一种基于DBI的ROP攻击检测

黄志军 郑 滔

(南京大学软件学院 南京 210093)

摘 要 随着Return-Oriented Programming(ROP)思想的提出,程序安全面临新的挑战。ROP攻击操作粒度细致,特征隐蔽,构造精巧,静态特征稀少,因此传统的安全防御措施很难对之有效。在这种背景下,利用执行时的动态特征去识别、防御ROP攻击变得非常重要。Dynamic Binary Instrumentation(DBI)技术的引入,为ROP攻击动态特性分析提供了有力的支持。介绍一种利用DBI分析技术识别ROP攻击序列的方法,即通过识别恶意的程序执行流,约束库函数的调用规范,来检测ROP攻击。与此同时,还设计了一套可扩展的程序防御框架,用于检测程序的扩展,由此说明该检测工具的通用性与可扩展性。

关键词 ROP,动态二进制插桩,程序安全,特征检测,图灵完备性,程序控制流

中图法分类号 TP31 **文献标识码** A

ROP Attack Detecting Method Based on DBI

HUANG Zhi-jun ZHENG Tao

(Software Institute, Nanjing University, Nanjing 210093, China)

Abstract As the promotion of the idea of return-oriented programming (ROP), programs will face many new kinds of challenges from virus programs. With fine granularity, covert virus features, deliberate and sophisticated construction and rare static characteristics, ROP attack can circumvent many traditional defending measures. Under this circumstances, it's imperative to discover the dynamic features of ROP attack program, identify its characteristics and defend it when it is executed. At this time, introducing the technology of dynamic binary instrumentation provides powerful support for dynamic analysis of ROP attack. We introduced a defending measure to ROP attack with the help of DBI technology. By identifying malicious program execution flow and restricting the call specification of libraries, we detected ROP attack. Furthermore, we designed an extensible defending framework over ROP attack to prove the generality and portability of our detect tool.

Keywords Return-oriented programming, Dynamic binary instrumentation, Programming security, Characteristic detection, Turning-complete, Control flow

1 概述

ROP思想[1,4,5]的提出,对程序安全造成了新的威胁。ROP思想是Return-Into-Libc[2,3]思想的延伸和扩展,而Return-Into-Libc是ROP思想的特化形式。通过利用已有程序中以ret指令结尾、长度为2到5条指令的短序列,构建具有一定语义表达和计算能力的gadget集合,并选取这些gadget集合进行合理串联,形成具有很强计算能力的指令序列。ROP攻击通过在已有的程序库(包括libc、驱动程序等)寻找gadget序列,构建图灵完备的攻击程序,并攻击程序的正常控制流程,以提升攻击者操作的权限,甚至能够完全获取计算机的控制权。

ROP攻击思想的精髓在于,通过对以ret指令结尾的序列的发现、研究与分析,进行合理串联,构建短指令序列;进而通过栈溢出、字符串格式化攻击等方法,将包含库中的指令序列的地址、程序变量值填入栈或堆中;然后通过短序列中的ret指令来控制程序的执行流,达到攻击的目的。

目前ROP攻击程序构造的研究已经取得了很大进步。Hovav Shacham[1]在Intel x86体系结构Linux平台上,使用libc中发现的代码片段,组成很多gadget,进而使之有图灵完备的表述能力。这些gadget能够完成以下基本操作:Load/Store、算术和逻辑运算、流程控制(无条件跳转、条件跳转、系统调用、函数调用等操作),而这些操作按照一定规则运算,能够具备图灵完备的表述能力,和利用ROP攻击技术执行Shell脚本的攻击能力。Eric Buchanan[6]等在指令集严格对齐、多寄存器、执行对齐检查的SPARC机器上,使用ROP的思想实现了同样的攻击,并利用ROP技术进行了矩阵加法运算,从而证明ROP不仅对可变长指令集有效,而且对定长指令集的机器同样有效,即对CISC和RISC指令体系结构都有效。Ralf Hund等[7]设计了一套在Windows平台下使用已有

到稿日期:2011-11-11 返修日期:2012-02-29 本文受国家自然科学基金NSFC(61073027,61105069)资助。

黄志军(1987—),男,硕士生,主要研究方向为程序安全,E-mail:MG1032004@software.nju.edu.cn;郑 滔(1966—),男,教授,主要研究方向为程序安全、软件工程。

DLL 程序半自动化构建 gadget 序列、实施 Rootkit 攻击的程序。Thomas Dullien[8] 定义了一套中间元信息语言和框架，以解决跨平台 gadget 自动构建的问题，向 ROP 程序的自动化迈出了重要的一步。Tim Kornau[9] 和 Lucas Davi[10] 实现了在 ARM 体系结构中的 ROP 攻击。Stephen Checkowayy[11] 等实现了非 ret 指令结尾的 ROP 攻击，从而扩大了 ROP 攻击的候选代码集的范围，同时实现了在 Linux 和 Google Android ARM 平台上的 ROP 攻击。

