软件学报ISSN 1000-9825, CODEN RUXUEW E-mail: jos@iscas.ac.cn

*Journal of Software*, [doi: 10.13328/j.cnki.jos.000000] http://www.jos.org.cn

©中国科学院软件研究所版权所有. Tel: +86-10-62562563

基于硬件分支处理的控制流劫持攻击防御方法[[1]](#footnote-2)∗

李威威1,2,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 100190)

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

摘 要: 控制流劫持攻击是当今安全领域面临的主要威胁之一，面向返回编程（Return-Oriented Programming，简称ROP）攻击是最常见的控制流劫持攻击方式。控制流完整性（Control Flow Integrity，简称CFI）是针ROP攻击的一种主要防御方式，已有的控制流完整性方法通常面临以下几个问题：1）需要修改源码或者通过反汇编重写二进制代码；2）检查点过多；3）历史覆盖不足；4）对短配件链攻击不能进行有效检测，这些问题的存在大大影响了防御效果和性能。本文针对这些问题，提出了一种基于硬件分支处理的控制流劫持攻击防御方法。该方法基于最近分支记录（Last Branch Register）获取精确的分支信息，通过性能监控单元（Performance Monitoring Unit）得到预测失败的间接分支，结合内存信息和系统调用信息，识别出ROP攻击可能利用的配件链（gadget chain），并设计了相应的检测规则进行控制流劫持攻击检测。实验表明，该方法能够有效减少检测点，并解决历史覆盖不足的问题，在有效检测ROP攻击的同时，仅引入极小的性能开销。

关键词: ROP攻击，控制流劫持攻击，控制流完整性

Control Flow Integrity Defense Approach Based on Hardware Branch Processing

LI Weiwei1,2, WANG Junjie1,GAO Weiyi1,2,YANG Qiusong1, LI Mingshu1

1(Institute of Software Chinese Academy of Science, Beijing, China 100190)

2(University of Chinese Academy of Sciences, Beijing, China 100190)

**Abstract**: Return-Oriented Programming (ROP) attack is the most prevalent control flow hijacking attack and is one of the major security threats in the field of software security. Control Flow Integrigy (CFI) is a major defense against ROP attacks. However, traditional CFI methods usually face two major drawbacks: 1) introducing a large performance overhead; 2) need to modify the source code or rewrite the binary code through disassembly. To address these two drawbacks, this paper proposes a control flow integrity defense approach based on the hardware branch processing machnism. Our approach uses the hardware performance monitoring unit for data sampling, and the performance monitor interrupt triaggered by the branch sampling event is used as the ROP detection point. Experiment results show that our approach can effectively detect ROP attacks, and will only introduce neglectable performance overhead.

**Key words**: ROP attack, control flow hijacking attack, control flow integrity

控制流劫持攻击一直以来是软件安全领域的重要研究课题之一。面向返回的编程（Return-Oriented Programming，简称ROP）攻击作为当前最主要的控制流劫持攻击方式更是近年来的研究重点，ROP攻击是一种利用代码复用技术的攻击方法，攻击者扫描已有的动态链接库和可执行文件，提取出以返回指令结尾的指令片段（又叫配件或gadget），利用这些指令片段操作程序栈相关的寄存器，以此控制程序的流程，实现攻击者预设的目标[1,2,3,4]。

控制流完整性(Control Flow Integrity，简称CFI)技术作为一种直接面向控制流进行保护的方式，能够比较有效的防御ROP攻击。CFI技术通过监视程序运行过程中的控制流转移过程，使其始终处于原有控制流图所限定的合法范围内。具体的做法是分析程序的控制流图，重点关注间接转移指令，如间接跳转、间接调用和函数返回等指令，获取相应的白名单。在程序运行过程中对间接转移指令的目标进行检查核对，而攻击者对控制流的劫持会导致目标不在白名单中，此时可迅速进行阻断，保证系统安全[5,6,7,8,9]。CFI技术可分为细粒度和粗粒度两种实现方式。细粒度CFI技术（例如[5,6]）严格检查每一个间接转移指令的转移目标，但会严重影响程序的执行效率，通常会带来20%-50%的额外性能损耗；粗粒度CFI技术（例如[7,8,9]）将一组或相近类型的目标归到一起进行检查，可在一定程度上降低开销，但会使安全性降低，攻击者能够通过精心设计的配件（gadget），组合成配件链（gadget chain），继续实施代码重用攻击。以上的CFI技术均是基于二进制插桩的，需要修改源码或者通过反汇编重写二进制代码，这给这些技术的使用带来额外的工作量，并且会引入较大的性能开销或者检测效果不够精确。

此外，一些研究引入硬件机制来降低CFI技术的开销、提高检测精度。例如，Pappas等提出了kBouncer方法[10]，该方法利用最近分支记录（Last Branch Register）捕获最近的16次跳转信息，在敏感系统调用处对捕获的16次跳转进行安全性判断。Cheng等提出ROPecker方法[2]，该方法也是利用最近分支记录（Last Branch Register）捕获程序流的方式进行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配件链来进行攻击检测，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16个间接分支进行检查，会面临历史覆盖不足的问题，导致检测精度下降。Xia等提出了CFIMon[11]，该方法采用分支跟踪缓冲区（Branch Trace Buffer）来捕获程序运行过程中跳转指令的信息。虽然分支跟踪缓冲区能够将程序整个执行过程中的所有跳转指令的历史信息都记录下来，但相比使用最近分支记录，使用分支跟踪缓冲区会引入更大的性能开销。此外，以上三种方法均是针对长配件链的攻击进行检测，对于短配件链的攻击情况检测效果不佳。

