

代码复用对抗技术研究进展

乔向东¹, 郭戎潇¹, 赵勇^{2,3}

(1. 空军工程大学信息与导航学院, 陕西 西安 710077;

2. 清华大学计算机科学技术系, 北京 100871;

3. 江南计算技术研究所, 江苏 无锡 214083)

摘要: 代码复用攻击利用被攻击目标的固有程序代码实施攻击, 打破了恶意行为总是源自外部的传统假设, 是先进内存攻击技术的典型代表, 也是近期软件安全领域的研究热点。首先, 阐述了该技术的产生背景和实现机理; 其次, 从改进与变体、不同架构平台下的实现方法、自动化以及包括盲代码复用攻击和基于代码复用的非控制数据攻击在内的代码复用攻击技术的重要拓展等方面系统阐述了近年来国内外相关研究进展; 再次, 对代码复用攻击的防御机制和对抗防御机制的方法进行了讨论; 最后, 对代码复用对抗技术的发展趋势和研究方向进行了展望。

关键词: 软件安全; 代码复用; 返回导向式编程; 研究进展

中图分类号: TP309.2

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2018017

Research progress in code reuse attacking and defending

QIAO Xiangdong¹, GUO Rongxiao¹, ZHAO Yong^{2,3}

1. Information and Navigation College, Air Force Engineering University, Xi'an 710077, China

2. Department of Computer Science and Technology, Peking University, Beijing 100871, China

3. Jiangnan Institute of Computing Technology, Wuxi 214083, China

Abstract: Code reuse attacks make use of binary code existed in the attacked target to perform attack action, such technique breaks out the traditional assumption that malicious behavior always be introduced from the outside, it is representative sample of the advanced memory corruption techniques and also the focus of attention in the software security research field. The generation background and implementation principle were described firstly, and then the recent progresses of the technique, including improvement and variants, implementation methods under the different architecture platforms, automatic construction and important extension including blind ROP and non-control data attacks based on code reuse attacks, were introduced respectively. Various defense mechanisms and possible counter-defense methods for code reuse attacks were also discussed. Finally a perspective of the future work in this research area was discussed.

Key words: software security, code reuse, return-oriented programming (ROP), research progress

1 引言

自 20 世纪 80 年代起, 基于堆、栈溢出漏洞

利用和程序控制流劫持的代码注入攻击对网络信息安全构成了严重威胁。究其原因, 本质上是由于冯·诺依曼架构平台进程内存中的数据

收稿日期: 2017-12-23; 修回日期: 2018-01-26

通信作者: 乔向东, qxdcm@sina.com

基金项目: 国家自然科学基金资助项目 (No.61672061)

Foundation Item: The National Natural Science Foundation of China (No.61672061)

指令共存于一个逻辑地址空间,易造成数据和指令的混淆,使注入的数据可被当作指令执行,这也被认为是造成计算机安全问题的重要根源之一。为对抗代码注入攻击,21 世纪初以来,包括 Intel、AMD、微软在内的计算机软、硬件厂商陆续提出了各自基于软件或硬件实现的数据执行保护机制。Intel 和 AMD 分别在其处理器中增加了 XD(eXecute Disable)和 NX(No eXecute)保护开关,通过内存页面标记,使内存空间中不存在既可写又可执行的页面,以实现 $W \oplus X$ 保护;微软在 Windows XP SP2 版本之后的系统中新增了数据执行保护开关 (DEP, data execute prevention)^[1];Linux 操作系统也对应 PaX 补丁包项目^[2]。这些保护机制使恶意代码即便注入也无法执行,从而很大程度上抑制了代码注入攻击。

早在 20 世纪末,尽管当时软硬件厂商还未提出完整的数据执行保护机制,研究人员已经开始设想复用被攻击目标中的已有代码来帮助攻击者完成特定操作^[3],并于 1997 年首先提出了复用 libc 函数库(一种最为常见的基础函数库)中的函数二进制代码来实施攻击的方法,即 Return-into-libc 攻击^[4~6]。利用该方法实施攻击时,攻击者无需注入恶意代码(尽管此类情况极少),只需在攻击载荷中提供必要的 libc 库函数的地址,在劫持目标程序控制流后即可使被攻击目标程序按照攻击者的意图执行特定操作,从而绕开数据执行保护机制。然而,由于复用的是函数二进制代码,Return-into-libc 攻击也存在一些缺陷:1) 只能顺序调用函数实施线性攻击,无法实现跳转及循环,不具备图灵完备的计算能力,攻击能力有限;2) 完全依赖于系统的库函数,移除或修改库中的函数将严重限制攻击能力^[7]。

返回导向编程技术 (ROP, return-oriented

programming) 是 2007 年由 Shacham^[8,9]提出的代码复用攻击技术,其核心思想是利用目标设备程序中可实现特定操作的、以 ret 指令结束的指令序列片段 (Gadget),而非完整函数二进制代码,帮助实现目标设备的执行恶意操作。一般而言,单个 Gadget 并不具有恶意语义,但通过 ret 指令将多个 Gadget 串联起来是可以完成某种恶意操作的,因此一次攻击往往需要跳转执行若干个 Gadget。通常情况下,ROP 攻击中需将攻击所需各个 Gadget 的地址以及相关数据按照精心设计的格式发送至目标程序存在安全漏洞的函数栈空间中,以确保漏洞触发后程序控制流发生跳转并转而执行 Gadget,最终实现恶意操作。图 1 给出了基于栈空间控制流劫持的 ROP 攻击原理示意,攻击指令序列是由 5 个指令序列所含 ret 指令串联而成。

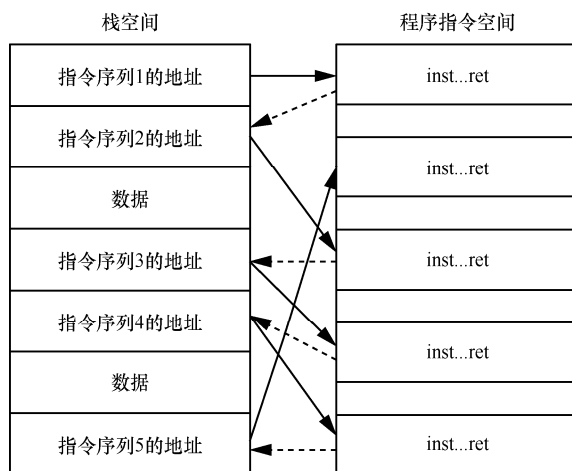


图 1 ROP 攻击指令序列示意

Shacham 证明了只要短 Gadget 足够多,ROP 攻击即可具备图灵完全的计算能力。特别地,目标设备处理器指令集中的指令越密集,越有利于代码复用,这是因为通过对已有指令字节序列的偏移解码,有极大概率得到具有新语义的有效指令序列。通过改变指令执行的起点,图 2 和图 3