ROP 攻击的商业化已经实现，ROP 攻击的危害不容小觑。之前很多针对 Return-Into-Libc 的方法对 ROP 都将失效，其中最为著名的是 $W \oplus X$ 技术。随着 $W \oplus X$ 等技术的引入，传统的代码注入攻击几乎被消除，Return-Into-Libc 攻击受到很大程度的抑制，该技术已在 Linux PAX[12,13] 项目中得到实现，但是 $W \oplus X$ 技术对于 ROP 攻击却束手无策，因为 ROP 技术不会向栈或堆中注入任何可执行代码。随着研究的深入，一些针对 ROP 攻击的专门的检测技术和方法应运而生，其优势显而易见，但同时也都具有自身的局限性。下面将分类进行介绍，并比较已有方法和工具的利弊。目前已知的技术有地址随机化、基于编译的防御方案、栈保护、控制流完整性检查、基于 ret 指令序列频率的检查方案、即时编译分析技术、内存着色技术、甚至硬件防护的解决方案等。

动态二进制插桩(DBI)分析技术的引入为分析程序动态执行流提供了新的途径，也为程序动态行为的检测和分析提供了基础平台。本文将分析 ROP 短序列的动态特性，利用 DBI 技术跟踪分析程序的执行流，检测 ROP 攻击。利用 Pin 工具开发出一个 ROPDetect Pin tool 来检测 ROP 攻击，并设计一套跨平台的 ROP 检测框架。

2 背景知识

2.1 ROP 攻击思想

ROP 攻击利用程序库中的代码片段来实施攻击。攻击者首先通过静态、动态代码反汇编工具来分析已有库中的指令序列；然后找出有用的、以 ret 结尾的短序列及其首地址，并合理安排栈帧的数据，构建 gadget；最后劫持程序的正常控制流，按照预先设置的序列的顺序执行恶意的流程，进而达到攻击的目的。栈帧的控制结构安排如图 1 所示。

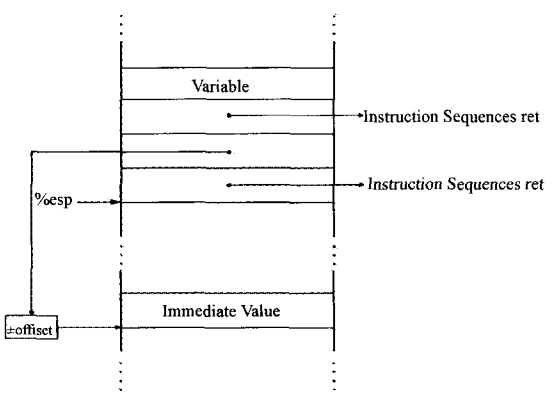

图 1 ROP 攻击栈帧布局

当正常程序的栈被攻击后，程序控制流转入 ROP 攻击程序，ROP 攻击程序的执行流程如图 2 所示。ROP 攻击程序利用被破坏的栈帧，填入 ROP 控制结构的数据和中间变量，控制程序流执行 gadget 序列，进而实现恶意的计算。

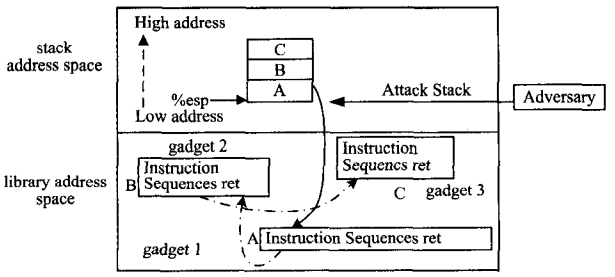

图 2 ROP 攻击流

2.2 ROP 攻击序列特性

研究发现，ROP 指令有以下特性。首先，ROP 攻击的短序列的长度较短，一般在 2 到 5 条左右。原因如下：其一，较短的有效指令序列便于使用自动化的手段去构建有效的 gadget；其二，较短的指令序列将产生较高的灵活性，且粒度小，构建 gadget 的灵活性和可行性大增，难度降低；其三，过长的指令序列在一次性完成更为复杂的操作的同时，会产生副作用(当有多条指令协作完成一个基本操作时，前一条指令存储在寄存器中的状态被破坏)，而这往往会抵消该长指令序列所带来的效果。其次，ROP 指令序列以 ret 结尾，以实现控制流的跳转，因而 ret 指令的出现频率会非常高。再者，gadget 指令序列的起始位置往往是在已有代码库函数中的中间位置，因为出于 gadget 构建的考虑，使用 Trie 树结构去搜索以 ret 结尾的指令，必然会导致短序列的起始位置处于库函数的中间位置，而不是入口位置，否则，ROP 攻击就退化为函数调用或者 Return-Into-Libc 攻击，那么 ROP 的 gadget 构建更为困难，其图灵完备性就消失，ROP 攻击也就失去意义。

