# ROPDetector：一种基于硬件性能计数器的 ROP 攻击实时检测方法

牛伟纳[1] 赵成洋[1] 张小松[1),2)] 黄晓祥[1] 蒋 廉[1] 张钶旋[1]

[1)](电子科技大学计算机科学与工程学院网络空间安全研究院 成都 611731)

[2)](鹏程实验室网络空间安全研究中心 广东 深圳 518000)

**摘 要** 面向返回编程(Return-Oriented Programming,ROP)是针对软件漏洞利用最广泛的攻击技术之一,能够绕过数据执行保护、地址空间布局随机化等防御机制.本文提出了一种基于硬件的 ROP 攻击实时检测方法,在不需要任何边缘信息(如源代码、编译器支持)和二进制重写的情况下,利用现代 CPU 中的硬件性能计数器监控目标程序执行过程,提取 ROP 攻击发生时底层硬件事件特征来实时检测 ROP 攻击.然后,在 32 位 Linux 实验环境下实现了原型系统 ROPDetector,使用真实的 ROP 攻击与漏洞进行实验,并与同类方法进行了对比实验,最后评估了系统的性能消耗.实验结果表明,该方法能有效地检测真实的 ROP 攻击,在分别以 6 次和 9 次错误预测返回指令为检测周期时,系统性能消耗仅有 5.05%和 5.25%,磁盘 I/O 性能消耗仅有 0.94%和 2%,网络 I/O 性能消耗仅有 0.06%和 0.78%.

**关键词** 面向返回编程;硬件事件;实时检测;硬件性能计数器;错误预测返回指令

**中图法分类号** TP309 **DOI 号** 10.11897/SP.J.1016.2021.00761

# ROPDetector: A Real-time Detection Method of ROP Attack Based on Hardware Performance Counter

NIU Wei-Na[1)] ZHAO Cheng-Yang[1)] ZHANG Xiao-Song[1),2)] HUANG Xiao-Xiang[1)]
JIANG Lian[1)] ZHANG Ke-Xuan[1)]

[1)](*School of Computer Science and Engineering, Institute for Cyber Security, University of Electronic Science and Technology of China, Chengdu* 611731)

[2)](*Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, Guangdong* 518040)

**Abstract** Return-oriented programming (ROP) is one of the most common attack techniques for software vulnerabilities. Attackers can use this attack technique to tamper with the program execution flow to run malicious code. ROP attack is a generalization of the ret-to-lib attack. In the ret-to-lib attack, the attacker reuses the entire libc library. While in the ROP attack, the attacker chains available gadgets to form a spiteful attack chain to carry out the attack, which can bypass the data execution protection, address space layout randomization defense mechanism, and ultimately destroy the user-level and kernel-level software modules. Among them, gadget refers to the instruction fragment ending with the return instruction in the target program or the library function called by the target program. A series of special registers for recording the number of hardware-related activities built into the CPU of modern computers are called hardware performance

收稿日期:2019-11-10;在线发布日期:2020-05-11.本课题得到国家重点研发计划(2016QY13Z2302)、国家自然科学基金(61902262,U19A2066)、四川省科技支撑项目(2017CC0071)资助.**牛伟纳**,博士,讲师,主要研究方向为网络攻击检测、恶意代码分析.E-mail:niuweina1@126.com.**赵成洋**,硕士研究生,主要研究方向为网络攻击检测.**张小松**(通信作者),博士,教授,主要研究领域为网络攻击检测、恶意代码分析.E-mail:johonsonzxs@uestc.edu.cn.**黄晓祥**,硕士研究生,主要研究方向为系统安全.**蒋 廉**,硕士研究生,主要研究方向为网络攻击检测.**张钶旋**,硕士研究生,主要研究方向为二进制安全.

counters (HPCs), which is designed for performance debugging of complex software systems. Developers can configure the HPC to monitor the execution of the target program. As with traditional software analysis tools, they can collect detailed performance information about the target program. It is helpful to understand program behavior at runtime and optimize their performance. In response to the threat of ROP attacks, a large number of researchers at home and abroad have invested in the research of ROP defense mechanisms, and have proposed a series of defense or detection methods. However, some existing defense techniques can be overcome by attackers, or suffer from high performance overhead. In this paper, we observed anomalies in exceptional underlying hardware events when the ROP attack was happening. Due to it, we proposed a real-time detection approach based on hardware without any side information (e.g., source code, compiler support), which used hardware performance counters in modern CPU to monitor target program execution process and extracted the low-level feature of hardware events against ROP attack in runtime. Then, we implemented a prototype on x86-based Linux platform called ROPDetector. We found a tiny program with buffer overflow vulnerability, and used ROPGadget to construct 40 real ROP payloads. We reproduced other method which is also based on hardware performance counters for comparative experiments and utilized real exploits which are found on the open source exploitation platform to verify feasibility of ROPDetector. The experimental results showed that our approach could efficiently detect real ROP attack with a higher detection rate. At last, we evaluated the performance overhead of our method through UnixBench, the disk I/O performance overhead of our method through Bonnie++, and the network I/O performance overhead of our method through the Apache web server httpd-2.4.41. When the detection interval $T$ of ROPDetector is 6 mis-predicted return instruction events, the overall system performance drops by 5.05%, the disk I/O performance drops by 0.94%, and the network I/O performance drops by 0.06%. When the detection interval $T$ of ROPDetector is 9 mis-predicted return instruction events, the overall system performance drops by 5.05%, the disk I/O performance drops by 0.94%, and the network I/O performance drops by 0.06%.

**Keywords** return-oriented programming; hardware events; real-time detection; hardware performance counter; mis-predicted return instructions

## 1 引 言

面向返回编程是一种高级代码复用攻击技术[1].攻击者可以利用这种攻击技术去篡改程序执行流来执行恶意代码.在 ROP 攻击中,攻击者将可用的 gadget 链接起来,构成恶意攻击链来进行攻击,能够绕过数据执行保护、地址空间布局随机化(Address Space Layout Randomization, ALSR)[2]防御机制,最终达到用户级[3]和内核级软件模块的破坏[4].其中,gadget 是指在目标程序或目标程序所调用的库函数中以返回指令为结尾的指令片段.近年来,如表 1 所示,大量与 ROP 攻击相关的漏洞被曝出来,例如 glibc(CVE-2015-7547)①、pdfium (CVE-2015-6,787)②、Wireshark(CVE-2015-87 36)③等.针对 ROP 攻击威胁,国内外大量研究人员投入到 ROP 防御机制研究中,并提出了一系列防御/检测方法.

