基于 Return-Oriented Programming 的程序攻击与防护

黄志军 郑 滔
(南京大学软件学院 南京 210093)

摘 要 随着 W⊕X 等技术的引入,传统的代码注入攻击几乎被消除,return-to-lib 攻击受到很大程度的抑制。在此背景下,Hovav Shacham 提出了 Return-Oriented Programming(ROP)的思想,该思想基于栈溢出的原理,通过使用程序库中有效的以 ret 指令结尾的短指令序列构建 gadget 集合,使之具有图灵完备特性,来完成计算和攻击。讲述 ROP 思想自提出以来的一些研究成果和其实际的攻击能力,阐述 ROP 自动化的当前成果与未来可能的发展方向,进而分析和预测 ROP 自动化的下一步的研究方向。同时,也将从 ROP 的几个特征分析消除这种攻击的策略和方法,并介绍目前已有的防护思想和成果,论述这些方法的优缺点和改进方向。综合阐述 ROP 攻击与 ROP 防护这一矛与盾的问题,力争使读者理解 ROP 的思想,知悉当前的发展状态,并在此基础上能够进一步推进 ROP 攻击及其防范的研究。

关键词 ROP,程序自动化,地址随机化,栈溢出,程序控制流,程序安全
中图法分类号 TP309.5 **文献标识码** A

Program Attack and Protection Based on Return-Oriented Programming

HUANG Zhi-jun ZHENG Tao
(Software Institute, Nanjing University, Nanjing 210093, China)

Abstract With the adoption of $W \oplus X$ technology, the traditional code injection attacks have been almost eliminated, so the return-to-lib attack has been greatly restrained. Under this circumstance, Doc. Hovav Shacham promoted the ROP idea, which is short for Return-Oriented Programming. Based on the theory of stack overflow, making using of the valid short instruction sequences that end with ret instructions to construct gadget collections with Turning-Complete features, the ROP idea can accomplish the task of compute and attack. In this paper, we presented achievements in ROP field and ROP's ability of attack since its promotion, and then illustrated the direction for development of the automation of ROP attack and its current achievements, after that, analyzed and predicted the future development of ROP automation. Simultaneously, we discussed strategies and methods aiming at eliminating this attack based on its characteristics, introduced exisiting achievements of defending this attack by comparing their merits and demerits, gave our own perspectives of these defending strategies and methods about how to change and improve them.

Keywords ROP, Program automation, Address randomization, Stack overflow, Program control flow, Program security

1 概述

程序安全对于计算机的发展和应用起着至关重要的作用。随着安全防护技术的发展,攻击者不停地尝试利用系统各种漏洞对程序进行攻击。起初攻击者尝试通过向栈中注入恶意代码,劫持正常的程序控制流程,将攻击程序的控制流转入攻击者预先设置的控制流,并执行恶意代码,进而实施攻击。

后来,出现了几种具有针对性的防护技术,包括 Solar Designer's Stack Patch[1]技术,即通过修改可执行程序的内存布局来使程序运行栈不可执行,防止注入攻击;再者是更为复杂的 W⊕X 技术,即通过硬件或软件的支持,来保证进程映像中的内存区域不能同时可执行或可写入,在最新的 Linux PAX[2]项目中,已实现这项技术[3],通过将无执行必要的页设为不可执行,限制在可写入页和可执行页间的转换,限制程序动态生成并执行代码的能力;而对于确实需要动态生成代码的程序,PAX 也提供解除这些限制的方法;与此同时,PAX 使得栈和堆不可执行,控制 ELF 的文件映射,限制访问权限,ELF 文件中,只有实际的代码段才能被执行,而 ELF 文件头等位置不能被执行。PAX 同时限制了既可执行又可写的文件映射的创建,防止动态生成恶意攻击代码。在最新的 OpenBSD, AMD, Intel 中也实现了类似的保护机制,从而有效地限制了恶意注入代码的攻击。

在此基础上,新的攻击技术 return-to-libc 继而出现,通过利用已有代码,如 libc 库函数中的代码,实施攻击。攻击者通过栈溢出,覆盖正常函数栈中的返回地址,使恶意调用的函数的参数覆盖栈中返回地址邻近区域的内存空间,然后函数调用返回时,会去执行 libc 中的攻击者指定的函数,如使用 sys-