ROP 攻击序列包含的是地址和程序变量，而不是指令本身。即在栈或堆中放置指向库中指令序列的地址以及需要用到的变量，从而决定了在 ROP 攻击中，栈或者堆中会出现大量的地址，这些地址指向库中以 ret 指令结尾的短序列，而且不是从库中实际的函数入口进入，而是从函数中间进入。更进一步说，从 ROP 攻击的视图观察，库中只有指令序列，而没有函数或者模块的概念。这是 ROP 攻击的一个本质的特征，我们的检测工具将利用其来检测 ROP 攻击序列。

2.3 ROP 的图灵完备特性

在可计算理论中，若一组数据操作的规则(一组指令集，编程语言，或者元胞自动机)满足任意数据按照一定的顺序可以计算出结果，则能计算出每个图灵可计算函数的计算系统称为图灵完备。那么其计算能力与一个通用图灵机等价[34]。通常，在现代高级语言中，能够完成以下基本操作集合的计算特性被称其图灵完备的，即其能够模拟通用图灵计算机的能力：1. 数据的移动、存储，如内存数据加载，寄存器的操作；2. 算术运算，如加、减、乘、除操作；3. 逻辑操作，如与、或、否等操作；4. 控制流的操作，如跳转、循环等操作；5. 函数调用；6. 系统调用。因此，实现这样一组操作的 ROP 攻击的 gadget 集合具备通用图灵机的计算能力，因而称为具有图灵完备特性。

2.4 动态二进制插桩技术

动态二进制插桩技术(DBI)能够在程序执行阶段动态监测程序的行为，而无需改动程序代码和编译过程。动态二进制插桩技术能够在不影响程序行为的情况下，提供程序动态

执行过程中的状态。动态二进制插桩技术一般通过向目标二进制代码中插入桩脚或者将二进制代码翻译成中间代码表示形式，然后在虚拟机中执行这些处理后的代码，来监视程序动态行为中的寄存器、内存、指令序列等状态。后者即是D&R[14](Disassemble-and-Resynthesize)技术，典型的实现工具是Valgrind[15,16]；前者即是C&A(Copy-and-Annotate)技术，典型的实现工具是Pin[17,18]。

3 基于DBI的解决方案

3.1 检测思想

ROP攻击的最核心的思想是利用已有库中的代码片段，通过库中以ret结尾的指令来构建完成基本操作(如Load，Store，Add等)的gadget，然后这些gadget集合满足图灵完备特性，以提高计算能力，进而实施有意义的攻击。因而gadget集合的构建和以ret结尾的短序列的来源是关键，如果没有足够类库的已有代码提供候选，ROP攻击的可行性、计算能力、攻击能力将大幅降低，那么ROP将会受到大幅抑制。因此，如果能够抑制ROP所能候选的库的范围，限制对已有代码的恶意使用，强制使用控制流和完整性检查规则，就能有效检测和防范ROP攻击。

ROP指令序列的选择会有一些明显特征。首先，出于gadget构建难度和长指令序列副作用的考量，短序列的长度较短；其次，在可变长指令集中，ROP短序列的选择会利用指令不进行对齐检查的弱点，曲解恶意使用二进制代码序列，如从长指令的中间位置解析指令，而这样选择的目的往往是搜寻ret、jmp指令的操作码；再者，ROP指令序列的选择会避开函数的入口代码序列，因为函数的前奏(function prologue)会改变栈结构，颠覆ROP攻击自身的控制流，使之失控；再者，从函数的入口开始选择短序列没有意义，否则Return-Into-Libc的防御方法对ROP将同样有效，况且，统计表明，在函数入口的前5条指令内，几乎不会有ret指令。因而，ROP短序列的入口地址不可能是函数的入口点，这是ROP攻击短序列与函数调用最大的区别，也是ROP攻击对类库的使用与正常使用类库API调用的最本质的区别。而这一本质特性将有助于我们有效检测ROP攻击序列。