G-free[5]是第一个实用的且可以防御所有形式 ROP 攻击的方法,它集成了对齐 sled、加密返回地址、帧 cookie 和代码重写技术.前三个技术都是在

① Google Security Research. Glibc-getaddrinfo Stack-Based Buffer Overflow. http://www.exploit-db.com/exploits/39454/

② Google Security Research. Pdfium CPDF_Function::Call Stack-Based Buffer Overflow. https://www.exploit-db.com/exploits/39165/

③ Google Security Research. Wireshark-file_read(wtap_read_byte_or_eof/mp2t_find_next_pcr) Stack-based Buffer Overflow. https://www.exploit-db.com/exploits/38997/

**表 1　ROP 相关 CVE**

| 序号 | 描述 | CVEs |
|---|---|---|
| 1 | Adobe Flash Casi32 Int. Ovf. | 2014-0569 |
| 2 | Adobe Flash Domain Memory UAF | 2015-0359 |
| 3 | IE COALineDashStyleArray Int. Ovf. | 2013-2551 |
| 4 | IE CButton UAF | 2012-4792 |
| 5 | IE Fixed Table Col Span Heap Ovf. | 2012-1876 |
| 6 | IE Execommand UAF | 2012-4969 |
| 7 | IE CDisplayPointer UAF | 2013-3897 |
| 8 | Wireshark | 2015-8736 |
| 9 | IE Option Element UAF | 2011-1996 |
| 10 | Adobe Flash Kern Parsing Int. Ovf. | 2012-1535 |
| 11 | pdfium | 2015-6787 |
| 12 | Nginx1.3.9 | 2013-2028 |
| 13 | Glibc2.2.0 | 2015-7547 |
| 14 | DNSTracer1.8 | 2016-10190 |
| 15 | FFmpeg3.2.1 | 2017-9430 |

保护二进制文件中可用的对齐的自由分支指令(对齐的概念是指它只使用代码中已经存在的“预期”指令),利用现有的堆栈保护机制,把具有危险指令的函数作为一个短代码块整体,确保该函数内任何指令不能单独执行,只能以一个整体执行,保证攻击者不能直接跳转到该危险指令执行,这样才能执行自由分支指令.此外,G-free 是针对各种可能出现分支指令未对齐的情况,通过代码重写或者一些其它手段来阻止这种情况发生.它有效消除了传统的利用 lib 库进行攻击的威胁,具有无需在目标进程中注入任何代码,性能开销非常低,且增加的文件大小也在接受范围内的优点.缺点是它是基于编译器进行定制的,需要提前收集编译器信息,实际应用不太现实.CFLocking[6]通过对目标程序进行重编译来限制或是锁定异常控制流的数量来检测 ROP 攻击,但是这种技术本质上就不能检测以未对齐分支指令为结尾的 ROP 攻击.Return-less kernel[7]是一种基于编译器的 ROP 攻击检测方法,它通过从内核镜像中移除 ret 相关操作,将控制流数据从栈移动到专用区域.基于此方法,Shuo 等人[8]提出了一种基于虚拟化的 ROP 攻击检测方法,但此方法需要被检测程序的源代码,然后在编译过程中在函数的头部和尾部插入控制流完整性检测.显然,以上方法都不能防御使用 jmp 或是 call 指令的 ROP 攻击.CFI-Mon[9]是通过分析运行时控制流轨迹来检测那些企图篡改程序控制流的攻击,但它具有较高的延时性,可能导致检测到攻击时攻击早就已经发生过了.MoCFI[10]是在智能手机上检测篡改控制流攻击的框架,它能够实时地进行控制流完整性检查,且不需要目标程序的源代码,实验结果表明它在对 IOS 上流行的应用程序进行监测时的开销并不高.Polychronakis 等[11]提出一种对任意数据都可以检测出 ROP payloads 的方法,该技术通过扫描目标程序地址空间中可执行代码的输入来检测 ROP payloads.

TRUSS[12]和 TaintCheck[13]使用二进制代码插桩的方式来检测 ROP 攻击,这类方法不仅会打破目标程序的二进制完整性,也会带来高昂的性能代价.例如,TaintCheck 的性能消耗更是高达 20 倍.为了克服性能消耗过高这一点,IPR[14]被提出来.IPR 目的是破坏 ROP gadget 而不改变代码规模.但是它总是只破坏可执行地址空间的明显的 gadget,很难保证剩余的 gadget 能否构成一次完整的 ROP 攻击.ROPDefender[15]通过跟踪目标程序二进制指令的执行过程来检查软件控制流完整性进而检测是否受到 ROP 攻击.一旦目标程序发出调用指令,Defender 会将返回地址的副本存储到一个单独的影子堆栈中,在运行时检查目标程序执行过程中执行的返回指令,将其和影子堆栈上的顶部返回地址进行对比,以此来判断是不是恶意构造的 ROP payloads. ROPDefender 能够检测 ROP 攻击,甚至可以检测到非预期的指令序列.缺点是系统会检查进程执行的每条机器指令,开销巨大,而且由于它的影子堆栈机制,只关注了以返回指令结尾的 ROP gadget 的防御,比较局限.CCFIR[16]将所有的间接控制流分支指令的合法目标地址存储在一个基础数据库里,然后插桩二进制代码,规定所有的间接分支指令的目标地址只能是基础数据库里的地址.但是,一方面该方法可移植性差,因为它事先收集的地址数据库只能是某一个固定平台(例如:Windows 10)上的共享代码库;另一方面,攻击者只需要使用非基础数据库里面的 gadget 就能绕过此检测机制.

为了有效扩展现有 ROP 攻击检测范围,基于硬件的方法 KBouncer[17]和 ROPecker[18]被提出来,它们都使用最后分支记录(Last Branch Record, LBR)来监控程序的执行并追踪间接分支的目标地址,将 LBR 中的地址和程序的控制流路径相比较来判断程序运行期间是否受到 ROP 攻击.这两种工具都具有良好的攻击类型覆盖和运行效率,相对于 G-free 来说,不太依赖编译器环境等多余信息.对于 ROPDefender 来说,检测 ROP 攻击的类型更广.其中,KBouncer 利用二进制重写技术编写代码,破坏了二级制代码完整性,引发了针对安全机制的兼容性问题,而且它只监视选定关键路径上的执行流,会错误遗漏其余路径上的 ROP 攻击.ROPecker 是一种基于 Linux 的反 ROP 方法,它使用 LBR CPU

寄存器来查找潜在的 gadget 执行,另外需要一个静态数据库检测将来可能调用的小工具.每当程序执行离开滑动窗口或调用关键的系统指令时,都会触发检查.但是这个滑动窗口机制可能会被攻击者利用来绕过.ROPK++[19]在 ROPecker 的基础上进行扩展,在其内核模块中加入了 ret-check()函数,加入了"ret"指令完整性检测机制,通过"call"与"ret"的一一对应性来加强 ROP 攻击防御能力.缺点是与 ROPecker 同样存在粗粒度被人为绕过的问题以及预先的静态数据库建立耗费大量的时间成本的问题.

基于硬件性能计数器(Hardware Performance Counter,HPC)的 ROP 攻击检测方法 HDROP[20]能够极大地减少性能开销,与前面的方法不同,它不再关注内存和 LBR 中的信息,从而省去了一个时间成本很大的环节.现代处理器利用分支预测器来提高性能,ROP 攻击常常使它无法预测正确的分支目标,DROP[21]通过检测给定执行路径上的错误预测是否异常地增加,来判断是否存在 ROP 攻击.SIGDROP[22]利用在一个检测周期内(6 个错误预测返回指令)gadget 长度与错误预测返回指令数的乘积小于等于总指令数作为检测逻辑来判断是否发生 ROP 攻击,但这种方法只能针对 ROP 攻击且攻击者可以利用 long gadget 来绕过这种检测机制.ROPSentry[23]通过使用硬件性能计数器跟踪特定的硬件性能事件来观察、分析 ROP 攻击行为模式,检测的精度较高,在防御基于堆喷射的 ROP 攻击时性能消耗较低,但在防御非基于堆喷射的 ROP 攻击时性能消耗也在 10%以上.

