**ROP-Hunt:在应用程序中检测返回导向编程攻击**

Lu Si, Jie Yu, Lei Luo, Jun Ma, Qingbo Wu, Shasha Li

国防科技大学 计算机学院 中国长沙 410073

{lusi,jackyu,luolei,majun,wuqingbo}@ubuntukylin.com,shashali@nudt.edu.cn

**摘要** 返回导向编程(ROP) 是一种新型漏洞利用技术，该技术通过重复使用已有的小代码序列(gadget)构造出一条gadget链，从而执行任意的非法操作。尽管很多防御机制已经被提出，但是一些新的ROP攻击的变种能够轻易绕过这些防御机制。在本文中，我们将介绍一个新工具ROP-Hunt，它能依据正常程序与恶意ROP代码间的差异来防御ROP攻击。ROP-Hunt利用插桩检测技术并在程序runtime检测ROP攻击。在我们的实验中，ROP-Hunt可以在众多实例程序中检测出所有类型的ROP攻击。我们使用了几个原版的SPEC2006基准来测试ROP-Hunt的性能，结果表明它具有零误报率和可接受的系统开销。

**关键词：**返回导向编程；缓冲区溢出；检测；代码复用攻击；二进制插桩检测

# **一、引言**

由于数据执行保护（DEP）[1]的广泛采用，确保了内存中的所有可写页面都是不可执行的，因此攻击者很难将被劫持的控制流重定向至他们自行注入的恶意代码。为了绕过DEP机制，代码复用攻击（CRA）被提出并成为了攻击者们的利器。攻击者们不再注入代码，而是通过复用被攻击的漏洞进程中的现有指令来构造恶意行为。Return-into-libc技术[37]是代码复用攻击的一种简单应用，攻击者利用缓冲区溢出漏洞，将位于栈中的返回地址覆写为攻击者挑选出的将要被执行的库函数地址。传统的return-into-libc攻击利用libc函数，并不支持在受害计算机上的执行任意操作。

返回导向编程（ROP）是另一种代码复用攻击技术，它执行称为gadget的短指令序列，而不是执行一整个函数。它最初由Shacham[35]提出并应用于x86平台，随后被拓展到其他体系结构[13,16,23,26]。ROP已被证明可实现图灵完备计算[36]。此外，一些允许攻击者使用ROP自动构造任意恶意程序的工具已被开发出[22,24,33,34]。

在过去几年中，许多用于减缓基于ROP攻击的软硬件防御方法已被提出。例如：如果连续执行以*ret*指令为结尾的小指令序列，DROP [17]和DynIMA [20]将触发警报。ROPdefender [21]则是维护一个影子栈，并验证所有返回地址。李等人[27]提出了一个用于x86平台的编译器，它能避免生成可用作恶意返回指令的“0xc3”字节。此外，它使用间接调用机制替换了预期的调用和返回指令。但是，这些机制只关注了以返回指令为结尾的gadget，并不能防御其他类型，即不以返回指令为结尾的gadget的类ROP攻击。CFLocking [11]和G-Free [30]旨在防御所有类型的ROP攻击，但它们需要源代码，对于实际的终端用户而言，这些源代码通常难以取得的。KBouncer [32]涵盖了所有ROP攻击类型，且不需要辅助信息，并实现了不错的runtime效率。然而，它只是监视目标关键路径上应用程序的执行流，例如系统API，如此，它不可避免地漏掉了那些不使用这些路径的ROP攻击。

ROP攻击将gadget链接在一起，以执行复杂的操作，它具有如下特征：gadget长度很短；连续的gadget并不在同一个例程中；它们总在某一时刻执行系统调用。基于这些特征，我们设计并实现了一个名为ROP-Hunt的工具，该工具通过检查程序执行的行为是否与这些特征匹配，从而对所有类型的ROP攻击进行动态检测。在ROP-Hunt中，基于危害程度，我们将ROP报告分为两类：*警告*和*攻击*。

总的来说，我们工作的主要贡献是：

* 统计分析了大量正常的应用程序和最新的ROP恶意代码，并提取了ROP攻击的特征。
* 提出了一种新方法，可以在不访问源代码的情况下保护传统应用程序免受所有类型的ROP攻击。
* 在x86框架的Linux平台上，设计并实现一个样例，即ROP-Hunt，并评估了其安全有效性和性能开销。

本文的其余章节安排如下：在第二、三章中，我们介绍ROP攻击并分析其特征。ROP-Hunt的设计和实现在第四章中介绍。第五章和第六章，分别讨论了参数选取和一种特殊的延迟gadget。第七章介绍ROP-Hunt的安全性和性能评估。第八章研究ROP-Hunt的局限性。最后，我们在第九章总结全文并讨论有待完成的工作。