因此，针对已有CFI技术面临的影响防御效果和性能的几个问题：1）需要修改源码或者通过反汇编重写二进制代码[7,8,9]；2）检查点过多[5,6,11]；3）历史覆盖不足[2,10]；4）对短配件链攻击不能进行有效检测[2,10,11]，本文提出了一种基于硬件分支处理的控制流劫持攻击防御方法，图1是该方法的基本框架。

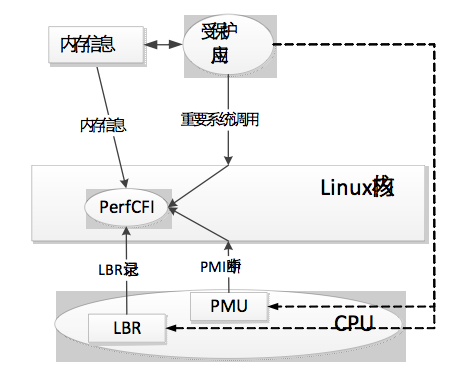


图1. 方法的基本框架

该方法基于最近分支记录（Last Branch Register）获取精确的分支信息，通过性能监控单元（Performance Monitoring Unit）获取预测失败的间接分支，结合内存信息和系统调用信息，识别出ROP攻击可能利用的配件链（gadget chain），并设计了相应的检测规则进行控制流劫持攻击的检测。

本方法具备以下优点：

1）基于最近分支记录（Last Branch Register）获取精确的分支信息，基于该信息进行动态二进制反汇编检测疑似配件（gadget），不需要修改源码或者重写二进制码，能直接针对COTS二进制代码进行保护，增加了方法的实用性。2）通过性能监控单元（Performance Monitoring Unit）采样预测失败的间接分支，对每个预测失败的间接分支处进行CFI检测，避免了对预测正确的间接分支进行检查，能够有效减少检测点、控制性能开销。

3）

3）针对历史覆盖不足的问题，一方面，本方法，这能够帮助；另一方面，针对检测时的系统状态来更精确的判定疑似“空”（NOP）操作，避免长空（long NOP）操作覆盖历史检测数据，更好的缓解了历史覆盖不足的问题。

4）引入系统调用参数检测方法，通过判断其与上一个配件对应的系统架构值之间的相似性来检测攻击，能够识别已有CFI检测方法覆盖不到的短配件链的ROP攻击。

本文后续章节组织方式如下：

1. 首先，介绍和本文方法的研究背景。
2. 然后，对本文提出的PerfCFI检测方法进行详细介绍；
3. 其次，对本文原型系统实现方式进行详细的描述；
4. 接着，对本文实现的原型系统进行性能及安全性评估；
5. 第五，针对相关研究进行阐述；
6. 最后，对本文进行总结与展望。

# 研究背景

。

ROP（Return-Oriented Programming，面向返回编程）：是一种利用代码复用技术的攻击方法。攻击者扫描已有的动态链接库和可执行文件，提取出可以利用的指令片段（gadget），这些指令片段均已ret指令结尾，即用ret指令实现指令片段执行流的链接。操作系统通过栈来进行函数的调用和返回。函数的调用和返回就是通过压栈和出栈来实现的。每个程序都会维护一个程序运行栈，栈为所有函数共享，每次函数调用，系统会分配一个栈桢给当前被调用函数，用于参数的传递、局部变量的维护、返回地址的填入等。ROP攻击就是利用以ret结尾的程序片段，操作这些栈桢相关的寄存器，控制程序的流程，执行相应的gadget，实施攻击者预设的目标。

ROP攻击利用系统中可以利用的指令片段来完成，这些指令片段称为gadget。

CFI（Contro-Flow Integrity，控制流完整性）：通过限制控制流转移的目标来检测控制流劫持攻击。在控制流劫持攻击中，攻击者将程序的控制流重定向到不会到达正常执行的位置。CFI方法通过运行时监控程序的执行、并将其状态与预先设定好的CFG的路径对比来将程序的控制流限制到有效的执行轨迹中。如果检测到一种无效状态，就会发出警告，通常是终止程序的执行。CFG（Control-Flow Graph）通常通过源码分析、二进制分析或者执行剖析得到。

PMU（Performance Monitoring Unit，性能监控单元）：是Intel引入的用于记录处理器事件的功能单元。PMU事件有好几百个，包含了处理器在运行过程中可能遇到的所有情形，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。

BPU（Branch Processing Unit，分支处理单元）：是PMU中专门对分支事件进行处理的单元。分支指令通常是一个逻辑的指令，或者是一个循环的终点，常会伴随内存跳跃指令来执行。BPU会预测每个分支处理的结果。

PMI（Performance Monitoring Interrupt，性能监视器中断）：是性能监控单元PMU产生的中断。PMU中的计数器都是固定大小的(32位/64位取决于CPU的体系结构)，当计数器上溢时，就会产生性能监控中断。