3.2 特征提取与比对

我们知道，上述的ROP特征是动态特性，即ROP攻击动态所表征出的行为。我们很难从静态分析去检测，甚至无法获得有意义的数据。而动态二进制插桩(DBI)思想的提出及其工具的出现给了我们实现的可能性。使用DBI工具，我们能够获得程序在执行时的状态，如当前执行指令地址、寄存器值、内存访问、引用的类库的位置等，与此同时，DBI工具不会改变程序自身的行为、状态和结果，因而能够在执行程序时获得一些我们无法在静态所获得的准确数据。而这些数据和状态足以提供足够的信息和特征来比对ROP程序的动态特征，以准确识别ROP攻击。

在检测方案中，通过DBI工具Pin在程序加载时，获取程序涉及到的库的信息，分析这些库，获取库中的函数列表、首地址、函数体大小、偏移等信息，并将其维护在HashMap结构中。然后通过Pin工具启动待检测程序，识别每条指令执行的状态。当指令地址落入库代码段中时，限制其对库代码的恶意使用，抑制其对函数的非法调用，即限制ROP序列从函数体中间位置进入函数体，而绕开函数前奏。ROP攻击一般通过栈中预填的指令地址，直接进入库中代码的具体位置，而不是从库入口或函数入口进入。如果检测到从非法的位置直接进入代码库，那么检测工具将检测到该非法行为，并识别ROP攻击。

3.3 ROPDetect体系结构

基于上述分析和论证的ROP攻击序列和正常的函数调用的特征的本质区别，设计了基于Pin内核的Pin tool检测工具ROPDetect，以检测ROP序列。其体系结构如图3所示。

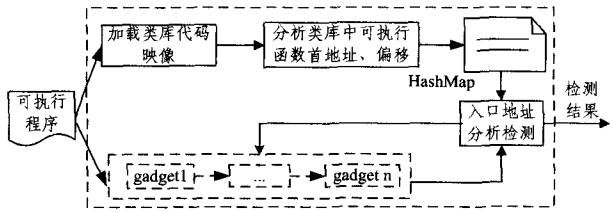

图3 ROPDetect体系结构

如图3所示，我们的ROPDetect工具是基于Pin内核的。首先，使用Pin内核调用加载器识别程序中所有用到的代码库，然后分析这些类库，提取出类库的加载起始地址、结束地址、所有函数的入口地址、各个函数体的大小以及相对库起始地址的偏移。然后将各个函数的信息存入HashMap数据结构中，以供后续检测。最后，使用Pin内核启动待检测程序，检测进入到代码库中的指令的入口是否合法。如果合法，则继续；否则，出现非法的指令入口位置引用，则报警。

4 实现方法

在Linux Ubuntu 10.10，Gcc 4.4.5的环境中，基于Pin 2.10开发ROPDetect检测工具，并使用libc.so.6库。ROPDetect有260多行代码。用Pin内核动态提供的程序数据和内存中库的信息，分析加载的类库，分析libc库中函数的条目信息、入口地址等信息，用STL的map来存储这些条目信息。使用Pin内核启动待检测程序，获取指令执行序列信息，提取指令地址、指令类型等信息；然后对进入libc的指令序列进行检测，检测是否合法地使用类库，而不是恶意使用，分析是否具有上述的ROP攻击特征。如果出错，则报警。

4.1 ROPDetect Pintool架构

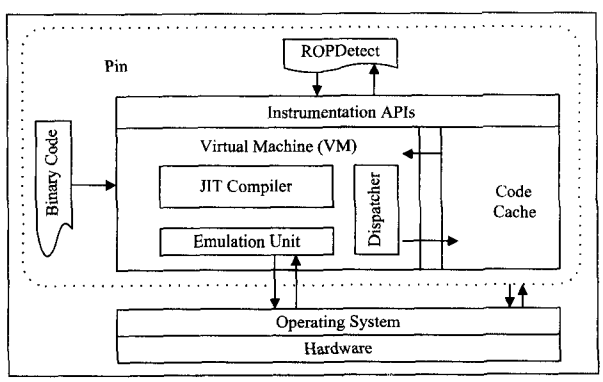

图4 ROPDetect PinTool

ROPDetect Pin tool架构如图4所示。我们通过Pin动态分析能力来实现ROPDetect工具，基于Pin内核获取程序

执行时的信息，提取加载进内存的共享库信息[20]，并通过其提供的回调处理函数机制来实现指令状态的动态检测。

4.2 函数入口信息提取

通过提取 ELF[19] 格式文件的符号表来获取库中所有函数的信息。ELF 符号表中包含了所有函数的列表、相对 Libc 其实地址的偏移量、函数的体的大小信息。可以使用 readelf[21]、objdump[22] 等反汇编工具获取这些信息。因为共享库的代码都是可重定位的代码，所以其符号表中的函数列表的地址都相对偏移，因而这部分可以静态获取。然后使用了 Pin 获取动态类库的实际加载位置，进而计算实际的函数的线性地址。