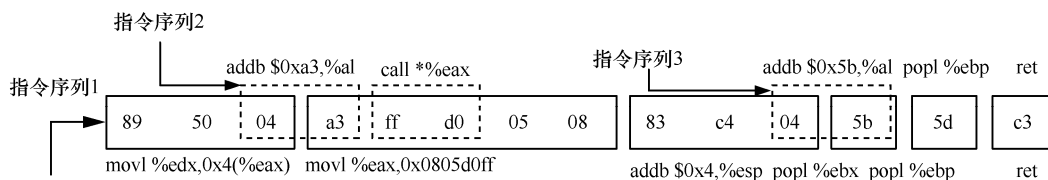


图 2 指令序列提取示意 (14 个字节 X86 平台程序片段)

分别给出了对应 X86 平台指令字节序列和 Z80 单片机 BIOS 指令字节序列的 Gadget 提取示意。从图 2 可看出,从仅有 14 个字节的指令字节序列中可得到 3 个 Gadget,其中 Gadget 1 是原有的指令序列,即有意代码段;而 Gadget 2 和 Gadget 3 则是具有新语义的指令序列,即无意代码片段。图 3 显示仅从 4 byte BIOS 指令字节序列中提取到 1 个无意指令序列。在代码复用攻击中,许多 Gadget 都是由无意代码片段组成的。通常可从规模较大的程序或函数库(如 libc、glibc)提取到具有各种语义的 Gadget,由这些 Gadget 可组装出具有内存读取/存储、算术和逻辑运算、流程控制、系统调用、函数调用 5 大功能的 Gadget 集合,从而实现图灵完全的计算能力。

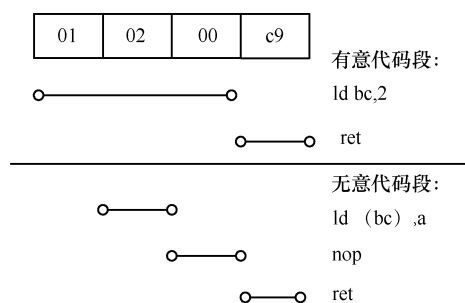


图3 指令序列提取示意 (4 byte Z80 单片机 BIOS 程序片段)

与 Return-into-libc 相比,ROP 攻击的代码复用粒度更细、针对性更强、应用更为灵活,减小了攻击载荷的长度,提高了攻击载荷的利用效率,并已成为主流的代码复用攻击技术。以 ROP 为代表的代码复用攻击技术,在建立新的攻击作用机制的同时,也使因数据保护机制而一度处于困境

的代码注入攻击重新焕发了生机。

2 代码复用攻击技术研究进展

2.1 ROP 的改进及变体

由于 ROP 攻击复用了单独的、以 ret 结尾的指令片段,使代码中的 call 和 ret 指令不匹配(正常程序中两者存在一一对应关系),加之通常会将多个 ret 结尾的指令片段链接为 ROP 攻击 Gadget 链,这些明显的特征使其容易被检测到。为此,研究人员对 ROP 攻击进行各种改进,提出了 ROP 攻击技术的各种变体。这些 ROP 变体的一个共同特征是,其 Gadget 调度和复用机制更加灵活多样且特征不再凸显,攻击更加隐蔽,从而提高了抗攻击检测的能力。

2010 年,Checkoway^[10,11]提出采用具有相同功能的 pop-jmp 指令序列来替代 ret 指令,但由于 pop-jmp 指令序列同样具有很鲜明的特征,并未从本质上解决 ROP 的缺陷,通过对指令序列特征的分析,如语义、频率等,还是很容易检测到攻击行为,改进的意义不大。早在 2003 年,专注于 Linux 内核安全研究的 PaX 项目组就提出采用 jmp 间接跳转指令构建代码复用攻击的想法^[12]。2011 年,Bletsch^[13]提出跳转导向编程攻击方法(JOP, jump-oriented programming),以 jmp 或 call 间接跳转指令结尾的指令片段构造功能 Gadget,并使用调度 Gadget 分发控制流,使具体的功能 Gadget 得以执行,每次功能 Gadget 执行完毕后都会跳回调度 Gadget,继而执行下一个功能 Gadget。图 4 给出了 JOP 的攻击原理模型。与 ret 指令相比,jmp

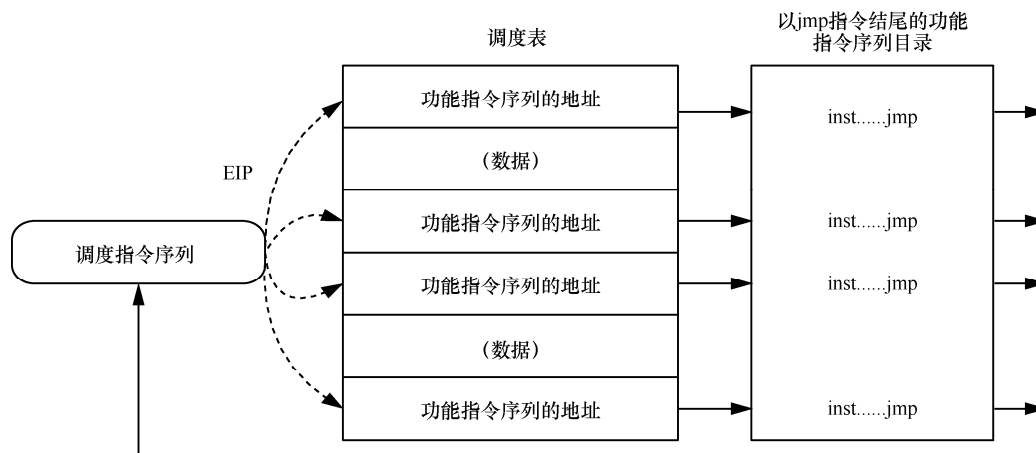


图4 JOP 攻击的原理

指令具有使用位置灵活、形式多样（间接跳转和直接跳转均可）及不依赖栈空间的特点，不仅可以摆脱栈的限制，且由于没有特定的使用形式和跳转的目标地址，使对攻击行为的辨识较为困难。

JOP 虽然消除了对 `ret` 指令的依赖，但存在其他缺陷。首先，指令序列所包含的 `call` 指令，并无相应 `ret` 指令与其匹配，使用栈保护技术可以识别；其次，JOP 具有寄存器依赖性，在某个 Gadget 中使用寄存器会影响到后续其他 Gadget 对该寄存器的使用，从而使其攻击能力受到限制，因而 JOP 的实用性并不高。陈平^[14]提出了 JOP Gadget 搜索方法及图灵完备的 JOP Gadget（包括调度和功能 Gadget），并实现了自动构建 JOP Shellcode 的工具，解决了 Gadget 和内存布局等问题。邢晓^[15,16]提出了一种分支指令导向攻击技术（BIOP, branch instruction-oriented programming），使用间接转移指令（如 `jmp` 指令或 `call` 指令）结尾的短指令序列传递控制流，由包含一个 `call` 指令的 Gadget 和一个含 `ret` 指令的 Gadget 组成的联合指令序列负责调用复用的 Gadget。由于并未引入新的指令序列特征，BIOP 可有效避免现有的防护技术，特别是解决了攻击构造时的内存冲突，避免了寄存器副作用对攻击的影响，实现了自动化的攻击构造。图 5 比较了 ROP 与 BIOP 的控制流跳转过程。