LBR（Last Branch Record，最近分支记录）是Intel提供的一组用于记录和追踪程序最近的若干次跳转信息的循环寄存器组，这些寄存器的数量与Intel处理器的微架构相关，在早几年的Haswell架构中有16个这样的寄存器，也就是说可以记录程序最近的16条跳转指令的信息(包括从哪跳转过来的，将要跳转到哪去)，而在最新的Skylake架构中有32个。LBR寄存器的强大之处在于其定制性很强，能够过滤掉一些不重要的跳转指令，而保留需要重点关注的跳转指令。

# 基于硬件分支处理的控制流劫持攻击防御方法

本文提出了一种基于硬件分支处理的控制流劫持攻击防御方法（简称BranchChecker）。基于PMU分支相关采样事件触发的PMI中断作为ROP检测点，借助PMU、LBR相关采样数据通过动态反汇编的方式进行ROP攻击检测。本章将先对本文针对的威胁模型进行描述，然后对perfcif设计过程中涉及到的采样事件的选择以及攻击检测算法进行详细描述。

## 威胁模型

本文采用与内存破坏及代码重用攻击文献一致的威胁模型：攻击者可以通过远程或本地正常终端访问应用，可以利用应用程序中的内存漏洞任意读取或修改应用数据内存页内容，也可以通过这些漏洞获取所有内存空间排布信息。但假定硬件，操作系统，动态链接器是可信的，目标操作系统支持NX或DEP技术，攻击者不能插入或修改应用代码，即攻击者被限定为仅能采用代码重用攻击。

## PMU采样事件选择

本文基于采样事件触发的PMI事件作为检查点，需要针对所有可能用于ROP攻击的指令片段进行检查。现代CPU提供了多种多样的PMU事件采样用于调试。ROP攻击的指令片段的一个重要特征是需要通过分支操作来实现指令片段间的衔接，因此，本文需要在分支相关事件中进行筛选。筛选的主要原则是：尽量降低采样事件频率，且不会遗漏ROP攻击相关分支。

本文结合ROP攻击特征及硬件相关特性进行了分支事件的选择：

1. PerfCFI针对应用程序进行保护，因此可将事件限定为用户态事件；
2. 基于我们的威胁模型，攻击者不能修改程序代码，因此直接分支的语义不会被修改，ROP攻击只能借助间接分支来进行攻击，因此触发事件可限定为间接分支事件；
3. 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，但最终可能会被抛弃，最终不会提交，所以采样事件分为执行时的分支事件和提交时的分支事件。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），因此可将触发事件限定为分支提交事件；
4. ROP攻击链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他预期的目标，因此可将触发事件限定为跳转分支事件
5. 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，将会与历史跳转目标或方向不一致，因此该分支会导致分支预测失败，因此可将事件限定为预测失败的分支事件；

综上所述，最为理想的分支事件为：用户态预测失败的跳转间接分支提交事件。但实际中各类CPU支持的事件粒度略有不同，因此可供选择的事件并不能保证能够包含最理想的情形，在原型系统实现中会详细描述具体采样事件的选择。

## ROP攻击检测策略

PerfCFI通过捕获系统运行时的硬件分支处理相关的特征，基于特征设计相应的ROP攻击检测策略。

ROP攻击利用系统中可以利用的指令片段来完成，这些指令片段称为gadget。PerfCFI首先识别出疑似gadget和critical gadget。当满足下列条件之一时，认为当前指令片段为疑似gadget：

1. Call-preceded ret：对于所有的return指令，如果目标地址的上一条指令不是call指令，将其标为疑似gadget；
2. Short basic sequence：如果上一个可疑分支到下一个可疑分支的指令长度低于一定阈值maxGadgetLength，将其标为疑似gadget。

为了提高检测的准确性，并且避免对历史信息的覆盖，PerfCFI对长无效片段进行识别和过滤。长无效片段是指副作用较小的长指令片段，通常用于绕过基于连续gadget特征检测的CFI检测方式。如果在上一个疑似gadget末尾到下一个疑似gadget开始，系统状态（主要是架构寄存器）的修改粒度低于一定阈值，将该段代码标为疑似长无效片段。

PerfCFI基于以下策略对gadget长度进行统计：

1. 初始gadget长度为0；
2. 如果是当前分支片段为疑似gadget结尾，那么将gadget长度加1；
3. 如果当前分支片段为疑似长无效片段，那么gadget长度不变；
4. 非2）和3）的情形则将gadget长度清0。

PerfCFI基于以下两个条件进行攻击检测，当满足其中一个条件，认为程序遭受了ROP攻击：

1. 连续的gadget数超过一定阈值maxChainLength；

该检测条件适用于通常的ROP攻击检测，ROP由一连串的gadget构成；

1. 敏感系统调用中的参数和上一个疑似gadget的系统架构值基本一致；

该检测条件适用于通过较短的gadget链直接进行敏感系统调用的情形。

# 原型系统实现

我们在Intel Ivy Bridge平台下的Ubuntu 15.04系统上实现了一个PerfCFI的原型系统，如图1所示。该系统主要由一个Linux可装载内核模块构成，该模块会劫持Linux内核的PMI事件处理以及重要系统调用，并利用LBR信息以及应用内存信息进行CFI检测。