本文提出了基于配置硬件性能计数器获取异常特征来检测 ROP 攻击的方法——ROPDetector,它可以在无源代码、无定制编译器支持和无二进制重写的情况下高效、低消耗地防御 ROP 攻击.同时,它适用于任何系统框架中,ROPDetector 是一种通用、实用和低消耗的 ROP 攻击检测机制.

本文的主要贡献可概括如下:

(1)设计了一种通用且实用的 ROP 攻击检测方法,ROPDetector 不需要源代码、定制编译器支持和二进制重写.

(2)ROPDetector 适用于任何平台,不依赖于特定的软件,不需要修改硬件.

(3)我们在基于 X86 的 Linux 平台上实现了 ROPDetector,验证了它的正确性,并评估了它的系统性能消耗、磁盘 I/O 性能消耗和网络 I/O 性能消耗这些指标.

## 2 背景知识

### 2.1 返回地址堆栈(Return Address Stack)

返回地址堆栈(RAS)[24]是一个存储着 return 指令目标地址的先进后出的硬件栈.由于 call 指令与 ret 指令是一一对应的,当有一个 call 指令发生时,处理器会将下一条指令(即返回目标地址)压入 RAS 的栈顶.当 ret 指令发生时,处理器会弹出 RAS 栈顶的预测地址,若与进程栈里的返回目标地址不同,则为预测失败.

### 2.2 面向返回编程(Return-Oriented Programming)

ROP 攻击是 ret-to-lib 攻击的推广,在 ret-to-lib 攻击中,攻击者复用整个 libc 库.在 ROP 攻击中,攻击者在内存(目标程序或者是共享库)中找到可用的代码片段——gadget,而后将构造一个有效载荷来将一系列 gadget 链接在一起.如图 1 所示,攻击者利用 return 指令将可利用库文件中的 gadget 链接起来,就可以执行想要的任何操作.

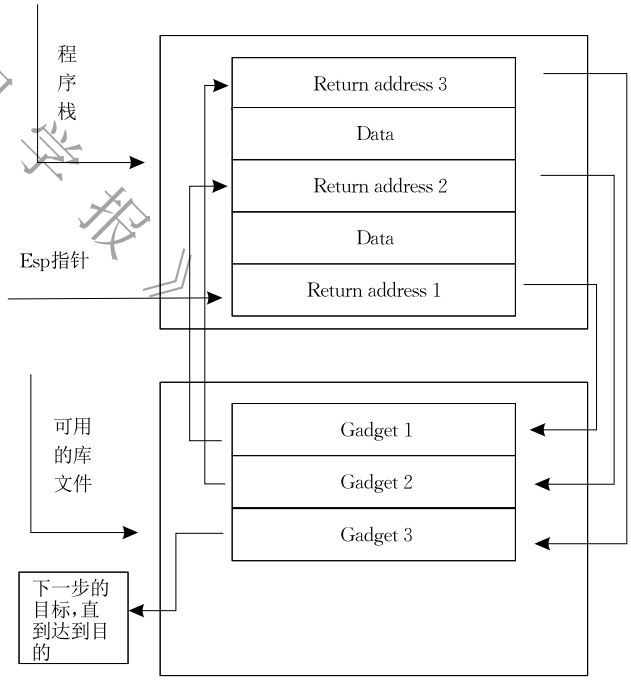


图 1 ROP gadget 链进行链接与攻击示意图

### 2.3 硬件性能计数器(Hardware Performance Counter)

现代计算机 CPU 中内置了一系列用于记录硬件相关活动发生次数的专用寄存器,被称为硬件性能计数器(Hardware Performance Counter,HPC)[25].HPC 与硬件事件选择器一起,当指定的硬件事件发生时,计数器将递增.

HPC 最初被设计用于复杂软件系统的性能调

试，利用 HPC，开发者可以更深入地理解运行时的程序行为和优化其性能[25]. 开发者可以通过配置 HPC 监控目标程序执行与传统的软件分析工具一样能收集到目标程序的详细性能信息，但系统开销却低得多并且不需要修改源代码.

早期的处理器仅支持少部分事件并且 HPC 的数量也很少. 在现代 CPU 中，预定义的可被监控的性能事件根据 CPU 微架构的不同也不同. 例如，Intel PentiumⅢ有两个 HPCs 和一百多种不同的事件；AMD Opteron 有四个 HPCs 和超过一百种事件；相比之下，第六、七代 Intel Core 处理器[26]有数百种事件，如表 2 所示的架构兼容事件和部分架构非兼容事件.

**表 2　第六、七代 Intel Core CPU 中部分性能事件**

| 架构兼容性能事件 | 描述 | 非架构兼容体系事件(部分) | 描述 |
|---|---|---|---|
| UnHalted Core Cycles | 时钟周期数 | UOPS_EXECUTED.THREAD | 在所有端口上执行的微指令数目 |
| UnHalted Reference Cycles | 访问时钟周期数 | BR_INST_RETIRED.CONDITIONAL | 条件分支指令数 |
| Instructions Retired | 有效执行的指令数 | DTLB_MISSES | DTLB 未命中 |
| LLC Reference | LLC 的访问次数(包括命中和未命中) | BR_INST_RETIRED.NEAR_RETURN | 返回分支指令数 |
| LLC Misses | LLC 未命中数 | ITLB_MISSES | ITLB 未命中 |
| Branch Instruction Retired | 有效执行的分支指令数 | BR_MISP_RETIRED.CONDITIONAL | 条件分支指令预测失败 |
| Branch Misses Retired | 分支未命中数 | MEM_INST_RETIRED.ALL_LOADS | 所有的载入指令数 |

## 3　系统设计

### 3.1　威胁模型

在我们的威胁模型中，我们允许攻击者利用目标程序软件漏洞，如缓冲区溢出等来获取程序栈的控制和发起 ROP 攻击. 因此，我们假定攻击者知道目标程序的全部细节，并能向目标程序发送任意的输入.

### 3.2　条件假设

我们假定 CPU 和操作系统开启了数据执行保护机制(Data Execution Protection，DEP)，编译程序过程中关闭栈保护机制. 同时，攻击者有权限执行目标程序的二进制文件及其共享库文件来获得可用的 gadget 链，利用信息泄露或 JIT-ROP(Just in time ROP)攻击绕过 ASLR.

### 3.3　ROPDetector 的具体设计

经过我们实际 ROP 攻击的观察，发现 ROP 攻击在硬件层面具有以下特征：

(1) 每一个 gadget 的执行都会出现错误预测返回指令(mis-predicted return instruction)：每一个 ROP gadget 都是以“ret”指令为结尾，因为这些 gadget 都是攻击者构造的，没有与之对应的“call”指令且在原本的程序中并不会执行，故而都会被返回地址栈(RAS)错误预测. 因此，当一个 ROP 载荷被执行时，会持续地出现错误预测返回指令.