本文受国家自然科学基金项目(60773171),国家"863"计划基金重点项目(2007AA01Z448)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。

tem()，执行 shell，达到恶意企图。在 Attack Lab[4] 中，介绍了 return-to-libc 攻击的简单实验，在 Nergal[5] 的文章中，介绍了基于 PAX 系统的高级 return-to-libc 技术。值得强调的是，不但 libc 中的代码可以被使用，任何可执行的代码都有可能被攻击者恶意调用和攻击。

return-to-libc 技术虽然使得攻击者绕过了基于 W ⊕ X 等技术的防护措施，但是与基于代码注入的攻击相比，还是受到很大限制。其一，在 return-to-libc 中，被恶意调用的函数的执行顺序只能是线性的，不能使用分支或者跳转，而恶意注入代码并无此限制；其二，恶意调用的函数只能是程序的代码段或者加载的库中可用的函数，如果去掉某些危险的库函数，将很大程度上限制攻击者的攻击能力。因而，W ⊕ X 技术某种程度上也限制了攻击者的能力。在此基础上，Hovav Shacham 提出了 Return-Oriented Programming(ROP) 的方法，即通过使用已有的程序库，包括 libc、驱动程序等可执行的代码片段，构建图灵完备的程序，达到攻击程序的正常控制流程、提升攻击者操作权限甚至完全获取计算机控制权的目的。该方法能够绕过当前很多杀毒软件的扫描和病毒库特征比对。完全绕过 W ⊕ X 等内存保护技术的防护对计算机安全造成的威胁将不可估量。

2 ROP 思想的起源

在 Hovav Shacham 的关于 ROP 经典论文[6] 中，作者阐述了关于 ROP 攻击的核心思想和对具体攻击实施的一些指导意见。首先，我们在 X86 机器上来阐述这一思想。计算机中存储着大量的机器代码，这些指令以二进制的形式贮存于计算机中，按照字节的形式驻留在磁盘或者内存中，CPU 则逐条执行这些指令。在这些字节序列中，由于指令集编码的缘故，随着解析的起点不一样，同一字节序列可以解析出多种有效的指令序列。我们引用 Hovav Shacham 论文中的例子来说明，在 libc 的 ecb_crypt 的函数入口处，有如下指令：

f7 c7 07 00 00 00	test $0x00000007,%edi
0f 95 45 c3	setnzb −61(%ebp)

这是程序编写者期望的汇编指令，但是如果我们向后偏移一个字节来解析这个字节序列，则得到如下指令序列：

c7 07 00 00 00 0f	movl $0x0f000000,(%edi)
95	xchg %ebp,%eax
45	inc %ebp
c3	ret

如上所示，结果将截然不同。在 X86 ISA 体系结构中，指令是稠密的，没有采用严格的对齐策略，因而这样的情况会大量存在。

ROP 的思想是通过在 libc 等库的代码字节序列中寻找以 ret 结尾的指令序列，然后将这些指令序列链接，形成能完成简单操作的指令序列，例如加法运算，称之为 gadget，并通过将这些 gadget 串联来完成一些复杂的操作。经验规律表明，在 X86 ISA 这样的指令集中，存在大量这样的指令序列来形成可以完成基本操作的 gadget 集合，这些 gadget 的组合能产生图灵完备的表述能力。不仅在 X86 ISA 体系结构中如此，在 SPARC 这样的定长指令 RISC 体系结构中，同样存在这样的图灵完备的表述能力[7]。

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，系统会分配一个栈帧给当前被调用函数，用于参数的传递、局部变量的维护、返回地址的填入等。栈帧是程序运行栈的一部分，在 Linux 中，通过%esp 和%ebp[13] 寄存器维护栈顶指针和栈帧的起始地址，%eip 是程序计数器寄存器。而 ROP 攻击则是利用以 ret 结尾的程序片段，操作这些栈相关寄存器，控制程序的流程，执行相应的 gadget，实施攻击者预设目标。

ROP 不同于 return-to-libc 攻击之处在于，ROP 攻击利用以 ret 指令结尾的函数代码片段，而不是整个函数本身去完成预定的操作。从广义角度讲，return-to-libc 攻击是 ROP 攻击的特例。