在实际程序中，使用了 Pin 的 Routine 和 Section API 去动态获取，并使用 Pin 直接获取了 libc 加载时的起始地址、结束地址、内存布局情况。

需要特别指出的是，地址获取是能够克服 Linux 地址随机化的。即，在开启地址随机化（Address Space Layout Randomization，ASLR[23]）的 Linux 情况下，依旧能够获取类库的加载位置。因为 Pin 工具是在类库已经加载完成后通过 Linux 自身的加载器去获取类库映像信息，所以地址的随机化不会对我们的工具产生影响，但能增加 ROP 攻击的难度[24]。

4.3 检测分析算法

检测算法的流程如图 5 所示。ROPDetect 检测待检测程序对于 libc 库的访问，检查待检测程序对于 libc 库函数的访问是否从函数的入口进入，是否绕过函数的前奏（function prologue）；通过 Pin 的指令回调接口插入检测代码；通过 STL 的栈来判定指令执行流是初次进入库函数，还是库函数调用其它库函数后的返回进入。

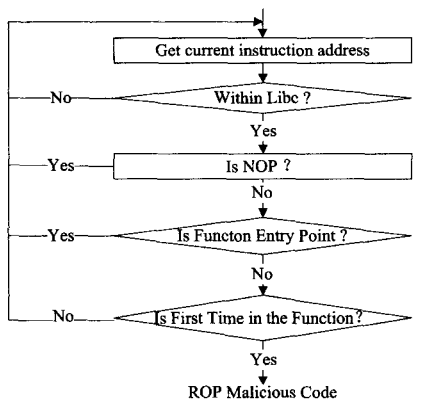

图 5 检测算法流程

5 实验数据分析

本节将阐述 ROPDetect 检测工具检测的正确性，分析 ROPDetect 工具的性能，探讨检测工具的束缚和改进方向。

5.1 检测的正确率

使用 glibc 的代码片段构造了 10 个 ROP 攻击的例子，其中包含以 ret 结尾和以 jmp 指令结尾的指令，我们的程序准确地预警了这些 ROP 攻击程序对于 libc 库的非法访问。在构造 ROP 攻击时，关闭了 Linux 的地址随机化功能，这样做的目的是降低 ROP 攻击构造的难度。但是，需要说明的是，Linux 的地址随机化对于检测程序没有影响，因为 Pin 是通过加载器获得 libc 加载的地址，而 ROP 攻击程序却不具备这样的能力。因而，地址随机化与检测工具的结合会增加 ROP 攻击的难度，提高检测的准确度。

同时，使用并改写了 Ping Chen[33] 等提供的 ROP 攻击案例，包括 kill process，beep，dup 等，这些例子既有传统的 ROP 攻击，也有非传统 ROP 攻击，这些使用 libc 库中的代码段构建的 ROP 攻击被 ROPDetect 准确检测出。与此同时，检测了 Linux 系统常见的应用程序，这些正常合法的程序准确地被 ROPDetect 检测工具识别为合法程序。

表 1 是测试的部分数据，在 ROPDetect 监护执行下的一部分程序的行为，包括 ROP 攻击程序和正常的合法程序。

表 1 ROP 检测正确性数据

Program	Size	Description	Exploit Type	Detected
-bin-sh-rop	3.77kB	execute /bin/sh	ROP	√
beep	5.38kB	make a sound	ROP	√
settime	3.92kB	set system time to 0	ROP	√
dup	6.65kB	copy and paste a file	JOP	√
killall5	4.33kB	killall5 shellcode	ROP	√
killprocess	3.79kB	kill all processes	JOP	√
gcc4.4.5	—	compile 180.9kB file	NO	pass
tar1.23	—	uncompress 28.7M file	NO	pass
cp8.5	—	copy 220.6M file	NO	pass
gzip1.3	—	compress 236.7M file	NO	pass

实验数据表明，程序能够准确检测构建于 libc 库之上的 ROP 攻击。

5.2 检测的效率

下面给出检测工具的效率，测试了 Linux 环境下一些常见的工具在 ROPDetect 工具下的效率（使用 Linux 的 time 命令统计 CPU 实际运行时间）。具体对比数据如表 2 所列。

表 2 性能测试

Program	Benchmark	Native Run	With ROPDetect	Perfor-mance
gcc4.4.5	Compile 180.9 KB file	27.694s	33.998s	1.2X
tar1.23	Uncompress 28.7 M file	3.968s	10.389s	2.6X
cp8.5	Copy 220.6 M file	1.368s	5.564s	4.1X
gzip1.3	Compress 236.7M file	1.144s	3.236s	2.8X
		AVERAGE		2.7X