(2) 单一 gadget 的短指令：为了避免执行 gadget 链过程中产生一些不必要的边际效应(如，更改了 CPU 状态、导致寄存器数值被异常修改而无法顺利执行 gadget 链剩余部分等)，每一个 gadget 都仅有几条指令. 最近的研究[5]表明真实攻击中的 ROP gadget 指令数一般都小于 6.

(3) 指令转译后备缓冲区未命中数和数据转译后备缓冲区未命中数会异常增加：由于 ROP gadget 链分布在不同的程序段，所以在执行过程中会不断地在多个页面间跳转，会导致大量的指令转译后备缓冲区(Instruction Translation Lookaside Buffer，ITLB)未命中和数据转译后备缓冲区(Data Translation Lookaside Buffer，DTLB)未命中.

ROPDetector 监控目标程序提取运行时特征进行比对来检测 ROP 攻击. 如图 2 所示，当目标程序运行时，ROPDetector 持续监控多种低级事件：总指令数、返回指令数、错误预测返回指令数、ITLB 未命中数、DTLB 未命中数. 然后，我们在指定的周期内记录当前各种预定事件的数值，再采用如下策略检测 ROP 攻击：

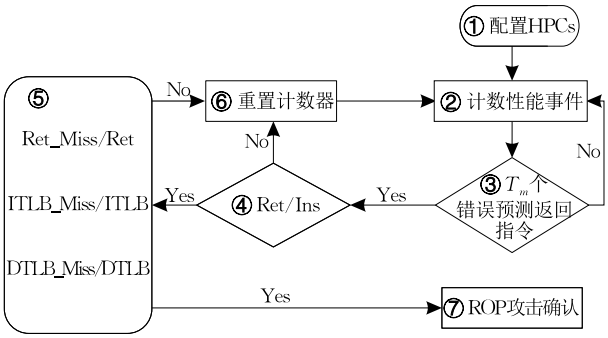


图 2　ROP 攻击检测逻辑

(1) Ret /Ins 表示在每个采样区间返回指令在总指令中的占比. 在 ROP 攻击期间，攻击者利用“ret”指令链接 gadget(即从一个 gadget 跳转到另一

个 gadget). 因此,返回指令在总指令中占比会升高.

(2) Ret_Miss /Ret 表示在每个采样区间错误预测返回指令在返回指令中的占比. 分支预测单元利用调用堆栈(call stack)在运行时预测返回指令目标地址. 然而,ROP 攻击使用大量的返回指令会导致异常的程序控制流执行. 所以,返回指令的未命中率会变得相当高.

(3) ITLB_Miss /ITLB 表示 ITLB 未命中数与 ITLB 访问次数的比率. ITLB 是一个高速缓冲存储器,是用来加快虚拟地址转换速度的内存管理硬件. 在现代处理器中,ITLB 的未命中率是非常低的. 然而,ROP 攻击中,程序执行过程会在多个页面间跳转,导致大量的 ITLB 未命中. 因此,ITLB 的未命中率会相当高.

(4) DTLB_Miss /DTLB 表示 DTLB 未命中数与 DTLB 访问次数的比率. 与 ITLB 情况相似,在 ROP 攻击中,也会导致大量的 DTLB 未命中. 因此,DTLB 的未命中率会相当高.

ROPDetector 通过配置 HPCs 来监控以上硬件事件,检测逻辑如图 2 所示. 当有 $T_m$ 个错误预测返回指令发生时,目标程序执行被挂起,然后触发检测逻辑. ROPDetector 从预先设配置的 HPCs 中读取对应事件的计数,如果 Ret/Ins、Ret_Miss/Ret、ITLB_Miss/ITLB、DTLB_Miss/DTLB 在设定的周期内超过阈值,ROPDetector 将报告 ROP 攻击发生和终止目标程序执行. 否则的话,ROPDetector 将重置计数器并继续进行下一个周期检测.

# 4 系统实现

## 4.1 整体架构

本文实现的 ROPDetector 作为一个轻量级内核模块安装到系统中,其架构如图 3 所示,该内核模块与 perf 驱动程序[①]协同使用. perf 驱动程序用于实现对 HPC 和 LBR 寄存器的控制,通过 perf 驱动程序可实现对 HPC 和 LBR 的配置,来实现对事件发生数的统计功能;此外,当错误预测的返回指令数达到预定的 T 时,该驱动程序还将中断被检测程序,以便 ROPDetector 执行检测逻辑. ROPDetector 检测模块作为判断逻辑的核心,主要实现两个功能:(1) 对 perf 驱动程序的操纵,实现对多种指令数的计数,并获得该驱动程序返回的多种指令的计数值;(2) 根据 perf 驱动程序返回的多种指令计数值,判断是否受到 ROP 攻击,并分别做出处理.

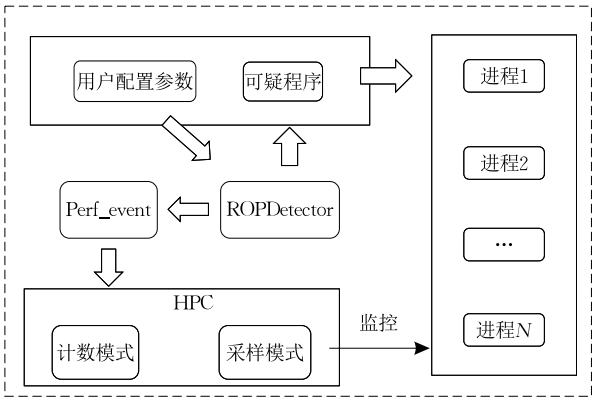


图 3 ROPDetector 总体架构

## 4.2 特征数据收集

ROPDetector 对 ROP 攻击的检测主要涉及以下特征数据:(1) 总指令数;(2) 返回指令数;(3) 预测失败的返回指令数;(4) ITLB 未命中数;(5) DTLB 未命中数;(6) ITLB 访问数;(7) DTLB 访问数. 因此指令数的统计主要统计这些指令,其中总指令数、ITLB 未命中数、DTLB 未命中数、ITLB 访问数、DTLB 访问数通过 HPC 硬件来统计,返回指令数和预测失败的返回指令数通过 HPC 和 LBR 配合使用来完成数据收集.

这些指令数的统计都通过 perf 驱动程序来完成,而 perf 驱动程序主要使用 Linux 内核中的 perf_event 来实现. Perf_event 主要有两种工作模式:计数模式和采样模式,两种模式均基于事件,即当被检测事件发生时就会增加一次计数或进行一次采样. 其中,计数模式主要实现对 HPC 的控制,本文中对总指令数的统计也是通过计数模式来实现;而采样模式相比计数模式会更复杂,采样时,往往是通过 HPC 和其它专用寄存器配合使用,如只有当 HPC 和 LBR 寄存器共同作用时,才能实现对分支指令的采样,进而才能判断分支预测成功与否.