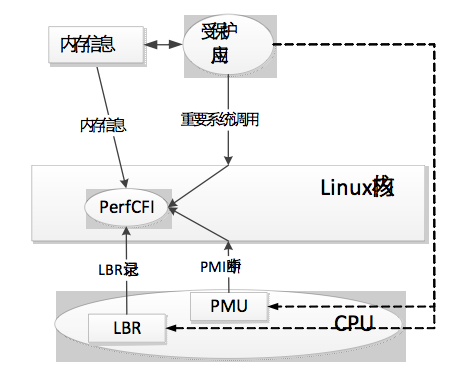


Fig.1 Structure of PerfCFI

图1 PerfCFI组织结构

下面将对检测触发点及具体的CFI检测算法进行详细描述。

## 检测触发点选择

前文中提到，最理想状态的检测触发点为用户态预测失败的跳转间接分支提交事件，但Ivy Bridge平台下并不支持该采样事件（支持的分支采样事件如表1所示），因此我们选择了最接近它的两个“超集”事件作为备选事件：用户态预测失败的跳转分支提交事件（事件编码为0xc5，掩码为0x20）和用户态预测失败的跳转间接分支执行事件（事件编码为0x89，掩码为0xac）。

**Table 1** Branch-related pmu sample event supported by Ivy Bridge

表**1** Ivy Bridge支持的分支采样事件

|  |  |  |  |
| --- | --- | --- | --- |
| 事件编码 | 掩码 | 事件名称 | 事件描述 |
| 0x89 | 0x01 | BR\_MISP\_EXEC.COND | Qualify conditional near branch instructions mispredicted. |
| 0x89 | 0x04 | BR\_MISP\_EXEC.INDIRECT\_JMP\_NON\_CALL\_RET | Qualify mispredicted indirect near branch instructions that are not calls or returns. |
| 0x89 | 0x08 | BR\_MISP\_EXEC.RETURN\_NEAR | Qualify mispredicted indirect near branches that have a return mnemonic. |
| 0x89 | 0x10 | BR\_MISP\_EXEC.DIRECT\_NEAR\_CALL | Qualify mispredicted unconditional near call branch instructions, excluding non-call branch, executed. |
| 0x89 | 0x20 | BR\_MISP\_EXEC.INDIRECT\_NEAR\_CALL | Qualify mispredicted indirect near calls, including both register and memory indirect, executed. |
| 0x89 | 0x40 | BR\_MISP\_EXEC.NONTAKEN | Qualify mispredicted non-taken near branches executed. |
| 0x89 | 0x80 | BR\_MISP\_EXEC.TAKEN | Qualify mispredicted taken near branches executed |
| 0x89 | 0xFF | BR\_MISP\_EXEC.ALL\_BRANCHES | Counts all near executed branches (not necessarily retired) |
| 0xc5 | 0x00 | BR\_MISP\_RETIRED.ALL\_BRANCHES | Mispredicted branch instructions at retirement. |
| 0xc5 | 0x01 | BR\_MISP\_RETIRED.CONDITIONAL | Mispredicted conditional branch instructions retired. |
| 0xc5 | 0x04 | BR\_MISP\_RETIRED.ALL\_BRANCHES | Mispredicted macro branch instructions retired. |
| 0xc5 | 0x20 | BR\_MISP\_RETIRED.NEAR\_TAKEN | Number of near branch instructions retired that were taken but mispredicted. |

针对这两种事件类型，我们利用perf工具对随机选择的Spec2006部分程序进行了采样，采样数据如图2所示：

Fig.2 sample data of the two branch event on part of SPEC2006 benchmark

图3 部分SPEC2006程序上两种分支事件的采样数据

从上图可看出，用户态预测失败的跳转间接分支执行事件相比于用户态预测失败的跳转分支提交事件更少，因此我们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，让其在发生该事件时计数器溢出，进而触发PMI事件。

## ROP攻击检测

本文借助Intel CPU的LBR机制精确的获取了当前触发事件分支指令的地址及目标地址，并通过动态反汇编的方式判定当前分支是否为疑似gadget。

Intel 的 LBR记录有限，因此我们借助了两种硬件机制过滤掉了不必要的分支事件以免影响真实所需的分支数据：

1. 直接利用LBR本身的过滤机制进行分支采样过滤，Ivy Bridge支持的过滤方式如表2所示，通过该过滤机制，我们可以将LBR记录限定为用户态预测失败的跳转间接分支信息（LBR本身不记录不跳转的分支信息）。

**Table 2** Event filter mechanism supported by Ivy Bridge LBR

表**2** Ivy Bridge lbr支持的事件过滤机制

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 位偏移 |  | 描述 |
| CPL\_EQ\_0 | 0 |  | When set, do not capture branches occurring in ring 0 |
| CPL\_NEQ\_0 | 1 |  | When set, do not capture branches occurring in ring >0 |
| JCC | 2 |  | When set, do not capture conditional branches |
| NEAR\_REL\_CALL | 3 |  | When set, do not capture near relative calls |
| NEAR\_IND\_CALL | 4 |  | When set, do not capture near indirect calls |
| NEAR\_RET | 5 |  | When set, do not capture near returns |
| NEAR\_IND\_JMP | 6 |  | When set, do not capture near indirect jumps |
| NEAR\_REL\_JMP | 7 |  | When set, do not capture near relative jumps |
| FAR\_BRANCH | 8 |  | When set, do not capture far branches |
| Reserved | 63:9 |  |  |

