

进程控制流劫持攻击与防御技术综述

王丰峰, 张涛, 徐伟光, 孙蒙

(陆军工程大学, 江苏 南京 210001)

摘要: 控制流劫持攻击是一种常见的针对计算机软件的攻击, 给计算机软件安全带来了巨大的危害, 是信息安全领域的研究热点。首先, 从攻击代码的来源角度出发, 阐述了进程控制流劫持攻击的相关研究; 其次, 根据控制流劫持攻击技术的发展现状, 基于不同防御思想介绍了近年来国内外的相关防御技术; 最后对控制流劫持攻防技术发展趋势进行总结和展望。

关键词: 软件安全; 控制流劫持攻击; 控制流完整性; 随机化

中图分类号: TP309.2

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2019058

Overview of control-flow hijacking attack and defense techniques for process

WANG Fengfeng, ZHANG Tao, XU Weiguang, SUN Meng

Army Engineering University, Nanjing 210001, China

Abstract: Control flow hijacking attack is a common attack against computer software, which brings great harm to computer software security and is a research hotspot in the field of information security. Firstly, from the perspective of the source of attack code, the related research was expounded on process's control flow hijacking attack. Secondly, according to the newest development status of control flow hijacking attack technology, the related defense technologies at home and abroad were introduced based on different defense ideas. Finally, the development trend of control flow hijacking offensive and defensive technology were summarized.

Key words: software security, control flow hijacking attacks, control flow integrity, randomization

收稿日期: 2019-01-23; 修回日期: 2019-03-19

通信作者: 王丰峰, princewff@163.com

基金项目: 江苏省优秀青年基金资助项目 (No.BK20180080)

Foundation Items: The Natural Science Foundation of Jiangsu Province for Excellent Yong Scholars (No.BK20180080)

论文引用格式: 王丰峰, 张涛, 徐伟光, 等. 进程控制流劫持攻击与防御技术综述[J]. 网络与信息安全学报, 2019, 5(6): 10-20.

WANG F F, ZHANG T, XU W G, et al. Overview of control-flow hijacking attack and defense techniques for process[J]. Chinese Journal of Network and Information Security, 2019, 5(6): 10-20.

1 引言

随着信息技术的不断发展, 计算机软件在人们的日常生活中扮演着越来越重要的角色。然而, 由于软件开发者的疏忽以及编程语言固有的局限性, 安全漏洞在计算机软件中普遍存在。自1988年Morries利用缓冲区溢出漏洞编写蠕虫病毒瘫痪了大量网络计算机以来^[1], 网络攻击, 尤其是控制流劫持攻击给国家和社会带来了巨大的经济损失。控制流劫持攻击是一种常见的针对计算机软件的攻击, 通常可以通过缓冲区溢出等漏洞改变进程的控制流从而进行恶意操作。针对控制流劫持攻击, 研究人员提出并实现了许多防护技术, 攻防双方相互依赖, 不断发展演进, 形成了如图1所示的攻防对抗演进过程。本文总结了控制流劫持攻击的基本原理和最新进展, 同时依据防御思想重点介绍了相关防御技术, 并进行了分析比较。因为现阶段的防御技术还存在性能开销大、兼容性差等方面的问题, 不少技术尚未进行广泛部署。因此, 本文总结分析控制流劫持攻击与防御的相关技术, 这对研究人员设计更加合理实用的防御技术具有一定的理论价值和实际意义。

2 进程控制流劫持攻击技术研究现状

控制流劫持攻击通过构造特定攻击载体, 利用缓冲区溢出等软件漏洞, 非法篡改进程中的控

制数据, 从而改变进程的控制流程并执行特定恶意代码, 达到攻击目的^[2]。根据攻击代码的来源, 可以将控制流劫持攻击划分为代码注入类攻击和代码重用类攻击。

2.1 代码注入类攻击

在代码注入攻击中, 攻击者通常利用进程的输入操作向被攻击进程的地址空间注入恶意代码, 并且通过覆盖函数返回地址等手段, 使进程非法执行注入的恶意代码, 从而劫持进程控制流达到攻击目的。代码注入类攻击成功的根本原因是在冯·诺依曼体系中, 指令和数据不加区别混合存储在同一个存储器中, 导致计算机可能将数据误当作指令执行。为了抵御代码注入攻击, 研究人员提出了数据执行保护(DEP, data execution protection)的防御措施, 通过将进程地址空间中的可写部分设置为不可执行, 在很大程度上抵御了代码注入类攻击。

2.2 代码重用类攻击

与代码注入类攻击不同, 代码重用攻击不需要攻击者向进程地址空间中注入恶意代码, 它仅利用程序或共享库中已有的指令来完成攻击, 因此能够顺利绕开DEP保护机制。这种攻击模式被证实具有图灵完备性, 但危害大、易扩展。其中, 最为常见的是面向返回的编程(ROP)攻击^[3]。ROP攻击可以利用进程已有的代码片段(称为gadget), 通过栈溢出等漏洞将各个gadget的地址