Hovav Shacham 在 Linux 平台上，使用 libc 中发现的代码片段，组成 gadget 集合，形成图灵完备的表述能力。这些 gadget 能够完成以下的基本操作：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)，这些操作按照一定规则运算，具备图灵完备的表述能力。同时介绍了使用 ROP 攻击技术执行 Shell 脚本的攻击能力。我们在 Linux 平台上验证了作者的实验，证明了其可行性。一个具有图灵完备表述能力的 ROP 技术能够产生不易被发现的危险攻击。

Eric Buchanan[7] 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 Hovav Shacham 的技术实现了同样的攻击，并利用 ROP 技术进行矩阵加法运算，证明 ROP 不仅对于可变长指令集有效，对于定长指令集同样有效。Tim Kornau[20]，Lucas Davi[21] 实现了在 ARM 体系结构中 ROP 攻击的实施。Stephen Checkowayy[22] 等实现了非 ret 指令结尾的 ROP 攻击[1]。

3 ROP 攻击的具体实施方案

ROP 思想的关键在于分析 libc 库[2] 中的代码片段，获取有效的指令片段，然后利用这些指令片段去构建 gadget，再使用这些 gadget 去构建一组操作，进而实施运算和攻击。

因此，攻击的第一步是在 libc 的代码段中寻找以 ret 指令结尾的指令序列，并获取这些指令序列的起始地址。在此，值得强调的是，寻找以 ret 结尾的长度为两到三条以内的指令序列的原因在于：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，粒度小，构建 gadget 所展现的灵活性大增；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，而这往往会抵消该长指令序列所带来的效果。一种极端的情况是，只寻找长度为 2、只包含一条指令和 ret 指令的短序列。

在 Shacham 的论文中，作者通过使用哈希树的变种 Trie

1) ROP 变种 JOP，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。

2) 这里我们以 libc 库去描述 ROP 攻击的实施方案，但我们强调的是，任何驻留在内存中的有效的代码块都可能是 ROP 攻击的候选利用对象，只要其能够提供足够的表达能力。

树结构来存储找到的指令序列，ret 指令作为树的根，使用递归反向遍列 libc 库的代码字节序列，节点存储有效指令序列的地址，在遍列完成后，从根节点到树中任何一个节点的路径上的所有序列构成一个有效的指令序列。我们可以取路径长度为 1 的节点，构成长度为 2 的指令序列；取路径长度为 2 的节点，构成长度为 3 的指令序列，以此类推，直至 X86 最长指令长度——20 字节。这种方法的好处在于能够找出所有的有效的指令序列，便于后续 gadget 的构建，其指令序列长度可长可短，灵活性高，表述能力强；但其耗时较多，效率较低。此时，可以通过限制 trie 树的层次，缩短待发现的指令序列的长度来提高效率。

其次，当构建好指令序列的 Trie 树结构后，我们需要利用这些有效的指令序列去构建具备图灵完备特性的 gadget 集合[3)]。一个最为直观的方法就是人工地分析找到有效的指令序列，分析这些指令序列之间寄存器的依赖关系，看其是否会产生副作用(如，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值是否被破坏)，分析操作数的类型限制。例如，如果只找到 andb 序列，却需要计算 and，那么必须对两个 32bit 的操作数进行 4 次 andb 操作，才能完成这两个操作数的 and 操作。虽然有时找到的指令序列能与需要的基本操作有很好的对应关系，但是会产生副作用，一般在 ret 指令序列前出现 push 指令，改变栈顶的值，就容易出现副作用。在 Shacham 的论文中，列举了 add 指令副作用及其消除方法。当构建好具备图灵完备特性的 gadgets 集合时，基本操作集就已经构建完成，此时，其就已经具备一定的计算能力。