传统上，要在 ROP 的功能代码中实现条件转移逻辑是一件较为复杂的事情。针对 X86 架构下的 ROP 攻击，Shacham 给出了间接实现条件跳转的一种思路，需要串接多达 11 个不同的 Gadget，构造过程中若缺少某些类型的 Gadget 或不能消除 Gadget 之间的副作用，都会使构造失败。为此，袁平海等^[17]研究指出，尽管大部分条件判断指令

无助于条件转移逻辑的实现，但仍有少部分条件判断指令每个分支的开始部分是可复用的经典指令序列且从不同的内存单元中取得下一个指令序列的地址，从而可以帮助 ROP 实现条件转移逻辑，并将这种以条件判断指令开始的代码片段称为 if-Gadget（包括 I、II、III 型），从而大大简化 ROP 中条件转移逻辑的实现，且在 Linux 和 Windows 上的实验结果表明，if-Gadget 在软件模块中普遍存在，从而进一步证实了攻击者可在可执行程序中构造纯 ROP 代码攻击，ROP 攻击对现实系统的威胁比原来认为的严重得多。

2.2 不同架构平台的实现

尽管初始提出 ROP 攻击时主要针对的是 X86 平台，但近年来研究人员相继在 SPARC、PowerPC、Z80、ARM、Atmel AVR 等不同处理器平台下实现了 ROP 攻击，大大拓展了 ROP 攻击的适用范围。

相较 Intel X86 平台的 CISC 架构，ARM 平台的 RISC 架构在指令集及实现方式上存在很大的差异。不同于 Intel X86 的不定长指令，ARM 指令是 32/16 bit 定长指令，且是 4/2 byte 对齐的。ARM 没有 `ret` 指令，但它提供对程序寄存器 PC 的直接操作，任何一条写 PC 寄存器指令都可以当作跳转指令来执行。2009 年，Kornau^[18]首先提出了针对 ARM 平台的 ROP 攻击方法，证明了通过复用应用程序中可执行代码和链接库代码能够实现图灵完全计算；2010 年，Davi^[19]采用间接跳转指令（`BLX reg`）和跳板块实现了针对 ARM 平台的 JOP 攻击，图 6 给出了攻击流程及步骤。钱逸^[20,21]在借鉴 Kornau 所提平台无关的（基于 REIL 语言（reverse engineering intermediate language））

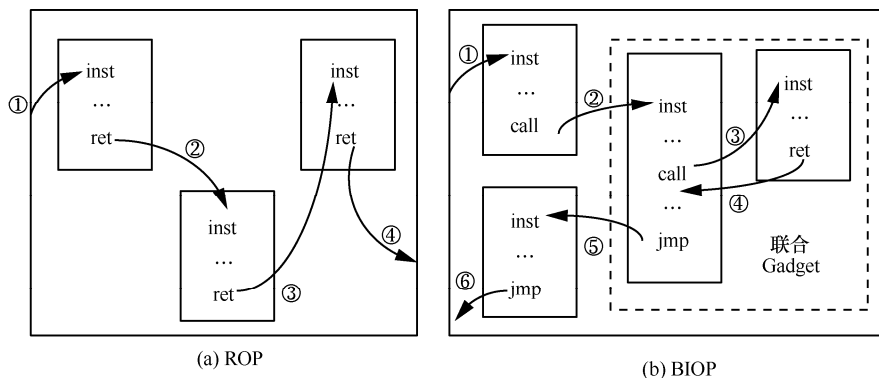


图 5 ROP 与 BIOP 控制流跳转的比较

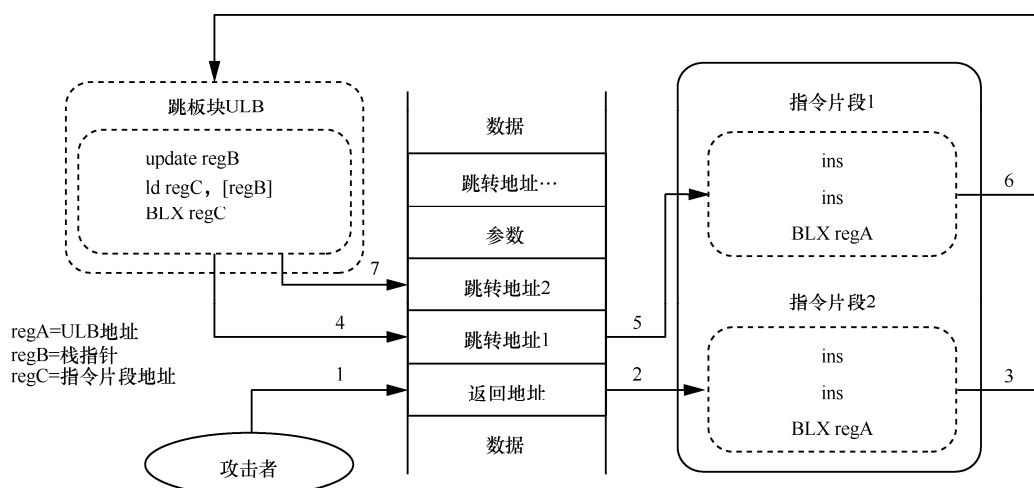


图6 针对 ARM 平台的基于 BLX 间接跳转指令的 JOP 攻击流程

自动搜索算法和 Shacham 所提搜索算法的基础上, 提出了针对 ARM 架构平台的攻击指令序列的自动搜索算法。

与冯·诺依曼架构处理器不同, 哈佛 (Harvard) 架构处理器的指令和数据各自享有独立的存储器和访问总线, 不允许指令和数据并存。由于程序和数据内存空间相互隔离, 也使被当作数据接收的、存储于数据内存空间的指令很难得到执行的机会。由于哈佛架构处理器的特殊内存布局, 传统上认为对其实施内存注入攻击是不可能的。针对采用哈佛架构 Atmel AVR Atmega 128 处理器的无线传感器网络节点, Francillon 成功实现了针对传感器节点内存的永久代码注入攻击, 其核心研究思路是两阶段注入, 即首先基于 ROP 技术将恶意数据注入数据内存空间, 其次借助 ROP 技术将包含在恶意数据中的恶意代码从数据内存空间注入指令内存空间并执行, 这其中会利用到该处理器指令集中的个别特殊指令^[22]。由于无线传感器网络节点所采用的 IEEE 802.15.4 协议数据分组大小有限且注入的代码量较大 (如蠕虫攻击需增加自传播模块), 两阶段注入过程均需综合利用程序控制流恢复及大伪栈技术, 以便循环利用漏洞完成大量数据的注入, 图7给出了相关示意。该项研究对于针对嵌入式设备的代码注入攻击开辟了新的路径。