(1) 返回指令数和错误预测返回指令数的统计. 本文对这两种指令数的统计均是通过 perf_event 的采样模式来实现. 具体实现过程中,由于 perf_event 的采样模式是基于计数模式,即采样是通过计数来触发,当计数达到一定阈值时将触发一次采样,因此使用采样模式时需要先配置采样触发所需要的 HPC 计数器,然后再设置采样的频率和欲采样的信息. 其中 HPC 计数器监测返回指令事件,并将采样频率设置为 1,即每执行一次返回指令则进行一次

① Performance counters for Linux. http://lwn.net/Articles/310176, 2010

采样，尽可能保证每一次采样获得的数据包含一条返回指令的信息. 对于采样信息的设置，我们将采样的类型设置为 PERF_SAMPLE_BRANCH_STACK，即采样 LBR 寄存器中的数据，而 LBR 中具体的分支类型设置为 PERF_SAMPLE_BRANCH_ANY_RETURN，即 LBR 中只存放返回指令信息. 到此，采样的配置即可完成，下面将考虑采样信息的处理.

采样信息的处理是基于信号的，即 perf 采样完成后会发出采样处理的信号，因此需要设置信号发出的频率，从而控制样本处理的频率. 考虑到采样与处理的及时性，本文中将信号发出的频率设置为 1，即每完成一次采样则发出一次信号，进行一次处理. 在处理过程中，遍历每一条 LBR 中返回指令的信息，判断该返回指令是否为错误预测的指令，同时也可统计返回指令数，如此可保证返回指令数和错误预测返回指令数完全保证同步.

(2) 总指令数的统计. 总指令数的统计是通过 perf_event 的计数模式来实现，即使用 HPC 对 PERF_COUNT_HW_INSTRUCTIONS 事件发生次数进行计数.

(3) ITLB 访问数与 ITLB 未命中数的统计. ITLB 访问数与 ITLB 未命中数的统计通过 perf_event 的计数模式统计，通过配置 HPCs 对 PERF_COUNT_HW_CACHE_ITLB 进行统计，再设置 PERF_COUNT_HW_CACHE_RESULT_ACCESS 和 PERF_COUNT_HW_CACHE_MISS 分别对 ITLB 访问数和 ITLB 未命中数进行配置.

(4) DTLB 访问数与 DTLB 未命中数的统计. DTLB 访问数与 DTLB 未命中数的统计通过 perf_event 的计数模式统计，通过配置 HPCs 对 PERF_COUNT_HW_CACHE_DTLB 进行统计，设置 PERF_COUNT_HW_CACHE_RESULT_ACCESS 和 PERF_COUNT_HW_CACHE_MISS 分别对 DTLB 访问数和 DTLB 未命中数进行配置.

在该 HPC 值的读取方面，由于对返回与错误预测返回指令的统计是通过采样来完成，而对采样的处理有一定延迟性，因此为了保证多种指令的同步，我们将对上述其它 HPC 值的读取也放在采样中. 如此，处理一次采样即可完成对多种指令数的统计，并使得多个采样数据保持同步.

此外，本文使用 perf 工具对两种事件进行了统计，因此使用了 perf 文件描述符分别处理这两类事件. 为了保证三种指令的同步性，这两个文件描述符需要同时开启与关闭，故我们将这两个文件描述符放到同一个 perf 事件组中.

### 4.3　攻击检测

ROPDetector 通过配置 HPCs 收集完各种指令数据后，根据指定的检测周期 $T_m$，即错误预测返回指令的数目，进行周期性的攻击检测. 在每个检测周期 $T$ 内挂起目标程序，我们通过对 ROP 攻击发生时的特征进行提取，设定了以下几个阈值：

(1) Th_Ret_M_Ret：表示错误预测返回指令在返回指令中的占比.

(2) Th_Ret_Ins：表示返回指令在总指令中的占比.

(3) Th_DTLB_M：表示 DTLB 未命中与访问数的比值.

(4) Th_ITLB_M：表示 ITLB 未命中与访问数的比值.

首先我们将判断错误预测返回指令占比是否超过设定阈值，若超过，则进行其它阈值条件判断，确定 ROP 攻击后进行处理，杀死目标进程，并报告 ROP 攻击发生若无 ROP 攻击发生，则进行下一周期判断.

## 5　实验与评估

### 5.1　实验设置

为了验证 ROPDetector 对于 ROP 攻击检测的效果，我们在如表 3 所示的实验环境下实现我们的原型系统并构造了相应的 ROP 攻击实例.

表 3　实验环境

| 软/硬件 | 参数 |
|---|---|
| 操作系统 | 32 位 Ubuntu 14.04 |
| CPU | Intel(R) Core(TM) i5-7400 CPU@ 3.00 GHz |
| 内核版本 | Linux 4.15.0-65-generic |

#### 5.1.1　参数设置及其影响

在 ROPDetector 中，检测周期 $T$（即错误预测返回指令的数据）的值设定比较关键. 因为在 ROP 攻击检测中，检测周期的不同会影响到检测的精度，而检测周期又取决于 gadget 链的长度. 根据近期研究表明[5]，一般的 gadget 链长度在 11～16 之间，故而我们将 gadget 链长度设置为 12. 那么，为了降低误报和漏报，同时实现最佳精度，我们将检测周期 $T$ 的值设置为 6、9 进行对比实验.

在攻击检测过程中，我们通过利用 20 个真实的 ROP 攻击来获取 4.3 节中 4 个阈值在发生时的异常，如图 4 所示，确定了 Th_Ret_M_Ret 为 0.7，

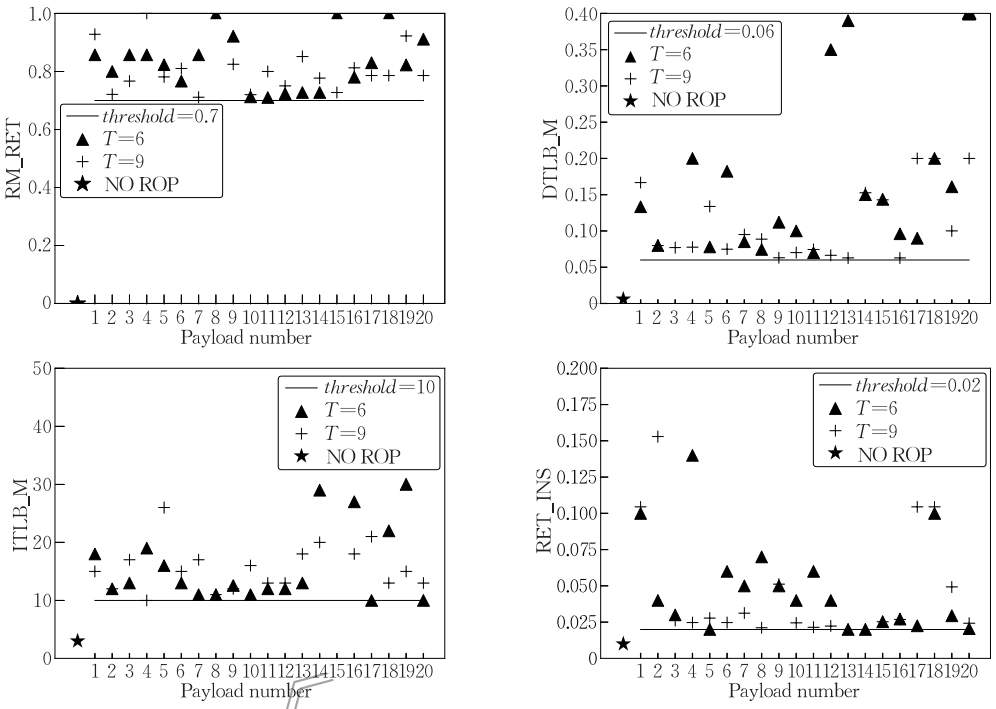


图 4 ROP 攻击发生时底层异常硬件事件

Th_Ret_Ins 为 0.02，Th_DTLB_M 为 0.06，Th_ITLB_M 为 10.