# **二、ROP攻击**

在不向程序地址空间注入新代码的情况下，ROP攻击由称为gadget的短指令序列组成。每个gadget执行一些小的计算操作（例如：将两个寄存器的值相加或将某个值加载到内存）并以返回指令为结尾。我们可以将这些gadget链接在一起并通过向栈中写入适当的参数，实现将控制流从一个gadget转至另一个gadget。

图1说明了一般的ROP攻击流程。第一步，攻击者利用漏洞程序的如缓冲区溢出这类的与内存相关的漏洞，将堆栈指针（ESP）移至第一个返回地址所在位置。例如，Aleph在文章[31] 中，通过栈溢出覆写了函数返回地址。由于原返回地址在其所在位置被返回地址1覆盖，ESP的值将自动改为此点。第二步，栈中的返回地址1被pop出栈，程序执行流将被重定向至第一个gadget。这个gadget以另一个返回指令为终止，该返回指令将栈中的返回地址2 pop出栈（第三歩），并将程序执行流重定向至下一个gadget（第四歩）。每个gadget就通过这种方式逐个执行，直到攻击者达到目标为止。



图1 一般的ROP攻击

最近，一些不使用*ret*指令的ROP攻击新变种被提出。Checkoway等人[15]发现可以通过搜寻尾随有间接跳转的pop指令（例如*pop edx; jmp [edx]*）来进行返回导向编程。这种指令序列的行为类似于返回指令，亦可用于将gadget链接在一起。

跳转导向编程（Jump-Oriented Programming, JOP）[12]是ROP攻击的另一种变种，它使用寄存器间接跳转代替了返回指令。JOP使用调度程序表来保存每个gadget的地址。每个gadget对应一个调度程序。调度程序是一段可以控制程序控制流的指令序列。调度程序用作虚拟程序计数器(PC)，将程序控制流转换为调度表中存储的地址条目，其中这些地址是特殊的、具有跳转导向功能的gadget的地址。在这些gadget的结尾，攻击者通过间接跳使程序控制流回归至调度程序。随后，调度程序将指针指向下一个gadget。一个简单的调度程序如下：*add edx,4; jmp [edx]*。

调用导向编程（Call Oriented Programming, COP）[14]由Nicholas Carlini和David Wagner于2014年提出。攻击者用以间接调用指令为结尾的gadget代替以返回指令为结尾的gadget。COP攻击不需要调度程序，它通过依次将内存间接位置指向下一个gadget的方法，来将gadget链接在一起。



(a) 传统shellcode (b)ROP恶意代码

图2 简单混合式ROP攻击

为了绕过现有的保护机制，攻击者更喜欢使用组合型gadget。图2展示了一个仅由4个gadget构成的非常简单的混合式ROP攻击，它是由传统shellcode[3]派生出的，在x86架构下，用于关闭正在运行的进程。为方便起见，我们使用系统调用*exit(n)*（n表示非零整数）代替*exit(0)*。其中，寄存器*eax*中存储系统调用号，*ebx*中存储参数。由于DROP[17]和DynIMA [20]只检测基于*ret*的连续的gadget，攻击者可以利用上述简单的ROP恶意代码来绕过这两种防御机制。

# **三、ROP攻击特征**

ROP攻击检测的关键是找出ROP恶意代码和普通程序代码间的差异。ROP中的一个重要因素是gadget的长度。 在文章[20]中，研究者们发现ROP攻击中使用的指令序列长度为2到5个指令。DROP[17]也指出gadget中的指令数不超过5条。Kayaalp等人[25]从libc标准库中提取了所有gadget并研究了其平均长度。研究结果表明，随着gadget长度的增加，副作用的数量呈线性增长，使得gadget越来越难以被利用。

在现有的检测机制中还考虑了一些其他因素。如果三个小指令序列一个接一个地被执行，DynIMA[20]将报告这是一次ROP攻击。Fan Yao等人[38]发现很少有两地址相离较近的gadget存在。

基于编写ROP恶意代码的经验，我们发现了它的另外两个特性。其一，无论是以跳转还是调用指令为结尾，连续的gadget都不会位于同一个例程中。其二，shellcode总是利用系统调用将控制流从用户态转移至内核态。