当然，gadget 构建也可进行自动化，程序通过分析 Trie 树中的有效指令序列，分析这些指令序列与需要的 gadget 的计算能力之间的关系，自动分析指令序列中的寄存器、内存区域的使用的依赖关系、操作数的传递规则，自动构建所有的可能的 gadget，进而提高 ROP 自动化的效果。在 Ralf Hund 等的论文[9]中，作者介绍了在 Windows 平台上，使用驱动程序和核心库代码段，基于 Shacham 的 Trie 方法，自动构建 gadget 方法，程序通过分析所有已找出的有效指令序列，逐个比对有效指令和需要的功能的指令序列之间的关系，通过有向图分析各条指令使用的寄存器、内存区域的依赖关系，进而发现所有有效的候选 gadget，并选择序列长度最短的 gadget 作为最终的 gadget，实现了一定程度的自动化。但是其束缚还是比较明显的，首先，其只筛选所有序列长度为 2 的指令序列，这将约束 gadget 的完备性，限制 ROP 程序的通用性。因为，这将需要大量代码段去找到满足图灵完备的 gadget 集合，使得程序的防护者通过去除某些库片段来防范 ROP 攻击成为可能；其次，程序无法对指令序列长度为 3 及以上的指令序列自动生成 gadget，很难通过简单的自动化去消除长指令序列的副作用。一个好的方法就是首先自动化寻找短指令序列，看其是否能够满足绝大多数具备图灵完备的计算能力，然后再在剩下的长序列中寻找有效的指令序列去构建剩余的gadget，两者结合，使这些 gadget 的集合具有图灵完备的计算能力。

当所有的 gadget 构建完成后，通过组织这些 gadget 来进行运算。理论上来说，可进行各种运算。在 Ralf Hund 等的论文[9]中，作者使用 ROP 构建了 Return-Oriented Rootkit，进而隐藏恶意进程，同时实现了基于 ROP 的排序算法。在此，可以认为，如果 ROP 的 gadget 表述能力能够具备图灵完备的特性，那么完全可以使用 ROP 完成任何操作，而绕过很多病毒防护措施。

4 ROP 攻击的自动化与高级编程语言

ROP 攻击始于 gadget 的构建，然后组织 gadget 进行运算。这是个很繁琐和枯燥的过程，如果纯粹手工完成，那 ROP 攻击构建效率将大幅降低。因此，需要通过自动化来完成 gadget 的构建和 ROP 程序的生成。

基本思路是通过程序自动化扫描并分析程序库，发现有效的短指令序列，自动构建 Trie 树，并获取短指令序列集合。然后分析这些有效的短指令序列，构建图灵完备的 gadget 集合，然后将这些 gadget 集合作为基础的“指令”。第二步是定义一门高级程序语言，如像 Java、C 一样的高级语言，定义通用的语法，让不懂得 ROP 原理的程序人员，能够像写一般 Java 代码或者 C 代码一样，编写他们的程序，完成他们需要的功能。在此基础上构建这门语言的编译器，目前有很多根据预定义的语法规则[10]自动生成编译器的工具，例如 Lex[11]。通过该编译器，将定义的高级语言编写的 ROP 高级语言程序编译为 gadget 序列的组合。需要强调的是 gadget 是指一个数据集合，每个数据或者是实际代码片段的首地址，或者是实际操作数的集合，而不是代码片段，因而这样的 gadget 序列可以看作是字符串序列。将这些 gadget 串联起来，得到一个长字符串，注意，这个长字符串中间没有 NULL 字符或者‘\0’。然后通过栈溢出等机制，将这个长字符串写入到栈中，来攻击当前程序，当前函数调用返回时，正常程序流程被劫持，进而按照预先定义的 gadget 的序列去执行攻击者的意图。

更进一步的自动化在于，能够有程序自动去搜索待攻击程序的可能的栈溢出漏洞，例如 C 程序中的 strcpy，printf，char[]等指令，自动找出程序的可能的漏洞，并实施自动化的栈溢出发现和攻击。Halvar Flake[12]在其演讲中介绍了常见的发现栈溢出漏洞的方法和策略，其中的一些方法可以用来搜索和发现常见的栈溢出可能的场景，有助于栈溢出漏洞的自动化搜索。

目前自动化做得比较好的 ROP 攻击是 Ralf Hund 等的论文[9]中的实验。在这篇论文中，作者基于 Windows 平台，使用驱动程序和 Win32 内核程序库，使用 Shacham 的 Trie 结构搜寻长度为 2 的短序列，自动化构建 ROP 攻击程序的过程。该实验构建了一种类 C 语言来实现 ROP 从高层次抽象到实际 gadget 的过程。在其实验中，其系统分为 3 个模块——构建模块、编译模块和加载模块。构建模块搜寻候选的有效短指令序列，然后构建自动 gadget；编译模块使用预定义的类 C 语言，将使用该类 C 高级编程语言编写的程序翻译成 gadget 的组合；加载模块将高级语言编译好的 gadget 集合加载进内存中，修改其相对地址，并启动该 ROP 程序。加载器通过修改栈顶指针，将程序的控制流引导到 gadget 的起始处，然后启动 gadget。与此同时，作者使用该高级语言，完成了一个 Rootkit 攻击实验，绕过了内核完整性保护机制，如