Checkoway 等^[23]利用 ROP 攻击对采用 Z80 单板机的美国大选投票机实施了攻击, 整个攻击过程并未替换安装有选举程序的系统只读存储器, 却成

功改变了投票机存储的投票结果。需要指出的是, 该投票机的所有程序均存储在 3 块 64 kB 的 EPROM 只读存储器中, 这一研究成果打破了有关嵌入式设备固件程序锁定后即无法对其实施攻击的传统认识。此外, Buchanan^[24]提出了针对 SPARC 架构 (RISC 微处理器架构之一) 的微处理器的 ROP 攻击实现方法; Lidner^[25]则针对采用 PowerPC 微处理器的嵌入式设备, 提出了 ROP 攻击的实现方法。

2.3 代码复用的自动化

ROP 攻击的核心在于分析目标程序代码、获取所需的 Gadget, 利用所得指令序列构建攻击 Gadget 链, 以完成特定操作。一般而言, 实现上述过程既复杂又繁琐, 传统手工分析设计效率低也容易出错, 因此, 近年来 ROP 攻击研究的一个重要方向就是自动化, 主要围绕 Gadget 的自动化搜索和自动化组装, 目的是为了快速、高效地找到实现特定功能逻辑的 Gadget。

针对 Gadget 的自动获取, Shacham^[8]提出了被称为 Galileo 算法的 Gadget 搜索算法, 其核心思想是通过反汇编二进制程序映像, 以递归反向遍历的方式, 在目标程序中搜索以 ret 指令结束的指令序列, 直至找到所有以 ret 指令结束的指令序列。公开的 Gadget 搜索工具, 如 ROPGadget 和 ropshell 均是基于 Galileo 算法实现的, 但该算法仅适用于 X86 平台下以 ret 指令作为控制流转移指令的 Gadget 搜索。然而, 即便是在 X86 平台下, 用于 Gadget 间控制流转移的指令也不只有 ret 指令, 还包括 call 指令, 更何况其他架构平台还有其特

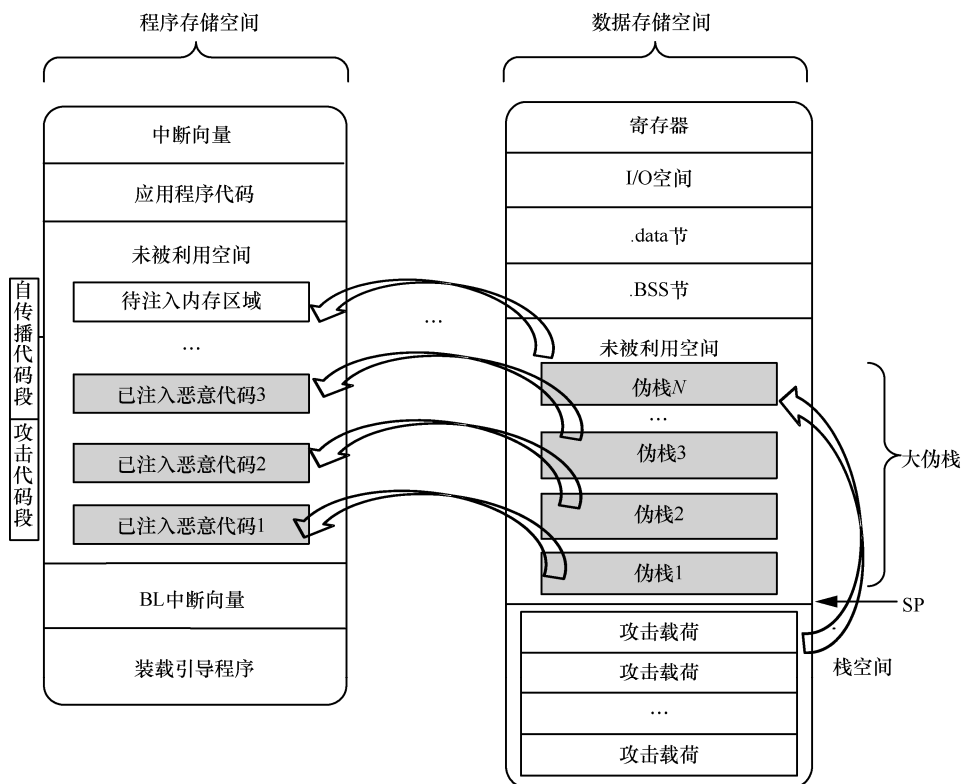


图 7 针对哈佛架构无线传感器节点的代码注入攻击

有的指令,如 ARM 平台下的 bx REG 指令序列控制流转移指令。彭国军等^[26]指出一种更加通用的搜索方式应根据代码复用的具体方式,构建关于可能的控制流转移指令的集合,以集合中的每一个指令作为匹配对象来搜索全部潜在的 Gadget。

关于 Gadget 的自动化组装(编译)是从搜索到的 Gadget 集合中选取适当的 Gadget,并按照适当的方式将其拼接组织起来形成链条,以实现所需的功能逻辑,其实现原理是在使用中间语言描述 Gadget 的基础上利用程序最弱前置条件方法对 Gadget 指令序列进行分类,然后基于编译器中的指令匹配原理从不同功能类别的 Gadget 集合中选择能够匹配用中间语言描述的功能逻辑需求的 Gadget。Buchanan、Komau、Hund、Lez^[18,24,27,28]在上述领域开展了深入的研究,相关的研究成果包括基于 REIL 的 Gadget 自动化搜索框架和基于 Qool 的 Gadget 链自动化生成框架 Q,但距离完全意义下自动化代码复用攻击仍有很大的差距,难点就在于如何优化地选择指令序列以实现特定功能逻辑。

针对 JOP 攻击的自动化构造,邢晓首先给出了一种针对 JOP 的 Gadget 快速搜索算法,并提出

了自动构造 JOP 攻击 shellcode 的一般方法和充分条件;为解决 Gadget 寄存器使用冲突的问题,邢晓还提出了 JOP Gadget 的排列算法。其所设计的 JOP 攻击自动化构造工具由指令序列采集器、指令序列排列器和内存优化器 3 部分组成,可构建实现各种语义的 shellcode^[15]。常规 ROP 攻击会频繁调用 ret 指令结尾的短指令序列,这也成为 ROP 攻击防御检测的一个重要判据,有的检测方法会将连续 3 次长度小于 5 的指令片段调用视为发生了 ROP 攻击,采用长度更长的 Gadget 将使攻击更为隐蔽和不易被检测。为此,杨超^[29]对基于长指令序列的 ROP 攻击自动化进行了研究,在确保图灵完备的前提下降低了长指令集合的寄存器依赖度,并通过实验验证了基于长指令序列自动化填充和链接的 ROP 自动化攻击的有效性。研究人员还就 ROP 攻击的简化实现开展了深入研究,Long^[30]提出先利用一小段 ROP 攻击代码关闭 ASLR 及 DEP 保护,并将恶意代码复制到指定的内存空间,继而篡改控制流执行以实现恶意代码的注入。Payer^[31]的做法则是首先利用格式串漏洞关闭 ASLR 及 DEP 的保护,继而实施 ROP 攻击。