测试了 Linux 的常用命令和应用程序，得出的统计数据表明，我们的工具对于程序的效率的影响是 2～10 倍，上述仅是部分数据。对于性能的影响，与被测程序对于 libc 库的使用的频率和使用的次数有关。但这也与 Pin 内核本身的性能束缚有关，Pin 内核本身的性能消耗增加 0.6 到 1 倍左右[18]。而且，测试表明，我们的程序对于大型程序的时间的消耗在 3 倍左右，而对于小程序的测试数据不是很稳定。这是由于测试的工具和精度的问题，而且操作系统调度等对时间测试的扰动比较大。因而，我们的性能是相对比较稳定和可以接受

的。

5.3 与已有 ROP 防护工具的比较

目前已有的采用动态分析技术的 ROP 防护工具有 ROPdefender[32] 和 DROP[33]。ROPdefender 只能检测出传统的 ROP 攻击序列，即以 ret 结尾的指令序列，对于新型的 ROP 攻击变种，如 JOP[30] 攻击，却毫无作用，对于将来可能的 ROP 变种[1]亦不能检测出。同时，ROPdefender 的防护思想是严格基于 x86 平台的，平台的可扩展性较差。DROP 基于统计学的规律使用 DBI 技术分析程序的统计特征，进而识别 ROP 攻击，其思想是较好的，但是，实验数据表明，这样的统计规律是难以成立的，攻击者很容易通过向 ROP 攻击序列中填充无用代码，如 NOP 指令，来稀释这样的统计特性，而且，对于设计巧妙的精悍的 ROP 攻击，这样的统计规律本身不成立。再者，对于非传统 ROP 攻击，DROP 也不能起作用，平台的移植性也较差。

6 相关的工作

目前，对于 ROP 攻击的检测和防御的研究已有相当进展。下面将分类介绍其中具有代表性的研究成果，分析其优劣，并与本文介绍的检测工具进行对比。

6.1 基于控制流的防护

ROP 攻击最本质的诱因是控制流被劫持，如果能够保证控制流完整性，那么 ROP 攻击就不可能得到实施，与此同时，很多漏洞和攻击都能被有效，即抑制。Martin Abadi[25] 等实现了根据 Control-flow Graph（CFG）检查控制流的方法。Vladimir Kiriansky[26] 提出了 Program Shepherd 的思想，即针对特定目标进行特定防护，监视程序的状态，确保控制流的完整性。Crispin[27] 等介绍了栈防护的一些方法，其一定程度上能够保护栈的完整性，防止程序栈遭受攻击。但是，出于性能、灵活度、算法复杂度、实现难度等各方面的考量，控制流的完整性检查还很不完善。

6.2 地址随机化

在 ROP 攻击中，gadget 的构建中很重要的一步就是获取短序列的首地址。因此，短序列的实际首地址将决定程序的控制流，如果每次都能随机加载程序库（libc），程序库每次加载的起始位置随机化，那么无疑将使 gadget 的控制流程和地址失效。因此，地址随机化将增加 ROP 攻击的难度。Shacham[24] 等描述了针对地址随机化的方法及其实际效果，并介绍了在 32 位机器上应对地址随机化问题的方法；Randal E. Bryant[28,29]，描述了攻击者如何通过空雪橇操作在地址随机化环境中定位地址的方法。在 32 位机器上，地址随机化的效果不是很好，地址可以很快通过很多方法定位，不会对 ROP 攻击造成很多束缚，但会增加攻击者的难度；在 64 位机器上的地址随机化将使地址的定位变得十分困难，ROP 的能力会受到很大影响。因此，向 64 位机器的迁移将大幅提高 ROP 攻击的难度，同时，在 32 位机器上的精细的随机化，将有助于提高 ROP 攻击的难度，但需要同其他方法相结合才能保证程序的相对安全性。

6.3 基于编译阶段的防护

基于编译阶段的 ROP 防护旨在通过修改编译器，在编译阶段消除能够被 ROP 攻击者恶意使用的指令序列，如减少 ret 指令。Jinku Li 等[30] 提出分析 ret 指令的来源，针对每一种情形分别进行处理，消除被曲解的 ret 指令，对于实际的 ret 指令，作者使用间接返回（return indirection）的策略，即将可能被篡改的返回地址抽取出来放在一个单独的表中，而在栈中原来的位置存放该地址在表中的索引。每次返回时，通过该索引去获取实际的返回地址，这将有效地保护返回地址，防止程序流程被篡改。但是，该方法需要重写编译器，需要重新编译原有代码，对于庞大的遗留代码很难有效。Kaan[31] 提供了消除 ROP gadget 的方法，但其也面临 Jinku Li[30] 方法中的同样问题。