在本实验中，我们从 32 位 Linux 的共享库文件中寻找 gadget，如表 4 所示我们统计了共享库文件中 gadget 数量. 同时，我们假设每一个 gadget 的指令数都少于 6 条指令，这符合大多数真实的 ROP 攻击. 然而，一些聪明的攻击者会尝试使用 long gadget（超过 6 条指令的 gadget）来降低 mis-predicted return instructions 的比例来绕过大多数单纯依赖 mis-predicted return instructions 占比的检测机制，但在 ROPDetector 中还利用局部性原理的其它检测机制组合来检测这种攻击.

**表 4 共享库文件中 gadget 数目**

| 序号 | 名称 | 大小/k | Gadget 数目 |
|---|---|---|---|
| 1 | ld-2.23.so | 223.0 | 3675 |
| 2 | libacl.so.1.1.0 | 76.0 | 1181 |
| 3 | libcap.so.2.24 | 32.5 | 527 |
| 4 | libdns-export.so.162.1.3 | 1300.4 | 19992 |
| 5 | libssl.so.1.0.0 | 461.0 | 6986 |
| 6 | libwrap.so.0.7.6 | 50.9 | 799 |
| 7 | libpam.so.0.83.1 | 65.9 | 1011 |
| 8 | libnss_files-2.23.so | 49.6 | 801 |
| 9 | libcrypto.so.1.0.0 | 2222.0 | 33494 |
| | …… | | |

### 5.1.2 ROP 真实攻击检测

在 Ubuntu 下，如图 5 所示，我们选择了一个带有 ROP 漏洞的小程序[27]，它需要一串很长的输入参数，此程序用 GCC 编译为二进制可执行文件. 我们利用 ROPgadget① 构造了 40 条不同 ROP gadget 链，再从 Exploits Database② 上选取 Linux X86 的 shellcode，即 40 个攻击脚本，每一条 gadget 链长度都超过 12，总共相当于 40 次攻击执行. 同时，我们在检测周期分别为 6 次错误预测返回指令和 9 次错误预测返回事件的情况下进行了实验. 同时，我们在和本文实验环境一致条件下复现了 SIGDROP，因为其基本的检测原理是在一个检测周期内 gadget 长度与错误预测返回指令数的乘积小于等于总指令数作为检测逻辑来判断是否发生 ROP 攻击，所以复现并不复杂. 我们将 ROPDetector 和 SIGDROP 来