简化的 ROP 攻击方式避免了繁琐的构造攻击的步骤和时间,大大提高了 ROP 的攻击能力。

2.4 其他重要拓展:盲攻击和基于代码复用的非控制数据攻击

利用 ROP 攻击方法需事先获得目标设备固件映像(至少是被攻击程序的二进制代码),这一条件并非容易满足。2014 年, Bittau 等^[32]提出了无需事先掌握目标程序的二进制代码的情况下对其实施 ROP 攻击的盲 ROP 攻击方法(BROP, blind ROP)。该方法利用服务软件(如 nginx、MySQL、Apache、OpenSSH、Samba 等)进程崩溃后会自动重启的特性,向该服务发送精心设计的探测请求数据分组,通过服务软件的返回响应所泄露的信息,逐步探测确定所需 Gadget 的地址(包括扩展的栈空间读取 Gadget、可使目标程序暂停的 stop Gadget、潜在可利用的 Gadget 以及 BROP Gadget 等),最终实现被攻击目标设备程序映像的远程 dump 操作,继而针对目标实施常规 ROP 攻击。在满足一定条件的前提下, BROP 攻击能够突破 ASLR、程序装载地址随机化(PIE, position independent executables)及堆栈检测等多种保护机制,对 32/64 bit Linux 系统中的服务应用实施 ROP 攻击。表 1 给出了 BROP 针对开启 ASLR、NX 及 canaries 栈保护的 64 bit Linux 系统中 3 种不同服务的攻击所需发送请求数目和时间,表 2 给出了 BROP 攻击各阶段累积发送的请求数目,从中可见针对源代码和二进制代码均闭源的轻量级私有服务, BROP 仅发送了 1 950 个探测请求后就成功对其实施了攻击,历时也仅 5 min。有关 BROP 的研究工作得到了 DARPA 的资助。目前, BROP 仅能针对 Linux 系统开展盲攻击,还不能适用于 Windows 系统;此外, BROP 假设被攻击目标和进程能够容忍多次的恶意连接尝试,这在现实中也还是较难满足的。

表 1 BROP 的攻击请求总数和耗时

被攻击服务程序	请求数目	时间/min
nginx	2 401	1
yaSSL + MySQL	3 851	20
轻量级私有服务	1 950	5

表 2 BROP 各攻击阶段的累积请求数目

攻击阶段	轻量级私有服务	yaSSL + MySQL	nginx
扩展的栈读取	1 028	406	846
搜寻 PLT 表	1 396	1 454	1 548
搜寻 BROP 指令序列	1 565	1 481	2 017
搜寻 strcmp 指令	1 614	1 545	2 079
搜寻 write 指令	1 624	1 602	2 179
程序映像远程 dump 及后续漏洞利用	1 950	3 851	2 401

鉴于针对控制流劫持的防御机制日渐丰富和完善,早在 2003 年就有学者提出了针对非控制数据的攻击(non-control data attack),非控制数据主要是指控制流转移指令不直接使用的内存变量。传统上一般认为非控制数据攻击的攻击载荷仅能实现有限的语义。例如,仅能破坏和泄露少量字节的安全敏感数据。2016 年,新加坡国立大学的 Hu 等^[33]提出了一种基于代码复用的非控制数据攻击技术,也称 DOP(data-oriented programming),该技术通过利用与内存错误相关的漏洞,攻击非控制数据(如函数局部变量、全局变量、函数参数等),继而复用搜索到目标程序中的 Gadget 和 Gadget 调度器,完成对目标程序的攻击。DOP Gadget 与 ROP Gadget 的差异主要有两点:一是 DOP Gadget 采用某一地址的内存空间作为虚拟寄存器,由虚拟寄存器保存并传递 Gadget 的计算结果,ROP Gadget 则可以采用内存或硬件寄存器存储计算结果;二是由于数据攻击并不破坏程序控制流程,DOP Gadget 的执行必须依据合法的程序控制流程执行,并且不必一个 Gadget 接着一个 Gadget 相继执行,而 ROP Gadget 的调用不必依据程序控制流程执行,可以在任何需要的时候执行。因此,通常 DOP Gadget 较 ROP Gadget 要求更严苛。DOP Gadget 调度器负责完成 DOP Gadget 的调度执行,通常由目标程序中的循环体以及对应的选择器(一般由内存错误所控制)构成,并以前次循环迭代中所执行特定 DOP Gadget 的输出作为输入,每次循环迭代会依据选择器的值选择执行特定的 Gadget,直至无法满足选择器的要求而循环终止,图 8 给出了 DOP 攻击模型的示意。Hu 证明了 DOP 具备图灵完备的计

算能力,大量实验结果也表明 DOP Gadget 和调度器的泛在性以及 DOP 攻击突破 ASLR 和 DEP 防御机制的能力。DOP 的最大优势为在不破坏目标程序控制流的前提下对其实施攻击,而其主要挑战在于 DOP Gadget 和调度器的辨识以及 DOP 指令序列的拼接组装,Hu 虽然给出了相关算法但其执行效率还需要评估;此外,目前 DOP 的攻击对象仅限于 X86 程序,并不能适用于其他指令架构程序的攻击。关于针对 DOP 攻击的防御,Hu 等认为全覆盖的内存安全措施、全覆盖的数据流完整性保护 (DFI, data-flow integrity)、非控制数据存储空间的细粒度随机化等措施有助于缓解甚至消除 DOP 攻击,但显然这些措施也会带来极大的计算负荷,影响系统运行性能。

3 代码复用攻击的防御机制及其对抗

随着代码复用攻击技术的发展,其攻击能力越来越强,威胁也日趋严重,研究人员开始考虑针对代码复用攻击的防御机制,并提出了不少研究成果。另外,为对抗这些防御机制,如何让代码复用攻击更为隐蔽和难以检测也成为重要的研究方向。围绕代码复用攻击,一场攻防博弈也就此展开。

3.1 防御机制

总体上讲,迄今提出的代码复用攻击防御机制可以划分为三大类:一是通过重编译从目标程序中移除可被利用的 Gadget 或者强化目标程序的控制流完整性 (CFI, control flow integrity);二是采用运行时保护技术为目标二进制程序提供透明保护^[14];三是基于随机化的防护方法。

就基于重编译的防护机制而言,其目的是通过定制的编译器重新编译目标程序,以使编译输出的二进制文件中没有可被 ROP 攻击利用的指

令片段。例如, Li 等^[34]设计实现了一个完全不包含 ret 返回指令的系统内核,内核中所有指令中的 C3 字节 (ret 指令的操作码) 全部被移除,需要通过查表才能找到有效的返回指令, Onarlioglu 等^[35]学者研发的 G-Free 编译器可完全移除目标程序的无意返回指令并加密返回地址。基于重编译的防护机制的缺点是需要获得目标程序的源代码。