1. 启用Intel支持的”freeze\_on\_pmi”特性，让触发PMI事件过后的分支事件不再写入LBR记录当中。

借助LBR中的分支地址及目标地址信息，我们可以在PMI事件处理函数中进行两个方向的动态反汇编进而进行疑似gadget和critical gadget的判定：

1. 以当前分支地址为起点，可以判定当前分支信息，如果为ret指令，那么可以向目标地址低地址方向进行反汇编，判定目标地址上一条指令是否为call指令，如果不是，那么当前分支片段为疑似gadget；
2. 以上一触发事件分支的目标地址为起点，向高地址方向进行反汇编，如果在不包含无条件跳转（包含中断）的情况经过较短（未超过gadget指令序列长度阈值）的指令片段到达当前记录分支地址，那么当前分支片段为疑似gadget；
3. 如果当前指令为系统调用指令，如果其参数和上一个疑似gadget的参数是相同的，则认为当前片段为critical gadget。

为了支持长无效片段的检测，我们需要进行如下支持：

1. 记录上一个疑似gadget末尾系统的状态；
2. 每次触发检查时，与当前分支的状态进行比较，如果未超过阈值，那么将其标为长无效片段；

为了进行疑似critical gadget判定，我们针对部分敏感系统调用进行了修改，在这些系统调用处理函数开始处插入对应的gadget检查机制，通过LBR记录的中断指令信息，采用类似的反汇编技术判定当前系统调用片段是否为疑似critical gadget。

结合这些gadget判定以及相应的ROP攻击判定策略，我们的原型系统会在发生疑似ROP攻击时及时终止受攻击的用户程序。

## 阈值选择

PerfCFI在检测过程中需要设置两个阈值：gadget指令片段最长长度(maxGadgetLength)以及gadget链最长长度(maxChainLength)。

maxGadgetLength阈值的选择需要满足以下条件：

1. 该阈值需要大于绝大部分理想的（状态可控、副作用较小的）gadget长度;
2. 该阈值需要尽量小，以减少误报（避免将正常指令片段识别成gadget）。

综合考虑，本文选取30作为该阈值，经验证，该阈值大于ROPgadget工具在常用软件及库中查找出的绝大多数常用的gadget长度(当前测试中发现最长的状态可控、副作用较小的gadget为27，指令片段如图3所示)。

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.3 The longgest ideal gadget found in the test

图3 测试中发现的最长的“理想”gadget

maxChainLength阈值的选择需要满足以下条件：

1. 该阈值需要大于正常指令执行流中出现的最长疑似gadget链;
2. 该阈值需要尽量小，以减少漏报（避免将ROP攻击链识别成了正常指令流）。

本文针对Spec2006程序以及一些常用的Linux程序进行了采样，识别的最大gadget链长度如下表所示：

**Table 3** Max Length of gadget chain on partial spec2006 and some linux application

表**3** 部分spec2006程序以及一些常用的linux程序中的最大gadget链长度

|  |  |
| --- | --- |
| 程序 | 最大gadget链长度 |
| spec2006(部分) | 3 |
| ls | 3 |
| dmesg | 1 |
| ps | 3 |
| cat | 1 |
| mkdir | 3 |
| ping | 2 |
| stat | 3 |
| passwd | 2 |
| gdb | 4 |
| tar | 2(错误输入强制退出时包含了5个) |
| vi | 4 |

由上表可知，正常运行的程序包含的最大gadget链一般不超过5个。已有研究中对于最大gadget链的长度限定一般在5-20[1,10]。本文采用较为保守的方案，将该阈值设置成10。

# 评估

本节将对PerfCFI原型系统的性能及安全性进行评估。

## 性能评估

我们的原型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3.13.19）。我们采用常用的SPEC2006作为Benchmark对PerfCFI的性能开销进行了评估。

首先，我们针对SPEC2006中分支的相关事件数进行了采样，如表3所示：

Table.3 Branch event count of SPEC2006 benchmark

表3 SPEC2006上分支相关事件计数

|  |  |  |  |
| --- | --- | --- | --- |
|  | 执行的间接分支数 | 预测失败的间接分支执行事件数 | 所需检查的分支减少比例 |
| 429.mcf | 1254075 | 8556327 | 85.34% |
| 445.gobmk | 33486205 | 808133481 | 95.85% |
| 456.hmmer | 1263206 | 8658138 | 85.41% |
| 462.libquantum | 695361 | 5009294 | 86.11% |
| 464.h264ref | 3860320 | 26420321 | 85.38% |
| 471.omnetpp | 3884031 | 28671677 | 86.45% |
| 473.astar | 1278374 | 8642528 | 85.20% |
| 433.milc | 653106 | 4457463 | 85.34% |
| 444.namd | 1185799 | 8169297 | 85.48% |
| 447.dealII | 607929978 | 4557668892 | 86.66% |
| 450.soplex | 2529271 | 17239736 | 85.32% |
| 470.lbm | 665000 | 4557532 | 85.40% |
| 482.sphinx3 | 10352577 | 75785775 | 86.33% |

