017. 23:1−23:6. [doi: 10.1145/3061639.3062291]

[18] Davi LV, Dmitrienko A, Nürnberger S, Sadeghi AR. Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and ARM. In: Proc. of the 8th ACM Symp. on Information, Computer and Communications Security (ASIACCS 2013). Hangzhou: ACM, 2013. 299−310. [doi: 10.1145/2484313.2484351]

[19] Koo H, Polychronakis M. Juggling the gadgets: Binary-level code randomization using instruction displacement. In: Proc. of the 11th ACM Symp. on Information, Computer and Communications Security (ASIACCS 2016). Xi'an: ACM, 2016. 23−34. [doi: 10.1145/2897845.2897863]

[20] Pappas V, Polychronakis M, Keromytis AD. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization. In: Proc. of the 2012 IEEE Symp. on Security and Privacy (SP). San Francisco: IEEE, 2012. 601−615. [doi: 10.1109/SP.2012.41]

[21] Chen X, Bos H, Giuffrida C. CodeArmor: Virtualizing the code space to counter disclosure attacks. In: Proc. of the 2017 IEEE European Symp. on Security and Privacy (EuroS&P 2017). Paris: IEEE, 2017. 514−529. [doi: 10.1109/EuroSP.2017.17]

[22] Sinha K, Kemerlis VP, Sethumadhavan S. Reviving instruction set randomization. In: Proc. of the IEEE Int'l Symp. on Hardware Oriented Security and Trust. McLean: IEEE, 2017. 21−28. [doi: 10.1109/HST.2017.7951732]

[23] Schwartz EJ, Avgerinos T, Brumley D. Q: Exploit hardening made easy. In: Proc. of the 20th USENIX Conf. on Security. San Francisco: USENIX Association, 2011. 25−25.

[24] Wilander J, Nikiforakis N, Younan Y, *et al*. RIPE: Runtime intrusion prevention evaluator. In: Proc. of the 27th Annual Computer Security Applications Conf. (ACSAC). Orlando: ACM, 2011. 41−50. [doi: 10.1145/2076732.2076739]

李威威(1990－),男,博士生,主要研究领域为软件安全.

高伟毅(1992－),男,助理研究员,主要研究领域为图像识别,目标检测.

马越(1984－),男,博士,高级工程师,主要研究领域为形式化方法,系统安全.

杨秋松(1977－),男,博士,研究员,博士生导师,CCF 专业会员,主要研究领域为软件工程,形式化方法,系统安全.

王俊杰(1987－),女,博士,副研究员,主要研究领域为智能软件工程.

李明树(1966－),男,博士,研究员,博士生导师,CCF 会士,主要研究领域为操作系统深度设计(包括安全操作系统、数据操作系统等),可信软件过程以及基础软硬件核心技术与应用.