基于控制流完整性的防护机制主要通过控制流图来保护关键的控制流对象或者代码不被篡改,包括跳转、调用及返回指令必须遵从事先通过源码分析、静态分析或运行时 profiling 获得的程序控制流图。已有的控制流完整性保护工具包括 Program Shepherd、StackGuard、XFI、SFI、Hypersentry 等^[15]。由于较难获得细粒度的程序控制流图,多数情况下该防御机制被简化为返回指令必须是 Call-preceded 的,以及间接跳转指令只能跳转至函数起始处的约束。2014 年, Goktas 和 Carlini^[36,37]经过独立研究后得出了相同的结论,即基于控制流完整性的防护机制并没有想象得强。2015 年, Otgonbaatar^[38]指出,即便获得了细粒度的控制流完整性,也并不能有效对抗攻击。王明华等^[39]学者提出一种新的面向二进制的细粒度 CFI 保护方案 BinCC,通过代码复制和静态分析,该方案将二进制代码分成几个互斥代码块,继而将代码中的各个间接转移归类为块间转移或块内转移,并分别应用严格的 CFI 策略来限制这些转移,通过减少合法转移目标,从而增加了攻击者绕过 CFI 限制实施 ROP 攻击的难度。当前有关代码复用攻击检测的研究主要关注的是针对应用层的攻击检测,针对内核的攻击检测方法较少。为此,陈志锋等^[40]提出了一种基于细 (函数) 粒度 CFI 的 Linux 内核代码复用攻击检测方法,

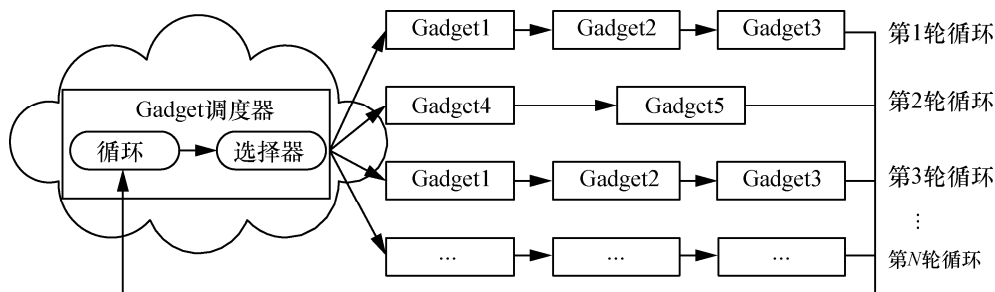


图 8 DOP 原理示意

其核心思路是针对与控制流相关的 `call`、`ret` 和 `indirect jump` 指令, 提出对应的 CFI 约束规则, 基于编译器插桩技术和 `vmcall` 指令生成并引入 CFI 标签指令, 在 Hypervisor 中执行的 CFI 标签指令会根据标签编号和状态属性验证 3 种指令使用的合法性, 实验结果表明, 该方法能够有效检测内核级代码复用攻击, 但对于 IO 密集作业时的性能开销较大。另外, 使用代码插桩技术将检查代码插入二进制程序中检测代码复用攻击不仅会破坏二进制文件的完整性, 通常也会带来较大的性能损失。

运行时保护是根据代码复用攻击 Gadget 的可能特征阻止针对正在运行的、被保护程序的攻击。例如, ROPecker 发现目标程序存在连续的 (10 个以上)、以 `ret` 指令结尾的 Gadget 调用且每个 Gadget 的长度均小于固定门限时, 就会关闭该程序^[37]。改进版的 ROPecker 的检测规则是通过过去 T 次检测过程中疑似攻击指令序列的执行次数来判断是否发生了 ROP 攻击^[41]。希腊学者 Papps 等^[42]开发的 kBouncer 则会在程序执行系统调用时停止程序执行, 并基于 Intel 处理器提供的最近指令执行分支记录 (LBR, last branch record) 的历史数据, 检测程序最近 16 次间接分支执行历史, 依据检测结果要么允许程序运行, 要么关闭程序。Davi^[43]通过构建影子栈开发了 ROP 防御工具 ROPdefender, 通过验证所有返回地址是否位于影子栈中判断是否发生了 ROP 攻击。

恶意代码攻击过程中通常利用程序中的假设不变量 (如地址、代码、数据、系统调用等) 进行攻击, 为此研究人员提出了程序随机化防御机制, 具体表现为 4 类随机化方法, 包括地址随机化、代码随机化、数据随机化和操作系统接口随机化^[15]。ROP 攻击的一个重要前提是假设程序中 Gadget 的地址是不变的, 因而通过改变程序内存空间的地址以防止 ROP 利用硬编码地址实施攻击成为 ROP 攻击的主要方法之一, 包括 ASLR、地址空间布局置换排列 (ASLP, address space layout permutation) 等。地址随机化现已被常用操作系统采用, 但由于并非对程序内存空间的全部地址进行随机化 (有的仅对堆栈和动态链接库地址进行随机化处理), 利用内存信息泄露漏洞或

暴力破解方式可以获得随机化之后的指令序列地址, 因而近年来绕过地址随机化保护的 ROP 攻击也不断出现, 包括前述 BROP 中提出的、扩展的栈读取技术。为此, Davi、Wartell 及 Pappas 等^[44-46]提出通过函数顺序重排列、指令顺序重排列、等价指令替换和寄存器重赋值等细粒度的代码随机化技术来阻止 ROP 攻击。

3.2 对抗防御机制

为了对抗针对代码复用攻击的各种防御机制, 研究人员也不断对以 ROP 攻击为主的代码复用攻击技术进行改进, 通过弱化攻击特征、消除历史痕迹等方法, 使攻击更加隐蔽和不易检测。2014 年, Carlini 和 Wagner^[37]提出了 3 种突破 ROP 攻击防御的基本方法, 即 Call-preceded ROP 攻击、规避攻击、历史刷新, 并阐述了如何综合利用上述方法构造可突破 ROPecker 和 kBouncer 这 2 种常见防御工具的 ROP 攻击。

对于一个正常的构造良好的程序, `ret` 指令和 `call` 指令总是成对出现, 且每个 `ret` 指令总是返回到一个紧跟在 `call` 指令之后的指令处 (Call-preceded 的具体含义), 而 ROP 攻击则背离了这一模式, 因而也成为 ROP 攻击检测和防御的重要判决规则。然而, Carlini 的研究表明, 即便所有 `ret` 指令服从该模式, 只要允许攻击指令序列足够复杂 (长且包含直接或间接跳转指令), 仍然可以实施 ROP 攻击, 即 Call-preceded ROP 攻击。实验中, Carlini 复用 70 kbyte 二进制文件中的代码实现了 Call-preceded ROP 攻击^[37]。

基于运行时检查的 ROP 攻击防御机制常会将程序指令片段辨识为 Gadget 或非 Gadget 指令序列, 而最常用的辨别依据就是指令片的长度, 较短的指令片段往往被识别为 Gadget, 较长的指令片段则被识别为非 Gadget 指令序列。如果防御机制在一定时间内检测出多个 Gadget, 就会判定发生了 ROP 攻击。一种绕过基于指令片段长度防护机制的方法就是采用长指令片段完成特定功能, 杨超^[27]对基于长指令序列的 ROP 攻击进行了深入研究, 通过实验验证了基于长指令序列的自动化 ROP 攻击方案的有效性。Carlini^[37]的研究进一步表明, 混合使用长、短指令序列的方式也可以实施 ROP 攻击, 并能绕过多种公开的防御机