3) 这些 gadget 能够计算每一个图灵完备的函数[8]。

NICKLE 和 SecVisor，并取得了成功，实现了 ROP 攻击的自动化。作者分析了 ROP 的程序与非 ROP 程序的运行时间的差异，发现 ROP 程序的时间负载与非 ROP 程序差异明显，特别是在构建 gadget 时，会非常耗时，对于一些时间敏感的系统，ROP 程序能够通过检查时间负载被甄别，这也是分析和检测 ROP 攻击的一个方向和思路。在这篇论文中，基于 ROP 的 Rootkit 成功对资源管理器隐藏了攻击进程。基于 ROP 的图灵完备特性，完全可以使用这个实验攻击正常程序的控制流和数据流的检查机制。并且，作者实验的思路也可以通过改进用于其它的平台，例如 Linux 平台，只要将其中用到的平台相关的函数转换成对应平台的具备同样功能的函数即可。同时，该实验的通用性，可以通过特征库来实现平台的兼容性和跨平台型，即提供一个平台相关的数据库，将平台相关的指令进行特征提取，进而用于跨平台的实验。

Thomas Dullien[23] 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 程序的自动化迈出一步。PinChen[24] 提出了在 X86 平台上半自动化构建 ROP 攻击变种 JOP 的方法。

据此，设计一种跨越所有平台的 ROP 自动化程序几乎是不可能的，但是可以抽取平台相关的特征信息，例如所用的地址映像函数、指令集特征，运用反射技术去自动化构建 ROP 的程序，以实现跨平台和平台的兼容性。

5 ROP 攻击的改进方向

ROP 的发展现在还处于初步阶段，当前很多技术还停留在理论阶段，很多实验停留在半理论阶段，存在较多的假设和限制。自动化程度还有所欠缺，目前很难通过完全的自动化产生一个 ROP 程序。当前 ROP 的跨平台性很差，若与机器指令集的关联度过高，指令集的微小变化就可能使原有的有效 ROP 程序失效。

因此，需要发现更多的方法，以确保 ROP 的可行性。我们认为需要从以下几方面着手：

1）诚如 Shacham[6] 在他的经典论文中所说，可以在 X86 中寻找 0xC3 ret 指令，但也可以寻找与 0xC3 有相似功能的指令，例如 C3、C2 imm16，cb，ca imm16，jmp imm(%esp)。虽然对这些指令的处理较为繁琐，会产生更多的副作用（例如栈回退及栈数据的破坏），但在 ret 指令受限的情况下（例如通过编译过程刻意消除某些 ret 指令，降低 ret 指令的密度[16,25]）可以用这些控制指令替代，以确保 gadget 集合的图灵完备特性。

2）不仅可以搜索 libc 库的代码段，任何驱动程序、操作系统内核库等可加载驻留在内存中的代码或者可以理解为代码的数据都可以作为有效指令序列的来源和 gadget 的构建元素，例如 ELF 文件头[4] 也可以作为有效指令序列的来源。

3）进一步深化在 gadget 自动生成方面的研究，更为合理有效地利用长度为 3～4 条指令的短序列自动构建长度，运用编译理论的寄存器的分配策略解决 gadget 的寄存器之间的依赖关系，消除副作用的影响，实现最大程度的自动化，力争使用最少的代码候选集，构造更为合理和高效的 gadget 工具集。

4）加强栈溢出的研究和探索，自动化发现程序的漏洞，进而能够做到 ROP 程序自动加载执行，Halvar Flake[12] 的理论可能对其有一定帮助。

6 ROP 攻击的防护与策略

ROP 攻击的程序主要使用栈溢出的漏洞，实现程序控制流的劫持。ROP 程序通过编译为 gadget 数据串（即 libc 等代码库中的短序列的首地址和 ROP 程序执行用到的数据），然后通过栈溢出的方式以字符串溢出的形式写入到栈中，覆盖正常程序栈内容，实现合法控制流向 ROP 控制流的转变。