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

void vulnerable_function(){
  char buff[128];
  read(STDIN_FILENO, buff, 256)
}
int main{int argc, char** argv}{
  vulnerable_function();
  write(STDIN_FILENO, "Hello World\n", 13);
  return 0;
}
```

图 5 漏洞程序代码

① SRopgadget. https://pypi.python.org/pypi/ROPGadget
② Exploits database. https://www.exploit-db.com/

进行对比实验. 检测结果如表 5 所示，在不同周期下我们成功检测到不同的 ROP payload，结果显示为检测到数量/成功检测比率.

**表 5　构造攻击检测结果**

| 类别 | 数量 | SIGDROP | ROPDetector |
|---|---|---|---|
| 正常输入 | 40 | 0/0 | 0/0 |
| ROP 攻击 | 40 | 25/62.5 | 36/90 |

另外，本文从 Exploits Database 选取了 4 个真实的漏洞进行攻击测试，如表 6 所示，测试结果表明 ROPDetector 能够成功检测到真实的 ROP 攻击.

**表 6　CVE 漏洞检测结果**

| 漏洞程序 | 漏洞编号 | 是否成功检测 |
|---|---|---|
| Nginx1.3.9 | CVE-2013-2028 | √ |
| Glibc2.2.0 | CVE-2015-7547 | √ |
| FFmpeg3.2.1 | CVE-2016-10190 | √ |
| DNSTracer1.8 | CVE-2017-9430 | √ |

## 5.2　性能评估

为了评估本方法部署在真实操作系统中的总体性能影响和对其它同一系统中的应用的性能影响，我们使用 UnixBench 5.1.3[①] 对整体系统性能进行评估，使用 Bonnie++1.03[②] 对磁盘 I/O 性能进行评估，使用 Apache Web Server Httpd 2.4.41[③] 对网络 I/O 性能进行评估.

### 5.2.1　整体性能评估

我们选择使用 UnixBench 对 ROPDetector 进行整体性能评估. UnixBench 是一个类 unix(Unix，BSD，Linux)系统下的性能测试工具，被广泛用于测试 Linux 系统主机的性能. Unixbench 的主要测试项目有：系统调用、读写、进程、图形化测试、2D、3D、管道、运算、C 库等系统基准性能，更进一步还包含 system 和 graphic 测试，在运行 10～30 min 后，即可得到各项测试得分结果，分数越高，性能越强.

为了得到 ROPDetector 的整体性能，我们使用 UnixBench 对操作系统中运行 ROPDetector 和未运行 ROPDetecor 的情况分别进行性能测试. 同时，再设定不同的检测周期 $T$ 进行性能对比. UnixBench 总计进行了 11 项测试，分别是 dhry2reg(字符串处理)、Whetston-double(浮点运算效率和速度)、syscall(系统调用消耗)、pipe(管道吞吐)、context1(基于管道的上下文交互)、spawn(进程创建)、execl(execl 函数吞吐)、fstime(文件读、写、复制)、fsbuffer(文件读、写、复制)、fsdisk(文件读、写、复制)、shell(shell 脚本测试). 结果如图 6 所示，运行了 ROPDetector 的系统，平均性能消耗仅 5.05% 和 5.25%，在可接受范围之内.

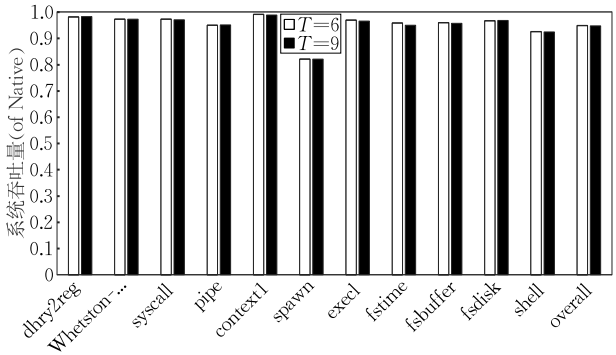


图 6　UnixBench 系统性能测试结果

### 5.2.2　磁盘 I/O 性能评估

我们使用磁盘压力测试工具 Bonnie++1.03 进行磁盘 I/O 压力测试，此工具依次以不同方式到特殊文件中读取、写入数据，读、写的粒度从字符变化到块级别，此次测试的文件数据大小为 15824 MB. 同时，我们还测试了 Random Seeks 的时间消耗. 测试结果如图 7 所示，在检测周期 $T=9$ 时，磁盘 I/O 性能平均损失 2%，而在检测周期 $T=6$ 时，磁盘 I/O 性能平均仅损失 0.94%，该结果表明 ROPDetector 的运行对系统中的磁盘 I/O 性能影响很小.

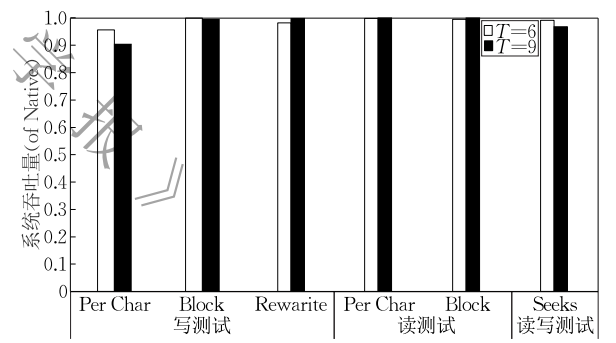


图 7　Bonnie++磁盘 I/O 性能测试结果

### 5.2.3　网络 I/O 性能评估

我们使用 Apache web 服务器 httpd-2.4.41 进行网络 I/O 性能评估，在本实验中，我们使用 2 台在同一个局域网内的计算机，在部署 ROPDetector 的主机上搭建 web 服务器，在另一台主机上不断发送访问 web 服务器的请求包进行网络 I/O 压力测试.

为了更好地进行性能评估，我们在请求主机上安装 ab 工具(即 Apache Benchmark)，该工具是用于测试 Apache 超文本传输协议(HTTP)的工具，

① Unixbench. http://code.google.com/p/byte-unixbench/
② Russell C. Disk Performance Benchmark Tool - Bonnie. http://www.coker.com.au/bonnie++/
③ The Apache Software Foundation, Apache HTTP Server. http://httpd.apache.org/

它会创建众多并发访问线程,模拟大量访问者同时对相同统一资源定位符(Uniform Resource Locator, URL)进行访问,用来测试 Apache 的负载能力.我们在请求主机上使用语句"ab -n 100000 -c 50 http://主机地址/index.html"对服务器以 50 并发访问、总计 100 000 次访问进行网络压力测试.此外,除了 Apache 服务器本身默认的 45 B 大小的网页外,我们还自己写了一个 8KB 大小的网页进行测试.为了更清楚地衡量测试结果,我们以每秒请求数、用户平均等待时间、服务器平均处理时间为指标进行测试,测试结果如图 8 所示,在检测周期 $T=6$ 时,访问默认网页时的网络 I/O 性能消耗仅 0.54%,在访问 8K 网页时的网络 I/O 性能消耗仅有 0.06%;在检测周期 $T=9$ 时,访问默认网页时的网络 I/O 性能消耗仅 1%,在访问 8K 网页时的网络 I/O 性能消耗仅有 0.78%.该结果表明 ROPDetector 在运行时对系统中网络 I/O 性能的影响微乎其微.

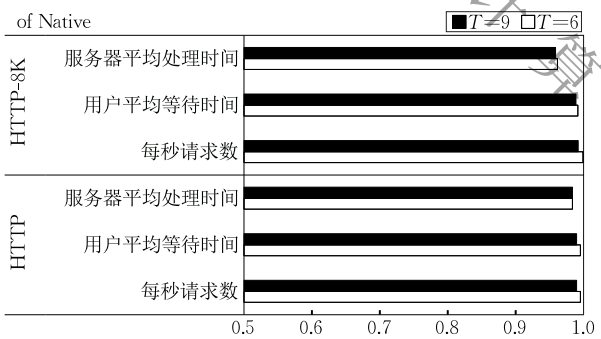


图 8 网络 I/O 性能测试结果

综上,我们通过 UnixBench 评估该方法的整体系统性能消耗,通过 Bonnie++评估了该方法磁盘 I/O 性能消耗,通过 Apache web 服务器 httpd-2.4.41 评估了该方法的网络 I/O 性能消耗,评估结果如表 7 所示,当 ROPDetector 的检测周期 $T$ 为 6 个返回指令预测失败事件时,系统整体性能下降 5.05%,磁盘 I/O 性能下降 0.94%,网络 I/O 性能下降 0.06%;当 ROPDetector 的检测周期 $T$ 为 9 个返回指令预测失败事件时,系统整体性能下降 5.05%,磁盘 I/O 性能下降 0.94%,网络 I/O 性能下降 0.06%.

**表 7 各项性能测试结果**

| 检测周期 $T$ | UnixBench/% | 磁盘 I/O /% | 网络 I/O /% |
|---|---|---|---|
| 6 | 5.05 | 0.94 | 0.06 |
| 9 | 5.25 | 2.00 | 0.78 |

## 6 结论与展望

ROPDetector 是一种基于硬件的实时 ROP 检测方法,它在内核中使用 perf_event 配置现代 CPU 中固有的硬件性能计数器去监控、收集 ROP 攻击发生时的底层硬件事件特征,并利用所收集到的数据进行攻击检测.而且,我们在 32 位 Linux 操作系统下实现了 ROPDetector 的原型系统,并构造了真实 ROP 攻击,与 SIGDROP 进行了对比实验;同时又使用真实的 CVE 漏洞验证了该方法的正确性,实验结果表明该方法可以准确地检测到 ROP 攻击,且各项性能消耗较低.

下一阶段,我们计划进一步降低 ROPDetector 的性能消耗和提高检测精度.首先,要实现检测周期的动态调整,这样能够优化我们的检测代码,可以进一步降低系统各项性能消耗.其次,我们致力于寻找更多的、更显著的 ROP 攻击发生时的底层硬件事件异常特征,还可以在有限增加系统性能消耗的前提下结合机器学习算法(如聚类、SVM、随机森林等)来提高检测精度,降低误报和漏报.最后,提高 ROP 攻击检测的广度,例如检测与 ROP 攻击类似的 JOP (Jump-Oriented Programming),更进一步地,可以检测所有的代码复用攻击.

## 参 考 文 献

[1] Buchanan E, Roemer R, Shacham H, et al. When good instructions go bad: Generalizing return-oriented programming to RISC//Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS). Alexandria, USA, 2008: 27-38

[2] Shacham H, Page M, Pfaff B, et al. On the effectiveness of ad-dress-space randomization//Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS). Washington, USA, 2004: 298-307

[3] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)//Proceedings of the 14th ACM conference on Computer and Communications Security (CCS). Alexandria, USA, 2007: 552-561

[4] Hund R, Holz T, Freiling C F. Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms//Proceedings of the 18th USENIX Security Symposium (SEC). Montreal, Canada, 2009: 383-398

[5] Onarlioglu K, Bilge L, Lanzi A, et al. G-Free: Defeating return-oriented programming through gadget-less binaries//

Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC). Austin, USA, 2010: 49-58

[6] Bletsch T, Jiang Xu-Xian, Freeh V. Mitigating code-reuse attacks with control-flow locking//Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC). Orlando, USA, 2011: 353-362

[7] Li Jin-Ku, Wang Zhi, Jiang Xu-Xian, et al. Defeating return-oriented rootkits with "return-less" kernels//Proceedings of the 5th European Conference on Computer Systems (EuroSys). Paris, France, 2010: 195-208

[8] Tian Shuo, He Ye-Ping, Ding Bao-Zeng. Prevent kernel return-oriented programming attacks using hardware virtualization //Proceedings of the 8th International Conference on Information Security Practice and Experience (ISPEC). Hangzhou, China, 2012: 289-300

[9] Xia Yu-Bin, Liu Yu-Tao, Chen Hai-Bo, et al. CFIMon: Detecting violation of control flow integrity using performance counters//Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). Denver, USA, 2012: 1-12

[10] Davi L, Dmitrienko A, Egele M, et al. MoCFI: A framework to mitigate control-flow attacks on smartphones//Proceedings of the 19th Annual Network & Distributed System Security Symposium (NDSS). San Diego, USA, 2012: 27-40

[11] Polychronakis M, Keromytis D A. ROP payload detection using speculative code execution//Proceedings of the 6th International Conference on Malicious and Unwanted Software (MALWARE). Fajardo, USA, 2011: 58-65