在计算机编程中，例程是一串代码序列，用于在程序执行期间被重复地调用和使用，在高级语言中，许多常用的例程被打包为函数。在传统的ROP攻击中，每个gadget都以返回指令为结尾，除了递归返回，在大多数情况下它们不在同一个例程中。ROPGadget[8]是一个开源的gadget搜索工具，我们用它从libc中提取gadget，并尝试用文章[12]中提出的算法构造几段JOP恶意代码。我们发现在相同例程中的连续的gadget极难被利用。

ROP恶意代码是由shellcode派生而出的，因此gadget的构建也是基于传统shellcode的。我们分析了文章[5]中全部247个shellcode，发现其中212个至少调用了一次系统调用。其他的shellcode为了绕过特征检测机制，使用了加密或自修改payload的方式，不直接使用*“int 0x80”*，从而避免敏感数据（如cd 80）的出现。但无论如何，为了获得更高的权限，他们终将在runtime调用系统调用。文章[2]中调用了内核*vsyscall*函数，该函数使用*sysenter*指令，将控制流从在第3特权级运行的用户态转移至操作系统。*sysenter*指令提供了对内核的快速访问，也可被视为另一种系统调用。

综上，我们认为（1）gadget的大小；（2）系统调用的执行；（3）连续的候选gadget不在同一例程中，这三点可以作为ROP攻击最具代表性的特征。我们基于ROP恶意代码和普通程序之间存在的这三种差异，开发了一个名为ROP-Hunt的工具，它通过检查程序运行轨迹是否偏离正常运行路径来动态检测ROP攻击。我们将在下一章中展示ROP-Hunt的设计。

# **四、ROP-Hunt设计与实现**

基于ROP攻击的特征，我们提出了能够有效地检测ROP攻击的方法。由于我们假设的前提是无法访问源代码，因此我们使用了插桩检测技术，该技术允许向程序添加额外的代码以观察、调试其行为[29]。

## **4.1假设及定义**

在本文中，我们定义gadget中的指令数为*G\_size*。*G\_size*大于阈值*T0*的gadget为候选gadget。连续候选gadget序列的长度定义为*S\_length*，*Max(S\_length)*表示*S\_length*的最大值。

为了模拟真实环境，我们做出如下假设：

1. 我们假设底层系统支持DEP[1]模型，该模型禁止了可执行内存的写权限。在这种情况下，基于代码注入的攻击是不可行的。现代处理器和操作系统默认启用DEP。
2. 我们假设攻击者能够通过缓冲区溢出[19,31,39]、字符串格式化攻击或非本地跳转缓冲区（使用*setjmp*和*longjmp* [4]）来发起ROP攻击。
3. 我们假设攻击者在用户态下进行操作，并且利用漏洞发起的攻击不会导致权限提升。
4. 我们假设我们无法访问源代码。

## **4.2系统概述**

图3展示了ROP-Hunt的流程图。ROP-Hunt根据我们在第三章中分析的ROP的特征动态地监视程序，并拦截系统调用指令和三个控制流敏感指令：*call*，*jmp*和*ret*。ROP报告分为两类：*警告*和*攻击*。*警告*表示该进程存在严重的被ROP攻击的风险，但是由于它没有调用系统调用来访问底层系统源，我们认为攻击者无法利用其进行任何有意义的攻击。如果统计值越过阈值且有调用系统被调用，ROP-Hunt将终止进程并报告这是一次*攻击*。

* *报告警告*：当ROP-Hunt识别到这三种指令（调用、间接跳转和返回指令）时，它会检查指令序列的长度是否大于阈值*T0*。若非，则提取目标地址和当前指令地址（尤应重视*ret*指令，因为*ret*指令会将目标地址pop出栈）。随后，ROP-Hunt会检查这两个地址是否位于同一例程中。若非，我们便将该指令序列记录为候选gadget。接下来，我们统计连续候选gadget的最大长度*S\_length*。如果*S\_length*小于或等于阈值*T1*，我们便将潜在攻击标志置为*True*并发出*警告*。
* *报告攻击*：系统调用是将控制流从用户空间转移至内核空间的唯一途径。故当识别出系统调用指令时，ROP-Hunt会检查潜在攻击标志是否为*True*。若是，则ROP-Hunt将报告这是一次*攻击*并终止该进程。

## **4.3实现细则**

为了证明我们方法的有效性并对其进行性能评估，我们在x86框架32位版的Ubuntu 14.04（内核版本3.19）上开发了一个样例。我们的样例，即ROP-Hunt，使用了二进制插桩检测框架Pin[28]（版本2.14）。