因此栈溢出漏洞的防护是阻挡 ROP 攻击最根源性的方法如果解决了栈溢出问题，ROP 攻击将会在很大程度上受到抑制。Crispin Cowan 等[14] 在其论文中介绍了几种防止栈溢出的方法。首先可以通过代码审计的方法检查常见的可能出错的点，并给出警示，例如，当使用 strcpy，sprintf 时，提示使用者注意字符串长度，建议使用 strncpy 和 snprintf 等。很多的编译器和 debug 工具也提供这样的功能。但是，这些方法只能在一定程度上降低栈溢出的可能性，而 C 语法的限制使得很难存在这样的工具来完全消除潜在的栈溢出。其次，进行数组边界检查，数组是栈溢出的潜在的问题，检出数组的边界，检查内存的越界访问，执行语言的类型安全检查同样能够防止栈溢出的发生，但是这些检查会影响程序的性能，降低程序效率，如果是在编译层进行检查，遗留系统将会难以兼容。再者，是代码指针检查，即在指针解引用之前检查指针是否被修改。

这里强调一种方法——金丝雀方法（canary）[13]。该方法能够侦测和预防很多栈溢出问题。Crispin Cowan 等[14] 在其论文中使用了两种金丝雀值，Terminator Canary 和 Random Canary，前者使用 0（null），CR，LF 和-1（EOF）作为 canary 值，截断字符串，防止字符串的溢出；后者使用随机值作为 canary 值，来保证该值的随机性和保密性，使得栈溢出很容易被发现。Canary 方法能够防止栈溢出，对 ROP 有一定的抑制作用，简单的写入 gadget 字符串，造成溢出，劫持程序流程变得不现实。PointGuard 方法是在 canary 方法上的延伸，在每个指针的附近放置 canary 值，检查每个指针是否被破坏，实施更为严格的检查。但是这些方法最大的问题在于程序的效率将会降低，占用更多内存，例如，如果 struct 结构中有 3 个指针，那么使用 PointGuard 方法保护的 struct 结构体占的内存空间将会增长一倍。而且对之前版本的程序的兼容性很差。即使不考虑 canary 的性能、占用内存空间和遗留系统的兼容性问题，canary 方法也是可以被攻破的。在 scut[15] 中，作者通过使用格式化字符串漏洞来构造格式化字符串，向内存的任意位置写数据。通过这种方法，可以将 gadget 数据写入到栈中，绕过 canary 的保护，进行 ROP 攻击。

接下来，从 ROP 攻击的代码片段的来源着手，ROP 程序关键一步是寻找以 ret 指令结尾的短指令序列。因此，最为直接的方法就是消除 ret 指令，通过编译阶段的特殊处理和优化，使生成的汇编指令序列的 ret 指令数量最小化，甚至完

4）ELF 文件头[13] 讲解了 ELF 文件的结构，以及程序链接过程，对于理解程序的重定位和 ROP 中 gadget 的指令序列的地址的形成有很清晰的理解作用。

全消除 ROP 中短序列的来源，防止 ROP 攻击。在 JinKu Li 等[16]的论文中，作者详细分析了 ret 指令的来源，包括实际的 ret 指令、非 ret 指令中的操作码片段、指令操作数中的立即数和操作数寄存器的编码，并针对每种情形提出消除 ret 指令的方法。对于实际的 ret 指令，作者使用间接返回(return indirection)的策略，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，函数调用返回时，通过该索引来获取实际的返回地址，这将有效保护返回地址，防止程序流程被篡改。对于操作数寄存器包含的 ret 指令，可以通过寄存器分配策略[10]的调整来消除。其余两类的 ret 序列，可以通过窥孔优化[10]的方法来消除，例如地址偏移量是 0xC3 时，可以在其间加入 NOP 指令，使地址偏移量变成 0xC4。作者的实验表明，用该思想设计的编译器编译出的代码不含 0xC3 指令，这些程序将很难被用于 ROP 攻击。这种方法将大幅降低 ROP 攻击的可行性，ROP 的短序列的图灵完备的计算特性将很难得到满足，这能够从根本上消除 ROP 攻击。但是，同样，这种方法将破坏计算机现有的体系结构，特别是内核的结构，并调用仍是本原的 return 指令，而新编译的程序则是间接 return 指令，这需要将其进行转换，从而导致程序内存访问的次数仍增加，开销增大，并且会造成遗留系统的不兼容，进而导致各种微妙的程序错误。同时，系统中未使用该方法编译的代码同样会使得 ROP 可行。但是可以通过一定程度的优化，在不破坏原有调用流程结构的情况下，尽量降低 ret 指令的密度，增加 gadget 的构建难度，这有助于 ROP 攻击的防护。