由上表可以看出本文利用硬件BPU的特性，能够将所需检查的间接分支平均减少86.49%（最多的减少了95%，最少减少了85.21%）。与CFIGuard [19]每16个分支做一次检查相比，PerfCFI所需进行的检查数约为其2.17倍，考虑到系统调用进行的检查，两者实际进行检查数的比率应该低于该值，因此本文检测方法引入的性能开销应该在6.3%以内（16\*2.17=6.3）。

为了进一步确认PerfCFI真实引入的性能开销，本文对开启PerfCFI和未开启PerfCFI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3所示。

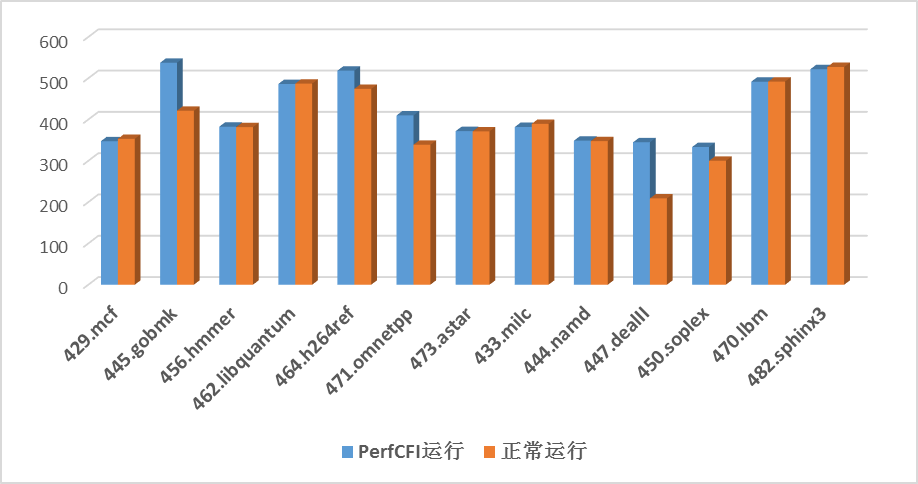


Fig.3 performance cost of PerfCFI on SPEC2006 benchmark

图3 PerfCFI在SPEC2006上的性能开销

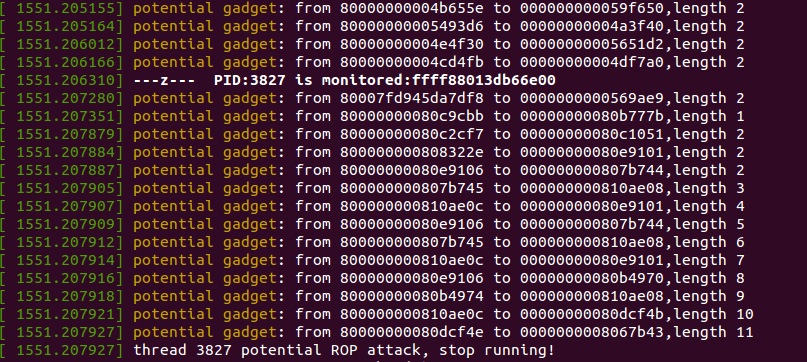
实验结果表明：PerfCFI平均引入5.4%左右的性能开销，对于绝大部分SPEC程序开销均在3%以内，少部分程序开销超过了10%，总体来说，性能开销在可接受的范围以内。

## 安全性评估

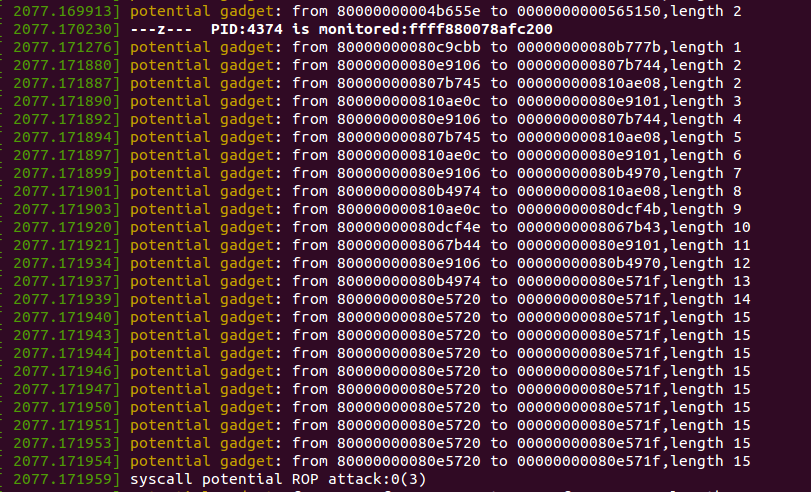
本文提出了PerfCFI检测方法，不同于kbouncer[10]和ROPecker[1]，是针对所有可能用于攻击的间接分支进行ROP检测，能够有效避开两者由于LBR有限引起的绕过问题。针对通过kbouncer和ROPecker会被长gadget打破检测链的问题，本文引入了两种新的检测方式：