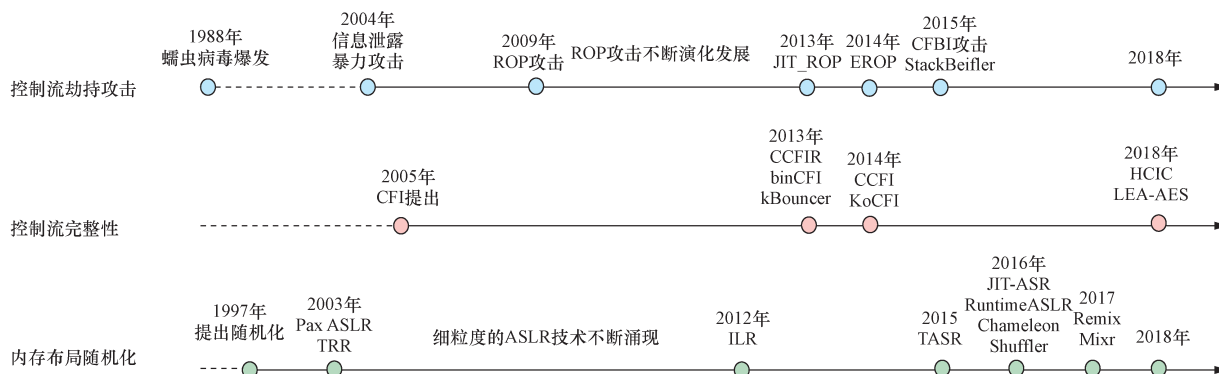


图1 劫持控制流攻防对抗演进过程

和其他相关数据相结合,精心构造 payload,使进程在执行函数返回时,劫持控制流跳转到指定 gadget,从而进行恶意操作^[4]。ROP 攻击的基本过程^[5]如图 2 所示,步骤 2 和步骤 3 就是构造恶意 gadget 的过程,当进程在步骤 6 进行函数返回时,则执行攻击者指定的步骤 7~步骤 9。由于 ROP 攻击使用了以 ret 指令结尾的 gadget 链导致其容易被检测出来。Bletsch 等^[6]提出了 JOP 攻击,以 jmp 或者 call 指令来串联 gadget; Checkoway 等^[7]提出利用 pop-jmp 指令来代替 ret 指令,完成重用攻击。这些 ROP 攻击的变种使 ROP 攻击更加灵活隐蔽,加大了检测难度。

2.3 代码重用类攻击的改进

尽管传统的代码重用攻击主要针对 x86 平台,但是近年来,研究人员在 ARM、SPARC 等平台下也实现了对计算机软件的代码重用攻击,证实了代码重用攻击在多个平台上的广泛存在性^[8-11]。Dullien 等^[12-13]甚至提出了利用中间语义 REIL 实现一个平台无关的 gadget 搜索框架,使跨平台的 ROP 攻击更加容易,进一步增加了计算机应用软件防御

代码重用类控制流劫持攻击的难度。

由于 ROP 攻击需要攻击者知道 gadget 的地址,因此地址空间布局随机化(ASLR, address space layout randomization)^[14]等在进程运行前提供防御措施的静态防御技术,可以有效抵御传统的 ROP 攻击。为了克服传统 ROP 攻击的缺陷, Snow 等^[15]提出了 JIT-ROP(just-in-time ROP)的攻击技术,它通过即时代码重用恢复内存布局,击败了细粒度运行前随机化技术。相比于传统的 ROP 技术, JIT-ROP 攻击可以反复使用内存泄露的方法动态映射应用程序的内存布局,同时发现 gadget 以及 API 函数,并利用这些 gadget 进行攻击。实验证明, JIT-ROP 可以绕开任意粒度的随机化。JIT-ROP 攻击方法的诞生揭示了运行前随机化无法在进程运行时动态引入随机性的致命缺陷。JIT-ROP 技术最大的特点是寻找并利用 gadget 的过程都在程序运行时动态进行的,其攻击形式多样,能够击败 ASLR 技术。