制,这也就是 Carlini 所提的规避攻击。

除指令片段长度外,基于运行时检查的防御机制也常常会在程序执行时通过检查程序指令执行的历史数据(如 Intel 处理器支持的 LBR 数据)判断是否发生了 ROP 攻击。出于性能考虑,这种检查也只是在特定时刻进行(如发生系统调用时)。由此,Carlini 提出在防御机制对应的检查进程未启动时执行 Gadget,并在检查进程启动前周期性地执行无意义的指令序列以清除 ROP 攻击的痕迹,而在检查进程执行时不执行任何 Gadget 并继续执行无意义操作,在检查进程检测完毕后继续推进 ROP 攻击,这样检查进程就看不到任何 ROP 攻击的历史痕迹^[37],图9给出了相关原理说明。

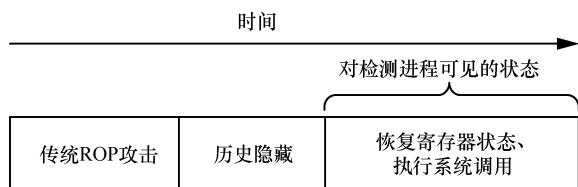


图9 历史刷新(隐藏)ROP攻击

4 结束语

代码复用对抗是近期软件安全领域的研究热点,且已取得了一大批有影响力的研究成果。随着研究的进一步深入,相信代码复用攻击技术依然会保持旺盛的生命力,有关攻防技术的博弈依然是软件安全理论研究的热点。特别地,攻防技术两端长期呈现的螺旋式发展态势,有关防御机制的研究依然将推动代码复用攻击技术自身的发展,并在提升自身能力、拓展应用领域的同时,不断提高技术实现的自动化和智能化程度。在今后研究中,有以下几个方向值得关注。

首先,在已提出的各种代码复用技术中,BROP突破了传统ROP攻击需事先获得被攻击目标的程序映像的限制,实现了盲代码注入攻击,在极大提升代码复用攻击能力的同时,也为今后针对源代码和二进制代码均未知的私有服务程序的攻击技术研究提供了新思路,值得持续开展深入研究,其研究成果有可能为今后具备初步平台自适应能力的代码复用攻击技术提供方法基础。

其次,基于非控制数据攻击的代码复用攻击

由于未改变程序即有的控制流程,包括CFI在内的代码复用攻击防护机制对于该类型攻击并不有效。已证明该类型攻击具备图灵完全计算能力,但该类型攻击较为复杂、实现难度较大。如何降低基于非控制数据攻击的代码复用攻击的实现复杂度以及将基于控制数据攻击和非控制数据攻击的代码复用攻击相结合是可供探索的方向。

从已发表的文献看,代码复用攻击中所涉及的代码复用源种类十分广泛,从传统的Intel X86、ARM平台下的代码,到Z80单板机的代码,再到无线传感器节点中的代码,涉及现有绝大多数通用微处理器,但尚未看到有针对数字信号处理器(DSP, digital signal processor)代码的验证性代码复用攻击,这或许和DSP主要运行数字信号处理算法代码而非控制程序代码有关。从理论上讲,复用DSP代码并非不可行,但由于控制代码复用和运行复用的代码涉及不同的指令集合的处理器(前者一般为通用处理器,后者为专用处理器),实现难度会很大,但基于新的代码复用源的代码复用攻击技术仍然十分令人期待。

最后,以ROP为代表的代码复用攻击技术的提出,使关于“好代码”和“坏代码”的界定变得模糊,因为那些看起来即便是“好的”代码也可被用来干坏事。通过前述的梳理分析可以看出,无论是基于重编译还是基于运行时保护的防护机制还不能完全阻止代码复用攻击的发生,攻击方总是想方设法突破各种防护机制,尽管这样会大大增加攻击实现的难度和代价。由此,借鉴深度学习和人工智能的最新研究成果,探索将其用于代码复用攻击检测及防护领域的可能,也将是具有较高理论和实践价值的研究方向。

参考文献:

- [1] MICROSOFT. A detailed description of the data execution prevention (dep) feature in windows xp service pack 2[EB/OL]. <http://Vsupport.microsoft.com/kb/875352>.
- [2] THE PaX TEAM. PaX project[EB/OL]. <http://pax.grsecurity.net/docs/randexec.txt>.
- [3] DESIGNER S. Bugtraq: getting around non-executable stack (and fix)[EB/OL]. <http://seclists.org/bugtraq/1997/Aug/63..>
- [4] Bypassing non-executable-stack during exploitation using return-to-libc.c0ntex[EB/OL]. <http://www.infosecwriters.com/textresources/pdf/return-to-libc.pdf>.