1. 本文提出了长无效gadget概念及检测方法，能够有效检测gadget链中基本不修改架构状态的长gadget，规避此类长gadget引入历史刷新及破坏攻击问题；
2. 本文引入的系统调用参数检测方法，通过判断其与最近检测出gadget时保存的系统架构值之间相似度来检测ROP攻击。对于ROP攻击链中引起大量架构状态修改的gadget，虽然会打破本文的常规ROP检测链。但在进行最终高危行为（通常为系统调用）之前，攻击者依旧需要短的gadget链来恢复系统调用所需的状态，会被PerfCFI检测到并保存当时系统状态，如果，在攻击者进行系统调用时被PerfCFI拦截，。

根据分析可知，本文提出的PerfCFI方法能够有效的避开kbouncer和ROPecker的刷新历史以及破坏攻击问题。为了进一步验证PerfCFI的有效性，本文重现了几个现实的攻击实例[20,21]，以jad overflow[20]为例，实验结果如图4所示，红框为检测到的ROP攻击链以及ROP攻击结果。实验结果表明，PerfCFI能够有效的检测出这些攻击，复现完整的攻击gadget链，并及时阻止程序的继续运行。



a) maxChainLength=10



b)maxChainLength=20

Fig.4 PerfCFI running result on jad attack program: maxlength =10 and maxLength =20

图4 PerfCFI在jad攻击程序上的运行结果：maxLength=10和maxLength=20两种情形

为了进一步验证PerfCFI能否检测较短gadget链构成的ROP攻击。本文构造了一个通过较短的gadget链（未超过阈值)调用系统调用来完成ROP攻击的攻击实例，如图5所示，该攻击利用缓冲区溢出漏洞进行ROP攻击，攻击链由3个gadget构成：1) pop rdi;ret ; 2) pop rax;pop rdx;pop rbx;ret ; 3) syscall。

#include <stdio.h>

int main() {

char name[64];

printf("%p\n", name); // Print address of buffer.

puts("What's your name?");

gets(name); // buffer overflow bug

printf("Hello, %s!\n", name);

}

return 0;

}

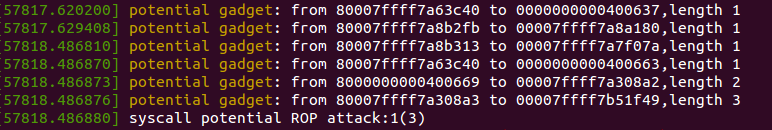


Fig.5 Victim program and ROP gadget chain check result

图5. victim程序以及ROP gadget链检测结果

实验表明，PerfCFI能够有效的检测出通过较短的gadget链进行系统调用的ROP攻击。

# 相关研究

控制流完整性（Control Flow Integrity，简称CFI）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控制流图（Control Flow Graph，简称CFG），以确保程序控制流不被劫持或非法篡改。具体作法是在控制流转移指令前插入检验代码，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[5]提出的细粒度的CFI技术，预先计算出所有间接转移指令的可能的目标地址，并为之赋予唯一的ID；在检查时，在每条间接跳转指令执行前增加检查，一旦违反控制流图的间接跳转就会报错。Abadi等[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，将传统的应用程序包装成XFI模块，在运行时提供保护与监控，并通过设置外部接口与操作系统交互。虽然这些细粒度的CFI技术能够提升系统的安全性，但其开销过大，难以得到实际部署。

粗粒度的CFI技术将一组或相近类型的目标归到一起进行检查，可在一定程度上降低开销。Zhang等[7]提出了CCFIR，该方法对间接调用指令和函数返回指令的目标进行区分，阻止未经验证的返回指令跳转到敏感函数的行为。Zhangn等[8]提出了binCFI，该方法将间接转移指令的操作数分为代码指针、异常处理程序入口、导出符号地址和返回地址等类型，通过精细的静态分析，得到不同类型的间接控制转移的合法目标集合，并基于此进行攻击检测。Mashtizadeh等[12]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、return指针类、方法指针类、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[13]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（可用于实施控制流转移攻击）的执行路径，通过使用二进制插桩涉及，在被监控路径上强制执行上下文敏感CIF的不变量。

以上的CFI技术均是基于二进制插桩的，需要修改源码或者通过反汇编重写二进制代码，这给这些技术的使用带来额外的工作量；并且会引入较大的性能开销或者检测效果不够精确。本文提供的PerfCFI方法并不属于该类别，不依赖于源代码，同时可以直接作用于传统程序，但本文提供的检测框架可以和基于二进制程序的方式结合实现基于CFG的CFI检查，该方法将作为我们未来的研究方向之一。

为了能够在性能和防御效果方面取得更好的效果，一些研究引入硬件机制来降低CFI技术的开销。

Pappas等提出了kBouncer方法[10]，该方法利用LBR（Last Branch Register）捕获最近的16次跳转信息，在敏感系统调用处对捕获的16次跳转进行安全性判断。Cheng等提出ROPecker方法[2]，该方法也是利用LBR捕获程序程序流的方式进行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gadget链来进行攻击检测，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16个间接分支进行检查，会面临历史覆盖不足的问题，导致检测精度下降。Xia等提出了CFIMon[11],该方法采用BTB（Branch Trace Buffer）来捕获程序运行过程中跳转指令的信息。虽然BTB能够程序整个执行过程中的所有跳转指令的历史信息都记录下来，但使用BTB的程序性能明显比LBR性能低。本文提供的方法属于该类别，但相比于其它基于硬件的ROP检测方法，如kbouncer，PerfCFI检查的粒度更小，同时新增了long nop-gadget检查以及系统调用检查，能够有效避免kBouncer被历史覆盖的问题。