6.4 基于 DBI 的防护

Lucas Davi[32] 等基于 DBI 实现了 ROP 的防护，即通过 DBI 提供的动态信息，使用 Shadow Stack 保护函数的返回值不会被篡改，从而有效防止传统的 ROP 攻击。但是该方法无法检测出以非 ret 指令结尾的非传统的 ROP[11] 攻击，而 ROPDetect 工具能够较好地解决上述问题。在 Ping Chen 等[33] 通过 ret 指令序列的长度及其连续出现的频率来识别 ROP 攻击，这在一定程度上能够识别传统 ROP 攻击，但是对于非传统 ROP 攻击同样不适用，而且，对于某些特别的 ROP 攻击序列，这样的统计规律不一定成立。

结束语 基于 ROP 攻击的 gadget 构建的来源、其对已有代码的非法利用，利用动态二进制插桩技术攻击其本质属性，能够准确检测传统的 ROP 攻击以及最近的 ROP 攻击的变种。ROP 攻击 gadget 的构建是其本质、也是其关键的一步，而图灵完备特性是 ROP 攻击的命脉，而这体现在 gadget 的构建上，即对于大量现有代码的非法使用，如果没有大量的现有代码，抑或没有一些关键代码，如系统调用的 libc 函数的汇编片段，那么 ROP 攻击就很难获得更高的权限。因此，如果能够有效地限制这些代码，那么无疑将有效限制 ROP 攻击。Libc 库为许多应用程序提供了内核调用的接口，检测工具已经准确地限制了 libc 中所有的关键代码片段的非法调用，这将杜绝 ROP 攻击利用 libc 库攻击，也杜绝了 ROP 的绝大多数的能力。

我们的工具可以对加载进内存的程序库进行被调用检测，限制对这些代码的恶意使用，因此，通过对该检测工具进行必要的修改，就能对加载进内存的代码进行合法性使用检查，而不会带来不可接受的性能开销。在这种条件下，ROP 的图灵完备性被打破，其计算能力将受到较大限制，更关键的是，其 gadget 的构建更为困难。同时，该检测工具移植性较强，因为基于函数完整性检查的思想的通用性和 Pin 的多平台支持特性，可以帮助我们很好地实现这一目标，所以能够在现有的绝大多数平台中完成 ROP 攻击的检测。

同时，我们不认为 ROPDetect 能够检测出所有攻击，只有多种防御手段结合起来，才能有效防止 ROP 攻击。地址随机化能够有效增加 ROP 攻击构建的难度；基于编译的优化和防护能够在将来新引进的代码中很好地抑制 ROP 攻击；控制流检查的强制实施及其硬件的支持，能够在不损失性能和灵活性的前提下，提高 ROP 攻击的防护和程序安全性。

参 考 文 献

[1] Shacham H. The Geometry of Innocent Flesh on the Bone[C]//

Return-into-libc without Function Calls (on the x86): CCS'07 Proceedings of the 14th ACM Conference on Computer and Communications Security. New York, NY, USA: ACM, 2007: 552-561

[2] Du Wei-liang. Return-to-libc Attack Lab [EB/OL]. http://www.cis.syr.edu/~wedu/seed/Labs/Vulnerability/Return_to_libc/Return_to_libc.pdf, 2007

[3] Nergal. Advanced return-into-lib(c) exploits (PaX case study) [EB/OL]. http://www.phrack.org/issues.html? issue = 58&id=4&mode=txt

[4] Romer R, Buchanan E, Shacham H, et al. Return-Oriented Programming: Systems, Languages, and Applications [C] // Return-Oriented Programming. USA: ACM: 1-40

[5] Krahmer S. x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique [EB/OL]. http://packetstorm.igor.onlinedirect.bg/papers/bypass/no-nx.pdf, 2005

[6] Buchanan E, Roemer R, Shacham H. When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC [C] // CCS'08 Proceedings of the 15th ACM Conference on Computer and Communications Security. New York, NY, USA: ACM, 2008: 27-38

[7] Hund R, Holz T, Freiling F C. Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms [C] // SSYM'09 Proceedings of the 18th Conference on USENIX Security Symposium. CA, USA: USENIX Association Berkeley, 2009: 383-398

[8] Dullien T, Kornau T, Weinmann R-P. A framework for automated architecture-independent gadget search [C] // Proceedings of the 4th USENIX Workshop on Offensive Technologies (WOOT). Washington, DC: USENIX Association, 2010