我们将ROP-Hunt直接整合到Pin框架中。Pin是程序插桩检测工具，它能够检测所有实际执行的指令。Pin有两种工作模式：探针模式和即时（Just-In-Time, JIT）模式。在JIT模式下，Pin能够在处理器执行每条指令前将其拦截（包括那些令编程人员们 “意想不到”的指令）。



图3 ROP-Hunt的工作流程

要想在runtime检测二进制文件，我们必须确定两点：其一，代码应该被插入在什么位置；其二，在插入点应该执行什么代码。Pin提供了名为Pintool的各种插桩检测工具，Pintool由C/C++语言编写。在使用Pin提供的丰富API的同时，用户还可以自定义插桩检测代码。我们设计并实现了自己的Pintool来在Pin框架下检测ROP攻击。

Runtime系统的总体架构如图4所示。我们的架构由Pin框架和Pintool ROP-Hunt组成。Pin是对二进制程序进行即时(JIT)插桩检测的引擎。Pin框架由三部分组成：虚拟机（VM）、代码缓存和供Pintool调用的API。其中，虚拟机包含JIT编译器、模拟器和调度程序。当程序开始运行时，各条指令先经JIT编译并检测，再交由调度程序激活并执行。经过编译的指令存储在代码缓存中，以便在多次调用代码段时降低性能开销。模拟器用于解释那些无法被直接执行的指令。

我们的Pintool，即ROP-Hunt，由记录单元和检测单元两部分组成：检测单元包含插桩检测例程和分析例程，检测单元利用各种插桩检测API与Pin进行通信；记录单元仅用于存储runtime中的一些统计值。

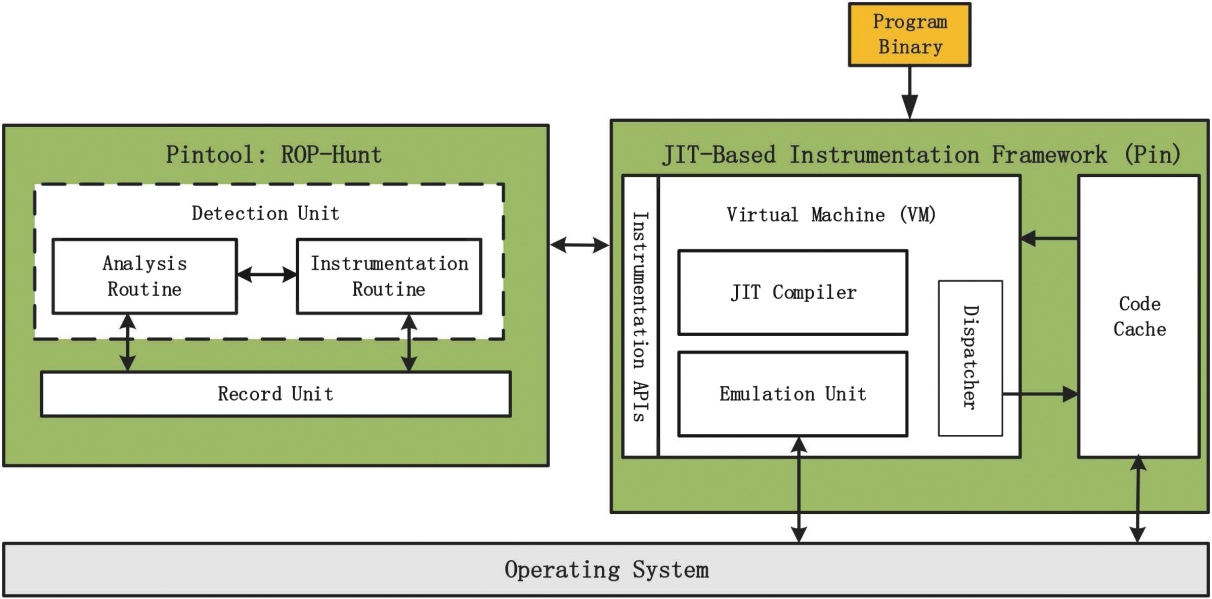


图4 在Pin框架下实现的ROP-Hunt

## **4.4插桩检测例程和分析例程**

据4.1节中所述，识别指令类型是关键点之一。ROP-Hunt的插桩检测例程通过Pin API提供的检查函数*INS\_IsSyscall(INS ins)*和*INS\_IsSysenter(INS ins)*来确定当前指令是否为*syscall*或*sysenter*，并通过*INS\_IsIndirectBranchOrCall(INS ins)*来确定当前指令是否为跳转指令。如果当前指令是间接跳转、调用或返回指令，我们将调用分析函数来提取该指令地址和目标地址。