- [5] MCDONALD J. Defeating solaris/sparc non-executable stack protection[EB/OL]. https://www.thc.org/root/docs/exploit_writing/sol-ne-stack.html.
- [6] TIM N. non-exec stack[J]. Bugtraq, 2000.
- [7] KRAHMER S. x86-64 buffer overflow exploits and the borrowed code chunks exploitation techniques[EB/OL]. <http://www.suse.de/krahmer/no-nx.pdf>.
- [8] SHACHAM H. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)[C]//ACM conference on Computer and communications security. 2007: 552-561.
- [9] ROEMER R, BUCHANAN E, SHACHAM H, et al. Return-oriented programming: Systems, languages, and applications[EB/OL]. <http://cseweb.ucsd.edu/liovav/papers/rbss11.html>.
- [10] CHECKOWAY S, DAVI L, DMITRIENKO A. Return-oriented programming without returns[C]//ACM Conference on Computer and Communications Security. 2010:559-572.
- [11] CHECKOWAY S, SHACHAM H. Escape from return-oriented programming: return-oriented programming without returns (on the x86)[R]. CS2010-0954, 2010.
- [12] PaX TEAM. What the future holds for pax[EB/OL]. <http://pax.grsecurity.net/docs/pax-future.txt>.
- [13] BLETSCH T, JIANG X, FREEH V. Jump-oriented programming: a new class of code reuse attack[EB/OL]. <ftp://ftp.ncsu.edu/pub/tech/2010/TR-2010-8.pdf>.
- [14] 陈平. 代码复用攻击与防御技术研究[D]. 南京: 南京大学, 2012.
CHEN P. Research on the attack and defense techniques of code reuse[D]. Nanjing: Nanjing University, 2012.
- [15] 邢晓. 自动化构造 Jump-Oriented Programming[D]. 南京: 南京大学, 2012.
XIAO X. Automatic construction of jump-oriented programming attack[D]. Nanjing: Nanjing University, 2012.
- [16] 邢晓, 陈平, 丁文彪, 茅兵, 谢立. BIOP: 自动构造增强型 ROP 攻击[J]. 计算机学报, 2014, 37(5):1111-1123.
XIAO X, CHEN P, DING W B, et al. BIOP: automatic construction of enhanced ROP attack[J]. Chinese Journal of Computers, 2014, 37(5):1111-1123.
- [17] 袁平海, 曾庆凯. ROP 图灵完备的普遍可实现性. 软件学报, 2017, 28(10): 2583-2598.
YUAN P H, ZENG Q K. Universal availability of ROP-based Turing-complete computation[J]. Chinese Journal of Software, 2017, 28(10): 2583-2598.
- [18] KORMAU T. Return oriented programming for the ARM architecture[D]. Bochum: Ruhr University, 2010.
- [19] DAVI L. Return-oriented programming without returns on ARM[R]. Technical Report HGI-TR-2010-002, 2010.
- [20] 钱逸, 王铁骏, 薛质. 基于 ARM 平台的 ROP 攻击及防御技术. 信息安全与保密, 2012, 10:75-77.
QIAN Y, WANG Y J, XUE Z. ROP attack and defense technology based on ARM[J]. Chinese Information Security and Communication Privacy, 2012, 10:75-77.
- [21] 钱逸. 基于 ARM 架构的 ROP 攻击与防御技术研究[D]. 上海: 上海交通大学, 2012.
QIAN Y. ROP attack and defense technology based on ARM[D]. Shanghai: Shanghai JiaoTong University, 2012.
- [22] FRANCILLON, C. CASTELLUCCIA. Code injection attacks on harvard-architecture devices[C]//ACM Conference on Computer and Communications Security. 2008: 15-26.
- [23] CHECKOWAY S, FELDMAN A J, KANTOR B, et al. Can DREs provide long-lasting security? The case of return-oriented programming and the AVC Advantage[C]//EVT/WOTE 2009. USENIX/ACCURATE/IAVSS. 2009.
- [24] BUCHANAN E, ROEMER R, SHACHAM H, et al. When good instructions go bad: generalizing return-oriented programming to RISC[C]//ACM Conference on Computer and Communications Security. 2008: 27-38.
- [25] LIDNER F. Developments in Cisco IOS forensics Confidence 2.0[EB/OL]. http://www.recurity-labs.com/content/pub/FX_Router_Exploitation.pdf.
- [26] 彭国军, 梁玉, 张焕国, 等. 软件二进制代重用技术综述[J]. 软件学报, 2017, 28(8): 2026-2045.
PENG G J, LIANG Y, ZHANG H G, et al. Survey on software binary code reuse technologies[J]. Chinese Journal of Software, 2017, 28(8): 2026-2045.
- [27] HUND R, HOLZ T, FREILING F C. Return oriented rootkits: bypassing kernel code integrity protection mechanisms[C]//Usenix Security Symposium. 2009: 383-398.
- [28] LEZ R. ARM exploitation ROPMap[C]//BlackHat, 2011.
- [29] 杨超. 基于长指令序列的 ROP 攻击的自动化工具的研究与实现[D]. 南京: 南京大学, 2015.
YANG C. Research and implement on an automatic ROP exploit based on long instruction sequence[D]. Nanjing: Nanjing University, 2015.
- [30] LONG L D, NGUYEN T. Payload already inside: data re-use for rop exploits[C]//Black Hack in the Box. 2010.
- [31] PAYER M. String oriented programming circumventing aslr, dep, and other guard[C]//Chaos Communication Congress. 2011:1-5.
- [32] BITTAU A, BELAY A, MASHTIZADEH and et al. Hacking blind[C]//IEEE Symposium on Security and Privacy. 2014: 227-242.
- [33] HU H, SHINDE S, ADRIAN S, et al. Data-oriented programming: on the expressiveness of non-control data attacks[C]//The 37th IEEE Symposium Security and Privacy. 2016: 969-986.
- [34] LI J K, WANG Z, JIANG X X, et al. Defeating return-oriented rootkits with return-less kernels[C]//European conference on Computer systems. 2010:195-208.
- [35] ONARLIOGLU K, BILGE L, LANZI A et al. G-free: defeating return-oriented programming through gadget-less binaries[C]//

- Computer Security Applications Conference. 2010: 49-58.
- [36] GOKTAS E, ATHANASOPOULOS E, BOS H, et al. Out of control: overcoming control-flow integrity[C]//IEEE Symposium on Security and Privacy. 2014:575-589.
- [37] CARLINI N, WAGNER D. ROP is still dangerous: breaking modern defenses[C]//Usenix Security Symposium, 2014:385-399.
- [38] OTGONBAATAR U. Evaluating modern defenses against control flow hijacking[D]. Massachusetts Institute of Technology, 2015.
- [39] 王明华,尹恒, BHASKAR AV, 等. 代码块:面向二进制程序的细粒度控制流完整性校验方法.信息安全学报, 2016,1(2):61-72.
- WANG M H, YIN H, BHASKAR A V, et al. Binary code continent: finer-grained control flow integrity for stripped Binaries[J]. Chinese Journal of Cyber Security,2016,1(2):61-72.
- [40] 陈志锋, 李清宝, 张平, 等. 面向 Linux 的内核级代码复用攻击检测技术[J]. 软件学报, 2017,28(7):1732-1745.
- CHEN Z F, LI Q B, ZHANG P, et al. Kernel code reuse attack detection technique for Linux[J]. Chinese Journal of Software, 2017, 28(7): 1732-1745.
- [41] CHENG Y Q, ZHOU Z W, YU M, et al. ROPecker: a generic and practical approach for defending against rop attacks[C]//ISOC Network and Distributed System Security Symposium. 2014: 1-14.
- [42] PAPPAS V, POLYCHRONAKIS M, KEROMUTIS A D. Transparent ROP exploit mitigation using indirect branch tracing[C]//USENIX Conference on Security. 2013: 447-462.
- [43] DAVI L, SADEGHI A, WINANDY M. ROPdefender: a detection tool to defend against return-oriented programming attacks[C]//ACM Symposium on Information, Computer and Communications Security. 2011: 40-51.
- [44] HISER J, NGUYEN-TUNONG A, CO M, et al. ILR: where'd my gadgets go?[C]//The 2012 IEEE Symp. On Security and Privacy (SP). 2012:571-85.
- [45] WARTELL R, MOHAN V, HAMLEN K W, et al. Binary stirring:

self-randomizing instruction addresses of legacy x86 binary code[C]//The 2012 ACM Conf. on Computer and Communications Security (CCS 2012). 2012. 157-168.

- [46] PAPPAS V, POLYCHRONAKIS M, KEROMYTIS A D. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization[C]//The 2012 IEEE Symp. on Security and Privacy (SP). 2012. 601-615.

[作者简介]



乔向东 (1970-), 男, 陕西佳县人, 博士, 空军工程大学副教授, 主要研究方向为网络空间安全、信息对抗及多传感器信息融合。



郭戎潇 (1982-), 女, 甘肃兰州人, 博士, 空军工程大学讲师, 主要研究方向为网络空间安全。



赵勇 (1978-), 男, 黑龙江方正人, 清华大学博士生, 主要研究方向为网络信息安全。