[12] Sinnadurai S, Zhao Qin, Wong Wengfai. Transparent runtime shadow stack: Protection against malicious return address modifications. 2008. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.120.5702

[13] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity soft-ware//Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS). San Diego, USA, 2005: 1-43

[14] Hiser J, Nguyen-Tuong A, Co M, et al. ILR: Where'd my gadgets go?//Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P). San Francisco, USA, 2012: 571-585

[15] Davi L, Sadeghi A, Winandy M. ROPdefender: A detection tool to defend against return-oriented programming attacks//Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS). Taipei, China, 2011: 40-51

[16] Zhang Chao, Wei Tao, Chen Zhao-Feng, et al. Practical control flow integrity and randomization for binary executables//Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P). Berkeley, USA, 2013: 559-573

[17] Pappas V, Polychronakis M, Keromytis D. A. Transparent ROP exploit mitigation using indirect branch tracing//Proceedings of the 22nd USENIX Conference on Security (SEC). Berkeley, USA, 2013: 447-462

[18] Cheng Yue-Qiang, Zhou Zong-Wei, Yu Miao, et al. ROPecker: A generic and practical approach for defending against ROP attacks//Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS). San Diego, USA, 2014: 1-14

[19] Moula V, Niksefat S. ROPK++: An enhanced ROP attack detection framework for Linux operating system//Proceedings of the International Conference on Cyber Security and Protection of Digital Services (Cyber Security). London, UK, 2017: 1-6

[20] Zhou Hong-Wei, Wu Xin, Shi Wen-Chang, et al. HDROP: Detecting ROP attacks using performance monitoring counters //Proceedings of the International Conference on Information Security Practice and Experience (ISPEC). Fuzhou, China, 2014: 172-186

[21] Chen Ping, Xiao Hai, Shen Xiao-Bin, et al. DROP: Detecting return-oriented programming malicious code//Proceedings of the 5th International Conference on Information Systems Security (ICISS). Kolkata, India, 2009: 163-177

[22] Wang Xue-Yang, Backer Je. SIGDROP: Signature-based ROP detection using hardware performance counters. arXiv preprint arXiv:1609.02667. 2016

[23] Das S, Chen Bi-Huan, Chandramohan M, et al. ROPSentry: Runtime defense against ROP attacks using hardware performance counters. Computers & Security. 2018, 73: 374-388

[24] Kolli A, Saidi A, Wenisch F T. RDIP: Return-address-stack directed instruction prefetching//Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Vancouver, Canada, 2013: 260-271

[25] Wang X, Karri R. NumChecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters//Proceedings of the 50th Design Automation Conference (DAC). Austin, USA, 2013: 1-7

[26] Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3. Technical Report, 2016. http://www.zota.ase.ro/bti/Manual%20procesor%20Intel.pdf

[27] Le L. Payload already inside: Datafire-use for ROP exploits//Proceedings of the Black Hat USA 2010. Las Vegas, USA, 2010: 49-54

**NIU Wei-Na**, Ph. D., lecturer. Her main research interests include network attack detection, malware analysis.

**ZHAO Cheng-Yang**, M. S. candidate. His main research interest is network attack detection.

**ZHANG Xiao-Song**, Ph. D., professor. His main research interests include network attack detection, malware analysis.

**HUANG Xiao-Xiang**, M. S. candidate. His main research interest is system security.

**JIANG Lian**, M. S. candidate. His main research interest is network attack detection.

**ZHANG Ke-Xuan**, M. S. candidate. His main research interest is binary security.

**Background**

Return-oriented programming is an advanced code reuse attack technique. The basic idea of ROP is to reuse instructions already residing in memory (e. g. shared libraries, software binary) to induce arbitrary code execution. ROP is a huge threat because it can tamper with the program execution flow to launch an attack. So, there are a lot of defense methods proposed by researcher. However, many existing defense techniques can be bypassed by attackers, or suffer from high performance overhead.

This paper focuses on leveraging hardware to detect ROP attack. We propose a real-time detection approach based on hardware without any side information (e. g., source code, compiler support), which use hardware performance counters in modern CPU to monitor target program execution process and extract the low-level feature of hardware events against ROP attack in runtime. Then, we implement a prototype on x86-based Linux platform called ROPDetector. We find a small program with buffer overflow vulnerability, and use ROPGadget to construct 40 real ROP payloads. We reproduce other method which is also based on hardware performance counters for comparative experiments and use real exploits which are found on the open source exploitation platform to verify feasibility of ROPDetector. The experimental results show that our approach could efficiently detect real ROP attack with a higher detection rate. At last, We evaluate the performance overhead of our method through UnixBench, the disk I/O performance overhead of our method through Bonnie++, and the network I /O performance overhead of our method through the Apache web server httpd-2. 4. 41. When the detection interval is 6 times or 9 times mis-predicted return instructions, the performance overhead is only 5. 05% and 5. 25%, the performance overhead of disk I/O is only 0. 94% and 2%, the performance overhead of network I/O is only 0. 06% and 0. 78%.

This work is partially supported by the National Key Research and Development Program of China (No. 2016QY13Z2302), which aims to research on vulnerability automated discovery, the National Natural Science Foundation of China (No. 61902262 and No. U19A2066), which aims to discover malicious code and cyberattacks in blockchain platforms, respectively, and Science and Technology Support Project of Sichuan Province (No. 2017CC0071), which aims to research on big data security technology.

Our research group has been working on program security, APT detection, system security, blockchain security, data security for many years. Related works were published in good-reputation journals and conferences, such as TIFS, TSE, TOS, CCS, ICSE, and INFOCOM.