ROP-Hunt为每个例程分配一个ID，该ID全局唯一，即同一个ID不会出现在两个内存镜像中。如果有同名例程存在于两个不同的内存镜像中（即它们在不同的地址中），那么每个例程将被分配不同的ID。如果内存镜像被卸载然后重载，那么其中的例程ID极有可能与之前的ID不同。ROP-Hunt使用*PIN\_InitSymbols()*函数来初始化符号表并从二进制文件中读取符号。由此，我们可以通过地址来获取例程ID。

记录单元分别为每个线程分配数据空间。我们使用Pin API中的线程本地存储（TLS）来避免一个线程访问另一个线程记录的情况。

# **五、参数选择**

能够代表ROP特征的因素有两个：gadget中指令数（*G\_size*）和连续候选gadget序列长度（*S\_length*），我们必须确定这两个因素的阈值。

Gadget大小的阈值（*T0*）会影响检测的准确度，较大的阈值通常会导致误报率较高。为寻找*T0*，我们使用两个知名的gadget搜索工具ROPGadget[8]和Ropper[9]测量了大量正常程序中的gadget大小。这些程序包括/bin和/usr/bin目录下的22种Linux常用工具（例如ls、grep和find），以及3个大型二进制文件（Apache Web服务器httpd 2.4.20、mysql 5.6和python 2.7）。我们总共收集了282341个gadget，其中125605个来自ROPGadget，156736个来自Ropper。如图5所示，最大的gadget大小为10，且近乎所有gadget的大小都小于8。在Ropper收集的gadget中，最大的大小为6。我们还分析了实际ROP攻击中的ROP恶意代码，并没有发现大小超过6的gadget。根据上述结果，我们可以安全地选择7作为gadget大小的阈值（*T0*）。如果指令序列的长度不超过7，ROP-Hunt将会视其为候选gadget。

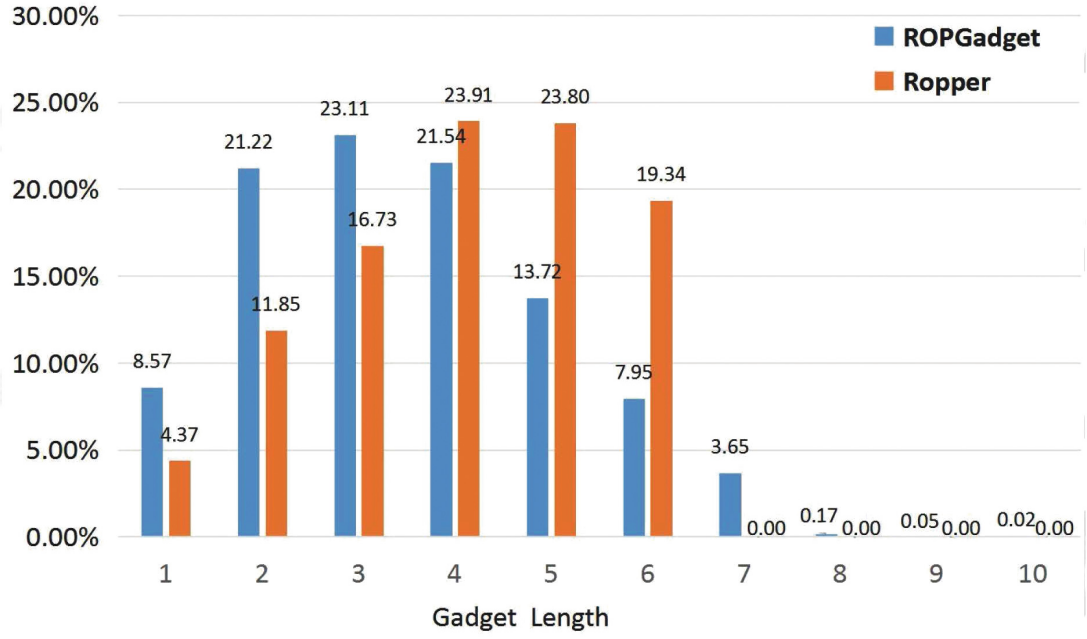


图5 Gadget大小的测量结果

在ROP攻击中，攻击者将少量gadget链接在一起以一次完成预期的恶意操作。为了构造一次系统调用操作，攻击者至少要用3个gadget才能将正确的参数放入参数寄存器并跳转至系统调用程序入口。我们认为，攻击者只用3个或更少的gadget无法进行任何有意义的攻击。所以，我们将阈值*T1*设置为3，也就是说，ROP-Hunt会检查连续的gadget是否超过3个。