最后，需要说明地址随机化的问题。gadget 构建首要在于获取短序列的首地址，并将这些地址以数据的形式写入到栈中，用以控制程序的流程。如果每次都随机加载程序库(如 libc)，那么将使固定的 gadget 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 的论文[17]中，描述了针对地址随机化的方法和实际效果，并介绍了在 32 位机器上应对地址随机化问题的方法；在文献[13,18]中，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。分析认为，在 32 位机器上，地址随机化的效果不是很好。可以通过多种方法定位地址，这不会对 ROP 攻击造成很多束缚，但会增加构建者的难度；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 位机器上的细粒度的随机化，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安全性。

目前，也出现了一些其它的 ROP 的防御方法。包括基于控制流检查法[26,27]、基于程序动态特性的分析法[28,29]以及基于硬件的方法。这些方法都有自身的特点，但其实施依然存在很大束缚。

结束语 ROP 思想的提出虽然不是很久，但是其表现出的图灵完备特性和对于当前常见程序安全防范措施的针对性，使其具有巨大的研究和发展潜力。当前 ROP 及其变种 JOP 实现了一定程度的自动化，但是这样的自动化还停留在试验阶段和特定平台、特定指令集相关特性中，离真正的实际应用还有较大距离。本文总结了 ROP 发展以来的研究成果和发展思路，提出了 ROP 自动化发展下一步的方向和思路。同时，针对 ROP 攻击，介绍了常见的防护措施，并说明了这些防护措施的优缺点，指出了防范的原则。

对于 ROP 的研究始终是一个矛与盾的问题，更好地探索 ROP 的攻击能力，将有助于确保程序的安全。随着研究的发展和深入，在栈溢出、程序控制流完整性保护等方面的相关策略将会有助于防御 ROP 攻击，更好地保护计算机系统的安全。

参 考 文 献

[1] Designer S. StackPatch[EB/OL]. http://www.opwnwall.com/linux

[2] Documentation for the PaX [EB/OL]. Project: http://pax.grsecurity.net/docs/

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

[4] Return-to-libc Attack Lab[EB/OL]. http://www.cis.syr.edu/~wedu/seed/Labs/Vulnerability/Return_to_libc/Return_to_libc.pdf

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

[6] Shacham H. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls(on the x86)[C]// CCS'07 Proceedings of the 14th ACM conference on computer and communications security, 2007. New York, NY, USA: ACM, 2007: 552-561

[7] 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, 2008. New York, NY, USA: ACM, 2008: 27-38

[8] Turing A M. On Computer Numbers , with an application the Entscheifungs problem, 1936

[9] 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, 2009. CA, USA: USENIX Association Berkeley, 2009: 383-398

[10] Aho A V, Lam M S, Sethi R, et al. Compilers: Principles, Techniques, and Tools(2nd Edition)[M]. 赵建华，郑滔，戴新宇，译. 北京：机械工业出版社

[11] Lex[EB/OL]. http://dinosaur.compilertools.net/

[12] Flake H. Auditing binaries for security vulnerablities[EB/OL]. http://www.blackhat.com/presentations/bh-europe-00/HalvarFlake/HalvarFlake.ppt

[13] Bryant R E, O'Hallaron D R. 深入理解计算机系统(第 2 版)[M]. 龚奕利，雷迎春，译. 北京：机械工业出版社：450-473

[14] 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: 1119

[15] Scut/team teso. Exploiting Format String Vulnerabilities

[16] 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, 2010. New York, NY, USA ACM, 2010: 195-208

(下转第 23 页)

LNCS 6223. Berlin: Springer-Vedag, 2010: 314-332

[6] Canetti R, Halevi S, Katz J, et al. Universally composable password-based key exchange[C]// Proc. of Advances in Cryptology-Eurocrypt 2005, LNCS 3495. Berlin: Springer-Vedag, 2005: 404-421

