**ROP-Hunt：在应用程序中检测返回导向编程(ROP)攻击**

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机制防御。

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**(instrumentation)技术并在程序运行时检测ROP攻击。在我们的实验中，ROP-Hunt可以在众多真实例程中检测出所有类型的ROP攻击。我们使用了几个原版的SPEC2006标准来测试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

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到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gadgets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，并实现了不错的运行时效率。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攻击组合gadget成链，以执行复杂的操作，它具有如下特征：gadget长度很短；连续的gadget并不在同一个程序中，并且它们都在某个地方执行系统调用。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，设计并实现一台样机，即ROP-Hunt，并评估了其安全有效性和性能开销。

本文的其余部分安排如下：在第二、三部分中，我们描述了ROP攻击并分析了它们的特征。ROP-Hunt的设计和实现在第四部分介绍。第五部分和第六部分，分别讨论了参数选取和特殊的延迟gadget。第七部分介绍了ROP-Hunt的安全性和性能评估。第八部分研究了它的局限性。最后，我们在第九部分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，并通过在堆栈上写入适当的值，使控制流从一个gadget转至另一个gadget。

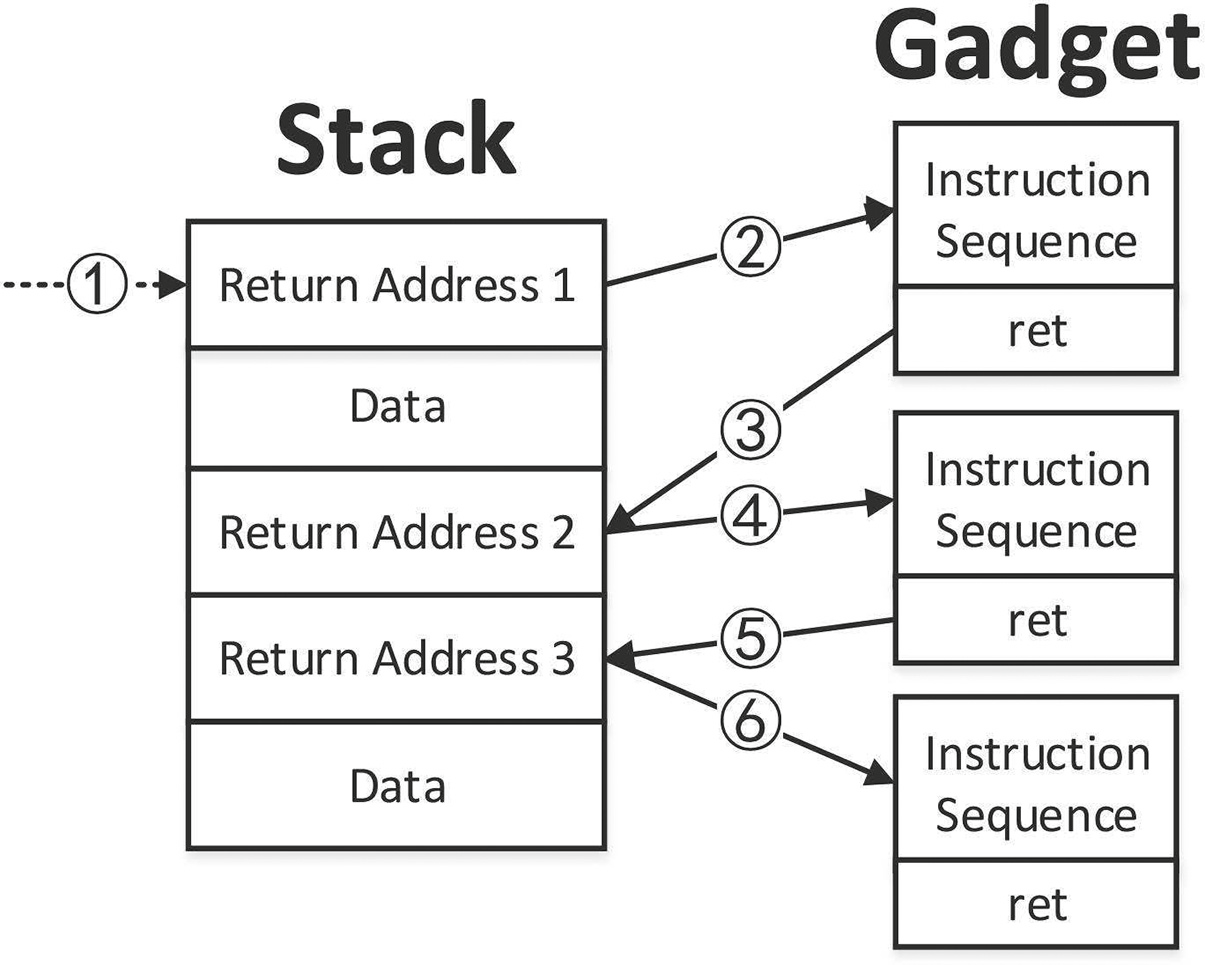


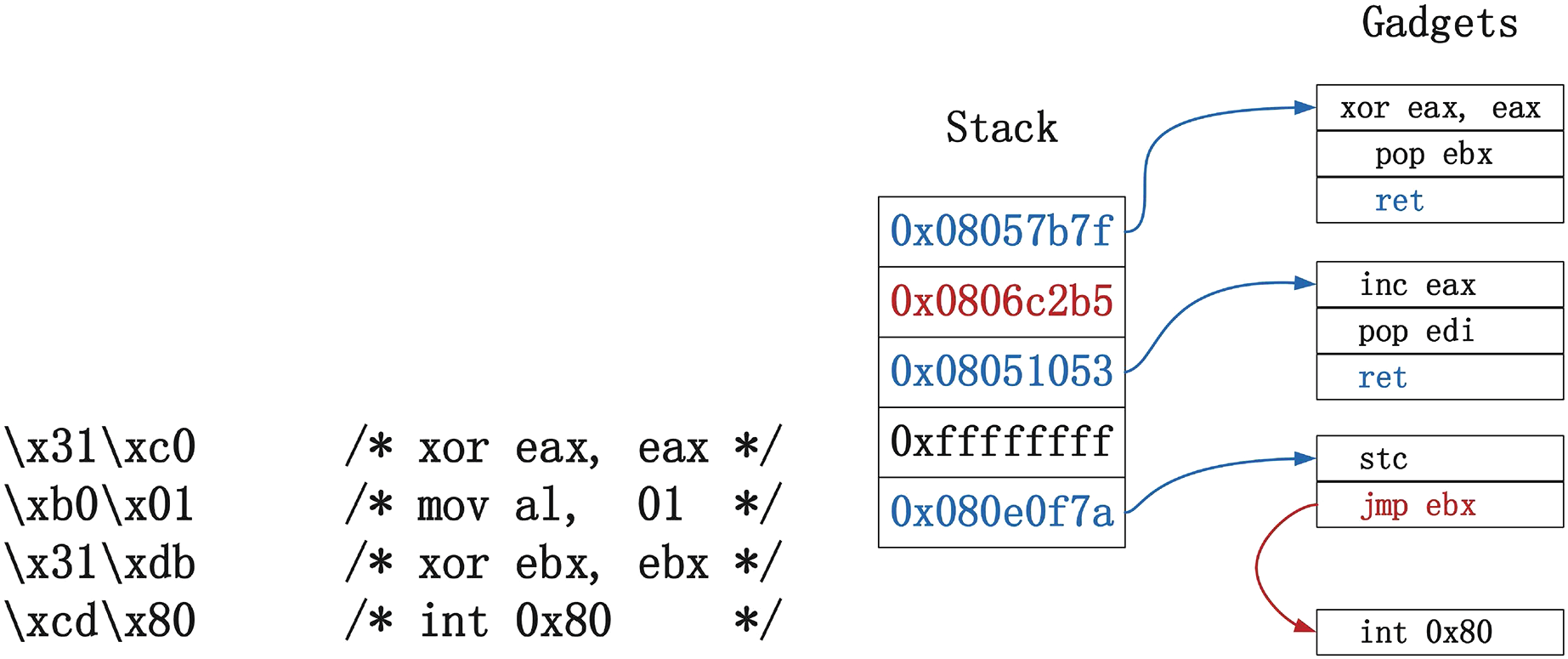
图1 一般的ROP攻击

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[31]这篇文章中，通过栈溢出覆写了函数返回地址。由于返回地址1覆盖了原返回地址所在位置，ESP的值将自动改为此点。第二步，通过pop栈中的返回地址1，程序执行流被重定向至第一个gadget。该gadget以另一个返回指令为终止，该指令pop栈中的返回地址2（第三歩）并将程序执行流重定向至下一个gadget（第四歩）。每个gadget通过这种方式逐个执行，直到攻击者达到目标。

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，亦可用作gadget的链接。

跳转导向编程（Jump-Oriented Programming, JOP）[12]是ROP攻击的另一种变体，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，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年推出。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，它通过依次地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]只检测连续的基于ret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。