[9] Kornau T. Return Oriented Programming for the ARM Architecture [EB/OL]. http://zynamics.com/downloads/kornau-tim-diplomarbeit-rop.pdf, Master thesis, Ruhr-University Bochum, Germany, 2009

[10] Davi L, mitrienkoy A, Sadeghi A-R, et al. Return-Oriented Programming without Returns on ARM [C] // HGI-TR-2010-002. Ruhr University Bochum, Germany, 2010

[11] Checkowayy S, Daviz L, Dmitrienkoz A, et al. Return-Oriented Programming without Returns [C] // CCS '10 Proceedings of the 17th ACM Conference on Computer and Communications Security. New York, NY, USA: ACM, 2010: 559-572

[12] Team P. Documentation for the PaX project [EB/OL]. http://pax.grsecurity.net/docs/index.html, 2003

[13] Team P. PaX non-executable pages design & implementation [EB/OL]. http://pax.grsecurity.net/docs/noexec.txt

[14] Nethercote N, Seward J. Valgrind: A Framework for Heavy-weight Dynamic Binary Instrumentation [C] // PLDI' 07 Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA: ACM, 2007: 89-100

[15] Valgrind Developers: Valgrind (2000—2005) [EB/OL]. http://www.valgrind.org/

[16] Nethercote N, Seward J. Valgrind: A Program Supervision Framework [J]. Electronic Notes in Theoretical Computer Science, 2003, 89(2): 44-66

[17] University of Virginia, Pin [EB/OL]. http://www.pintool.org/

[18] Luk C-K, Cohn R, Muth R, et al. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation [C] // PLDI' 05 Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, NY, USA ACM, 2005: 190-200

[19] Tool Interface Standards Committee. Executable and Linking Format (ELF) [EB/OL]. http://flint.cs.yale.edu/cs422/doc/ELF_Format.pdf, 1995-05

[20] Bryant R E, O'Hallaron D R. Computer Systems, A Programmer's Perspective (Second Edition) [M]. 2001: 523-556

[21] readelf [EB/OL]. http://www.sourceware.org/binutils/docs/binutils/readelf.html

[22] objdump [EB/OL]. http://sourceware.org/binutils/docs/binutils/obj-dump.html

[23] Team P. ASLR [EB/OL]. http://pax.grsecurity.net/docs/aslr.txt

[24] Shacham H, Page M, Pfaff B. On the Effectiveness of Address Space Randomization [C] // CCS' 04 Proceedings of the 11th ACM Conference on Computer and Communications Security. New York, NY, USA: ACM, 2004: 298-307

[25] Abadi M, Budiu M, Ligatti J. Control-Flow Integrity Principles, Implementations, and Applications [J]. ACM Transactions on Information and System Security (TISSEC), 2009, 13 (1): 4

[26] Kiriansky V, Bruening D, Amarasinghe S. Secure Execution Via Program Shepherding [C] // Proceedings of the 11th USENIX Security Symposium (Security '02). San Francisco, California, 2002: 191-206

[27] Cowan C, Wagle P, Pu C, et al. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade [J]. DARPA Information Survivability Conference & Exposition, 2000, 2: 119-129

[28] Bryant R E, O'Hallaron D R. Computer Systems, A Programmer's Perspective (Second Edition) [M]. 2001: 115-219

[29] Seacord R C. Secure Coding in C and C++ of strings and integers [J]. Security & Privacy, 2006, 4(1): 74-76

[30] Li Jin-ku, Wang Zhi, Jiang Xu-xian, et al. Defeating Return-Oriented Rootkits With "Return-less" Kernels [C] // EuroSys' 10 Proceedings of the 5th European Conference on Computer Systems. New York, NY, USA ACM, 2010: 195-208

[31] Onarlioglu K, Bilge L, Lanzi A, et al. G-Free: Defeating Return-Oriented Programming through Gadget-less Binaries [C] // ACSAC'10 Proceedings of the 26th Annual Computer Security Applications Conference. New York, NY, USA ACM, 2010: 49-58

[32] Daviy L, Sadeghiy A-R, Winandyz M. ROPdefender: A Detection Tool to Defend Against Return-Oriented Programming Attacks [C] // ASIACCS' 11 Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. New York, NY, USA ACM, 2011: 40-51

[33] Chen Ping, Xiao Hai, Shen Xiao-bin, et al. DROP: Detecting Return-Oriented Programming Malicious Code [C] // Prakash A, Gupta I, eds. Fifth International Conference on Information Systems Security (ICISS 2010). volume 5905 of Lecture Notes in Computer Science, Springer, 2009: 163-177

[34] Turing A M. On Computable Numbers, with an Application to the Entscheifungsproblem [C] // Proceedings of the London Mathematical Society 42. 1936: 230-265