[7] Abdalla M, Pointcheval D. A scalable password-based group key exchange protocol in the standard model[C]// Proc. of Advances in Cryptology-Asiacrypt 2006, LNCS 4284. Berlin: Springer-Verlag, 2006: 332-347

[8] Faster G R. shorter password-authenticated key exchange[C] // Proc. of TCC 2008, LNCS 4948. Berlin: Springer-Verlag, 2008: 586-606

[9] Groce A, Katz J. A new framework for password-based authenticated key exchange [EB/OL]. Cryptology ePrint Archive, Report 2010/147. http://eprint. iacr. org/ 2010 /147. pdf, 2010

[10] Katz J, Vaikuntanathan V. Round-optimal password-based authenticated key exchange [C] // Proc. of TCC 2011, LNCS 6597. Berlin: Springer-Verlag, 2011: 293-310

[11] 冯涛,马建峰. Y-SPH-OT 协议的安全性分析[J]. 计算机科学, 2006, 33(8): 125-129

[12] 李凤华,冯涛,马建峰. 基于 VSPH 的 UC 不经意传输协议[J]. 通信学报, 2007, 28(7): 28-34

[13] 冯涛,马建峰,李凤华. UC 安全的高效不经意传输协议[J]. 电子学报, 2008, 36(1): 17-23

[14] 冯涛,马建峰. 基于证人不可区分的通用可复合安全并行可否认认证[J]. 软件学报, 2007, 18(11): 2871-288

[15] 冯涛,李凤华,马建峰,等. UC 安全的并行可否认认证新方法[J]. 中国科学 E 辑:信息科学, 2008, 38(8): 1220-1233

[16] Raimondo M D, Gennaro R. New approaches for deniable authentication [J]. Journal of Cryptology, 2009, 22: 572-615

[17] Kiayias A, Zhou H-S. Zero-knowledge proofs with witness elimination[C]// Proc. of PKC 2009, LNCS 3494. Berlin: Springer-Verlag, 2009: 124-138

[18] Abdalla M, Chevalier C, Pointcheval D. Smooth projective hasing for conditionally extractable commitments [C]// Proc. of Advances in Cryptology-Crypto 2009, LNCS 5677. Berlin: Springer-Verlag, 2009: 671-689

[19] Cramer R, Shoup V. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack [C] // Proc. of Advances in Cryptology-CRYPTO 1998, LNCS 1462. Berlin: Springer-Verlag, 1998: 13-25

[20] Abe M, Gennaro R, Kurosawa K, et al. Tag-KEM/DEM: A new framework for hybrid encryption and a new analysis of Kurosawa-Desmedt KEM[C]// Proc. of Advances in Cryptology-EUROCRYPT 2005. LNCS 3494. Berlin: Springer-Verlag, 2005: 128-146

[21] Katz J, Ostrovsky R, Yung M. Practical password-authenticated key exchange provably secure under standard assumptions[C]// Proc. of Advances in Cryptology-Eurocrypt 2001, LNCS 2045. Berlin: Springer-Verlag, 2001: 475-494

[22] Dwork C, Naor M, Sahai A. Concurrent zero-knowledge [J]. J. ACM, Preliminary version in STOC'98, 2004, 51(6): 851-898

[23] Goldwasser S, Micali S, Rackoff C. The knowledge complexity of interactive proof systems [J]. SIAM Journal on Computing, 1989, 18(1): 186-208

(上接第 5 页)

[17] 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, 2004. New York, NY, USA: ACM, 2004: 298-307

[18] Seacord R C. Secure Coding in C and C++. Addison-Wesley, 2006

[19] 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), 2010. Washington, DC: USENIX Association, 2010

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

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

[22] 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, 2010. New York, NY, USA: ACM, 2010: 559-572

[23] 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), 2010. Washington, DC: USENIX Association, 2010

[24] Chen Ping, Xing Xiao, Mao Bing. Automatic Construction of Jump-Oriented Programming Shellcode(on the x86)[C]// ASIACCS '11. HongKong, China, March 2011

[25] Onarlioglu K, Bilge L, Lanzi A, et al. G-Free: Defeating Return-oriented Programming through Gadget-less Binaries[C]// ACSAC'10. NY, USA, 2010

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

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

[28] 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, 2011. New York, NY, USA ACM, 2011: 40-51

[29] 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