# **六、延迟gadget**

正因为我们假设可用的gadget很短，我们才能够从正常程序中区分出攻击代码，ROP-Hunt才能起作用。然而，高明的攻击者为了在攻击过程中绕过检测机制，可能会使用长gadget并容忍其带来的一些副作用。Mehmet Kyaalp等人[25]引入了延迟gadget，其长度足以破坏长度特征检测器中的gadget计数器。他们构造一个函数调用，该调用将导致大量指令被执行。按照惯例，当函数返回时，许多寄存器（例如*ebx、esi、edi、esp*和*ebp）*的值将会被保存。如此说来，延迟gadget的副作用可被大大降低。

延迟gadget的目的是绕过基于长度特征检测器的检测，它既不执行任何的攻击代码，也不会破坏攻击所需的机器状态。仅通过延迟gadget进行ROP攻击是不可能的。因此，当上一个gadget以调用某函数的调用指令为结尾时，若该gadget的大小超过了阈值*T0*，ROP-Hunt将忽略此gadget，并不会影响长度特征检测器。但若其大小没有超过阈值*T0*，计数器仍会加1。

# **七、评估**

在本章，我们将评估ROP-Hunt的安全有效性和性能开销。所有实验均在具有以下参数的计算机上进行：Intel Core i3 2370M CPU、4 GB RAM、32位Ubuntu（内核版本3.19）。在安全性评估方面，我们通过两次真实的ROP攻击和一个可由输入过长参数触发简单栈缓冲区溢出的小程序来验证我们的检测方法。在性能评估方面，我们在实验中使用了18个C和C++ SPEC CPU2006[10]基准，这些测试基准由gcc-4.8.3编译器编译。

## **7.1安全评估**

首次测试中，我们通过攻击两个实际程序：Hex-editer（2.0.20）和PHP（5.3.6）来评估ROP-Hunt的有效性。上述两段ROP恶意代码模板可以在网站[6,7]上找到。在PHP的漏洞利用过程中，我们向UNIX套接字输入了一个超长路径名，从而触发了缓冲区溢出，然后将控制流转移至ROP payload。Payload中含有31个连续的gadget，其中最大的gadget包含7条指令（没有超过阈值*T0*）。因此，ROP-Hunt发出*警告*并将潜在攻击标志置为*True*。连续gadget序列的最后一个gadget打算调用系统调用来直接执行/bin/sh。因此，ROP-Hunt报告了这是一次*攻击*并终止了该进程。

为了进一步评估ROP-Hunt的检测能力，我们使用了一个具有strcpy漏洞的简单目标程序（文章[31]中的示例）。该程序由gcc-4.8.4编译，链接的库为glibc-2.3.5。我们使用ROPGadget[8]分析该程序并生成了可用的gadget。随后我们手动挑选候选gadget，对Shell-Storm Linux shellcode仓库[5]中的30个典型shellcode进行了重构，重构的shellcode均由以*ret、jmp*或*call*指令为结尾gadget组成。由于副作用的存在，超过7条指令的gadget极难被利用。最简单的攻击需要4个gadget（大于*T1*）。正如我们在第三章中分析的那样，所有shellcode均利用系统调用来完成攻击。实验结果表明，ROP-Hunt可以无误报地检测出上述所有ROP攻击。

## **7.2性能开销**

我们选择基准工具SPEC CPU2006的测试套件[10]来测试ROP-Hunt的性能。具体来说就是在ROP-Hunt启用和关闭的情况下分别运行测试套件。测试结果如图6所示。

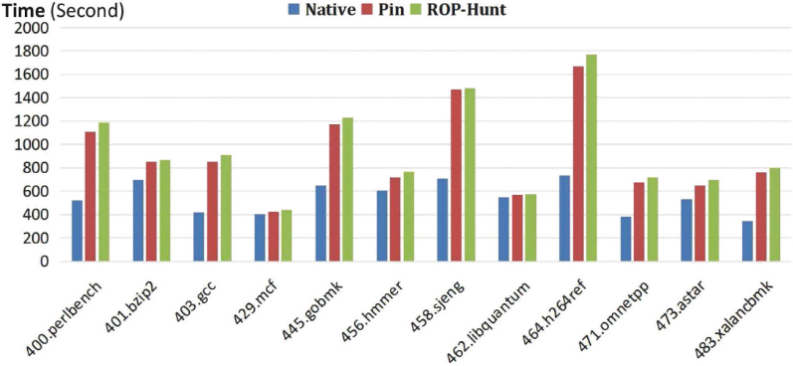


图6. SPEC CPU2006基准测试结果

在ROP-Hunt保护下运行的应用平均放缓了1.75倍。基准测试的放缓范围为1.05倍至2.41倍。我们将ROP-Hunt与其他基于插桩检测技术的ROP检测器进行了比较。根据文章[17,21]中的结果，在ROPdefender和DROP下运行的应用程序分别放缓2.17倍和5.3倍。文章[18]中的方法导致应用程序平均放缓3.5倍。