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[14-18]，该类技术通过将函数、内存页、基础块或指令排布等随机化，让攻击者不能准确预测所需gadget的位置，无法进行ROP攻击。我们的perfCFI可以和这些一起作用，用于阻止攻击者对应用程序进行ROP攻击。

# 总结及展望

本文针对当今软件安全领域面临的主要安全威胁之一ROP攻击展开研究，提出了PerfCFI ROP攻击检测方法，有效解决了传统的CFI方法面临着两大弊端（之一）：1）引入较大的性能开销；2）需要修改源码或者通过反汇编重写二进制代码。实验表明，该方法能够有效的检测ROP攻击，并且仅会引入的较小的性能开销。

尽管本文仅提出了先验知识的CFI检测方法，但本文的检测框架可以和基于CFG的CFI技术相结合，有效降低对控制流图检测的频率，进而降低基于CFG的CFI技术的性能开销，这也是我们未来的研究方向之一。

References:

1. Davi L, Sadeghi A R, 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, pp. 40-51, 2011.
2. Yueqiang Cheng, Zongwei Zhou, Miao Yu, Xuhua Ding, and Robert H Deng. ROPecker: A generic and practical approach for defending against rop attacks. NDSS14, 2014.
3. Ping Chen, Hai Xiao, Xiaobin Shen, Xinchun Yin, Bing Mao,and Li Xie. DROP: Detecting return-oriented programming malicious code. In Proceedings of the 5th International Conference on Information Systems Security (ICISS), 2009.
4. T. Bletsch, X. Jiang, V. W. Freely and Z. Liang, Jump-oriented programming:a new class of code-reuse attack, in Proceedings of the 6th ,4CM Symposium on Information, Computer and Communications Security,New York, NY, USA, pp. 30-40, 2011.
5. M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity: Principles, implementations, and applications. In ACM Conference on Computer and Communications Security (CCS), 2005.
6. M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti.Control-flow integrity principles, implementations, and applications. ACM Transactions on Information and System Security (TISSEC), (1), Nov. 2009
7. Zhang, T. Wei, Z. Chen, L. Duan, L. Szekeres,S. McCamant, D. Song, and W. Zou. Practical Control Flow Integrity & Randomization for Binary Executables. In IEEE Symposium on Security and Privacy, 2013.
8. M. Zhang and R. Sekar. Control flow integrity for COTS binaries. In USENIX Security Symposium, 2013.
9. B. Niu and G. Tan. Modular control-flow integrity. In PLDI’14, 2014.
10. Vasilis Pappas, Michalis Polychronakis, and Angelos D. Keromytis. 2013. Transparent ROP exploit mitigation using indirect branch tracing. In Proceedings of the 22nd USENIX conference on Security (SEC'13). USENIX Association, Berkeley, CA, USA, 447462.
11. Yubin Xia, Yutao Liu, Haibo Chen, and Binyu Zang. 2012. CFIMon: Detecting violation of control flow integrity using performance counters. In Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (DSN '12). IEEE Computer Society, Washington, DC, USA, 112.
12. Ali Jose Mashtizadeh, Andrea Bittau, Dan Boneh, and David Mazières. 2015. CCFI: Cryptographically Enforced Control Flow Integrity. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, 941951.
13. Victor van der Veen, Dennis Andriesse, Enes Göktaş, Ben Gras, Lionel Sambuc, Asia Slowinska, Herbert Bos, and Cristiano Giuffrida. 2015. Practical ContextSensitive CFI. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, 927940.
14. L. Davi,A. Dmitrienko, S. Nűrnberger,A-R. Sadeghi. Gadge me if you can: Secure and efficient ad-hoc instruction-level randomization for x86 and arm. In 8th ACM SIGSAC symposium on Information, computer and communications security ,2013.
15. Wartell, R., Mohan, V., Hamlen, K.W., Lin, Z. Binary stirring: self-randomizing instruction addresses of legacy x86 binary code. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 157–168, 2012.
16. M. Backes, S. Nurnberger, Oxymoron: Making Fine-Grained Memory Randomization Practical by Allowing Code Sharing, in the Proceedings of the 23rd USENIX Security Symposium,2014
17. V. Pappas, M. Polychronakis, and A. D. Keromytis. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization. In IEEE Symposium on Security and Privacy, 2012.
18. 詹珣. 针对ROP攻击的块粒度地址空间随机化防御技术的研究[硕士学位论文]. 南京. 南京大学软件学院 2014
19. Pinghai Yuan, Qingkai Zeng, Xuhua Ding. Hardware-Assisted Fine-Grained Code-Reuse Attack Detection. International Workshop on Recent Advances in Intrusion Detection. Pp.66-85
20. “HT Editor 2.0.20 Buffer Overflow（ROP PoC）,” http://www.exploit-db.com/exploits/22683/.
21. “JAD Buffer Overflow（ROP PoC）,” <http://www.exploit-db.com/exploits/42357/>.

1. ∗ 基金项目: xxxx

   Foundation item:xxxx

   收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos在线出版时间: 0000-00-00

   CNKI在线出版时间: 0000-00-00 [↑](#footnote-ref-2)