结果表明Pin框架本身也会带来1.66倍的平均放缓。我们确信，随着Pin框架的不断优化，ROP-Hunt的性能也将不断提高。

# **八、讨论**

我们设计并实现了用于在runtime检测ROP攻击的工具ROP-Hunt。目前ROP-Hunt基于动态二进制插桩检测框架Pin。尽管ROP-Hunt可以有效地检测ROP攻击，但仍存在以下局限性。其一，ROP-Hunt仅检测x86架构上的ROP恶意代码，然而，ROP技术并不受体系结构的限制，攻击者可重写恶意代码，并将之移植至其他体系结构。不过，诚然我们的方法也可以部署到其他架构。其二，ROP-Hunt检测到ROP攻击的前提是假设全部ROP恶意代码均满足第五章中讨论的阈值。虽然在我们定的假设范围之外进行ROP攻击是极为困难的，但是理论上仍存在这种可能性。其三，ROP-Hunt是在基于JIT的二进制插桩检测框架Pin下实现的，并造成了1.75倍的平均放缓。这样的性能开销，对于某些对时间要求严格的应用程序而言，也许是不能被接受的。

# **九、结论**

ROP是一种可绕过现有安全机制的强有力的漏洞利用技术。在本文中，我们研究并提取了ROP恶意代码的特征。基于识别ROP恶意代码在执行过程中表现出的独特固有属性，我们提出了一种新的实用方法来防御ROP攻击，并且无需访问受保护程序的源代码。实验结果表明，我们的样例ROP-Hunt成功检测出了所有ROP攻击，并且没有误报。ROP-Hunt利用插桩检测技术，造成了1.75倍的runtime开销，这额外开销与同类的基于插桩检测技术的ROP检测工具相比，较为可观。我们计划将该样例移植至其他架构，这将是我们未来的工作内容之一。

**致谢：**感谢匿名评审们建设性的意见，在这些意见的指导下，本文才得以定稿。感谢国防科技大学提供了必要的实验环境，有了这样的实验环境，本文才得以完成。本项工作在国家自然科学基金委员会的大力支持下开展，资助编号：61103015,61303191,61402504,61303190。

# **参考资料**

1. Data execution prevention. http://support.microsoft.com/kb/875352/EN-US
2. Linux/x86 - /bin/sh sysenter Opcode Array Payload. http://shell-storm.org/ shellcode/files/shellcode-236.php
3. Linux/x86 - sys exit(0). http://shell-storm.org/shellcode/files/shellcode-623.php
4. Setjmp - set jump point for a non-local goto. http://pubs.opengroup.org/onlinepubs/009695399/functions/setjmp.html
5. Shellcodes database for study cases. http://shell-storm.org/shellcode/
6. HT Editor 2.0.20 Buffer Overflow (ROP PoC). http://www.exploit-db.com/exploits/22683/
7. PHP 5.3.6 Buffer Overflow PoC. http://www.exploit-db.com/exploits/17486
8. ROPgadget - Gadgets finder and auto-roper. http://shell-storm.org/project/ROPgadget/
9. ROPPER - ROP GADGET FINDER AND BINARY INFORMATION TOOL.https://scoding.de/ropper/
10. Standard Performance Evaluation Corporation, SPEC CPU2006 Benchmarks. http://www.spec.org/osg/cpu2006/
11. Bletsch, T., Jiang, X., Freeh, V.: Mitigating code-reuse attacks with control-flow locking. In: Proceedings of the 27th Annual Computer Security Applications Conference, pp. 353-362. ACM (2011)
12. Bletsch, T., Jiang, X., Freeh, V.W., Liang, Z.: Jump-oriented programming: a new class of code-reuse attack. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, pp. 30-40. ACM (2011)
13. Buchanan, E., Roemer, R., Shacham, H., Savage, S.: When good instructions go bad: generalizing return-oriented programming to risc. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 27-38. ACM (2008)
14. Carlini, N., Wagner, D.: ROP is still dangerous: breaking modern defenses. In: 23rd USENIX Security Symposium (USENIX Security 2014), pp. 385-399 (2014)
15. Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A.R., Shacham, H., Winandy, M.: Return-oriented programming without returns. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 559-572. ACM (2010)
16. Checkoway, S., Feldman, A.J., Kantor, B., Halderman, J.A., Felten, E.W., Shacham, H.: Can DREs provide long-lasting security? The case of return-oriented programming and the AVC advantage. In: EVT/WOTE 2009 (2009)
17. Chen, P., Xiao, H., Shen, X., Yin, X., Mao, B., Xie, L.: DROP: detecting returnoriented programming malicious code. In: Prakash, A., Sen Gupta, I. (eds.) ICISS 2009. LNCS, vol. 5905, pp. 163-177. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10772-6 13
18. Chen, P., Xing, X., Han, H., Mao, B., Xie, L.: Efficient detection of the returnoriented programming malicious code. In: Jha, S., Mathuria, A. (eds.) ICISS 2010. LNCS, vol. 6503, pp. 140-155. Springer, Heidelberg (2010). doi:10.1007/978-3-642-17714-9 11
19. Chen, S., Li, Z., Huang, Y., Xing, J.: Sat-based technique to detect buffer overflows in c source codes. J. Tsinghua Univ. (Science and Technology), S2 (2009)
20. Davi, L., Sadeghi, A.R., Winandy, M.: Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks. In: Proceedings of the 2009 ACM Workshop on Scalable Trusted Computing, pp. 49-54. ACM (2009)
21. Davi, L., Sadeghi, A.R., Winandy, M.: Ropdefender: adetection tool to defend against return-oriented programming attacks. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, pp. 40-51. ACM (2011)
22. Dullien, T., Kornau, T., Weinmann, R.P.: A framework for automated architecture-independent gadget search. In: WOOT (2010)
23. Francillon, A., Castelluccia, C.: Code injection attacks on Harvard-architecture devices. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 15-26. ACM (2008)
24. Hund, R., Holz, T., Freiling, F.C.: Return-oriented rootkits: bypassing kernel code integrity protection mechanisms. In: USENIX Security Symposium, pp. 383-398 (2009)
25. Kayaalp, M., Schmitt, T., Nomani, J., Ponomarev, D., Abu-Ghazaleh, N.: SCRAP: architecture for signature-based protection from code reuse attacks. In: 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013), pp. 258-269. IEEE (2013)
26. Kornau, T.: Return oriented programming for the ARM architecture. Ph.D. thesis, Masters thesis, Ruhr-Universität Bochum (2010)
27. Li, J., Wang, Z., Jiang, X., Grace, M., Bahram, S.: Defeating return-oriented rootkits with return-less kernels. In: Proceedings of the 5th European Conference on Computer Systems, pp. 195-208. ACM (2010)
28. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: ACM Sigplan Notices, vol. 40, pp. 190-200. ACM (2005)
29. Nethercote, N.: Dynamic binary analysis and instrumentation (2004). http://valgrind.org/docs/phd2004.pdf
30. Onarlioglu, K., Bilge, L., Lanzi, A., Balzarotti, D., Kirda, E.: G-free: defeating return-oriented programming through gadget-less binaries. In: Proceedings of the 26th Annual Computer Security Applications Conference, pp. 49-58. ACM (2010)
31. One, A.: Smashing the stack for fun and profit. Phrack Mag. 7(49), 14-16 (1996)
32. Pappas, V., Polychronakis, M., Keromytis, A.D.: Transparent ROP exploit mitigation using indirect branch tracing. In: Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 2013), pp. 447-462 (2013)
33. Roemer, R.G.: Finding the bad in good code: automated return-oriented programming exploit discovery (2009)
34. Schwartz, E.J., Avgerinos, T., Brumley, D.: Q: Exploit hardening made easy. In: USENIX Security Symposium, pp. 25-41 (2011)
35. Shacham, H.: The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 552-561. ACM (2007)
36. Tran, M., Etheridge, M., Bletsch, T., Jiang, X., Freeh, V., Ning, P.: On the expressiveness of return-into-libc attacks. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 121-141. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23644-0 7
37. Wojtczuk, R.: The advanced return-into-lib(c) exploits: PaX case study. Phrack Mag. 0x0b(0x3a), Phile# 0x04 of 0x0e (2001)
38. Yao, F., Chen, J., Venkataramani, G.: Jop-alarm: detecting jump-oriented programming-based anomalies in applications. In: 2013 IEEE 31st International Conference on Computer Design (ICCD), pp. 467-470. IEEE (2013)
39. Zhang, M., Luo, J.: Pointer analysis algorithm in static buffer overflow analysis. Comput. Eng. 31(18), 41-43 (2005)