此外, Linux 系统中父进程 fork()中的子进程总是享有和父进程同样的地址空间布局,本文将

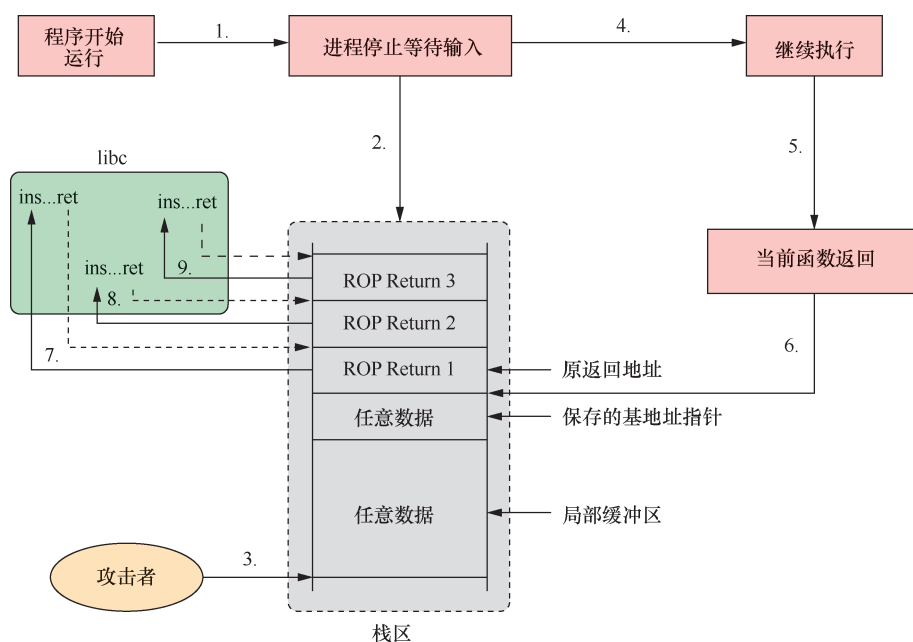


图 2 ROP 攻击流程

这种模式称为 fork-oriented 模式。而许多 C/S 架构的网络模型都在使用这样的运行模式，这会导致 BROP (blind ROP) 攻击。BROP 攻击由 Bittau 等提出，是 JIT-ROP 攻击的扩展模式^[16]。BROP 攻击可以在不获得被攻击进程二进制文件的前提下进行远程攻击，给网络中的计算机系统造成巨大危害。攻击者可以通过不断向子进程进行嗅探和猜测来获取关于地址空间布局的知识并找到足够的 gadget 进行恶意攻击。Bittau 等^[16]设计了一个完全自动化的 BROP 攻击工具，成功绕过了 64 bit 的 ASLR 技术，实现了对 nginx 的 BROP 攻击。

3 进程控制流劫持防御技术研究现状

随着控制流劫持攻击技术的不断发展，其对软件安全造成的威胁也日趋严重。因此研究人员提出许多相应的防御技术。总体上讲，根据防御技术引入的时间将控制流劫持防御技术分为运行前静态防御和运行时动态防御两大类。

3.1 运行前静态防御技术

运行前静态防御技术一般是通过修改编译器或加载器在进程运行前进行缓冲区边界检查、栈保护、替换链接库、随机化等保护措施，来确保运行时控制流不被篡改。数据执行保护通过区分数据和代码的不同权限可以有效阻碍代码注入类攻击。StackGuard^[17]是 GCC 编译器的一个补丁，编译时它通过在缓冲区和返回地址间加入随机 canary 值保护用户栈。Libsafe^[18]是在进程运行前替换易受攻击的库函数，从而保护 strcpy、printf 等存在危险的函数。FormatGuard^[19]通过比较提供给 printf 函数实际参数的数量与格式字符串调用参数的数量来抵御基于格式错误的控制流劫持攻击。此外还有很多类似的技术，在进程运行前加入防御机制以提高软件安全性。然而，这些防御方式留给攻击者充足的时间进行探索和攻击。

JIT-ROP 和 BROP 等攻击通过不断嗅探和猜测，利用进程运行时产生的动态信息，绕开了静态防御的保护，从而篡改进程控制流。这些运行时动态攻击方式甚至可以在没有二进制文件的情况下进行远程攻击，所以运行前防御技术已经无法抵御这种攻击技术。运行时动态攻击的出现，再次打破了攻守双方的平衡，使软件防御技术研究人员将目光转向了运行时动态防御技术上。

3.2 运行时动态防御技术

运行时动态防御技术则在进程运行过程中即时地进行安全保护，防止控制流被非法篡改。根据进程运行时动态防御思想的不同，本文可以将进程运行时动态防御技术分为两大类：基于可信计算的运行时防御技术和基于主动变换防御思想的防御技术。

3.2.1 基于可信计算的运行时防御技术

可信计算通过确保系统始终在可信状态下迁移来确保系统安全，基于可信计算的动态防御技术则通过将进程控制流与预期的正常控制流进行对比，从而保证进程运行过程始终可信。

基于可信计算的运行时防御技术的代表是控制流完整性 (CFI, control flow integrity)。CFI 最早由 Martín 等^[20]提出。CFI 基于动态可信度量，分析程序正常控制流，并依此构造出针对进程的控制流图 (CFG, control flow graph)。CFI 技术的关键是限制进程运行过程中的控制流转移，从而使进程的控制流只在其原有 CFG 所限定的范围内，保证进程运行过程可信^[21]。具体做法是通过分析程序的控制流图，获得间接转移指令目标的地址，形成一个白名单，然后在进程运行过程中，核对转移目标指令是否在正常控制流的白名单中。由于 JIT-ROP 等针对 Linux 进程的控制流劫持攻击会改变程序的正常行为，从而违背原有的控制流图。因此 CFI 能够抵御运行时动态控制

流劫持攻击。

在 CFI 提出后, 该技术得到了研究人员的广泛研究, 各种 CFI 防御技术层出不穷。CFI 可以根据划分目标指令的精细程度分为细粒度 CFI 和粗粒度 CFI。严格意义上讲, 最早 Martín 提出的 CFI 技术是细粒度 CFI。细粒度 CFI 仔细检查每个间接转移指令的目标地址, 并与正常控制流图进行比对, 确保每条转移指令只能转到其限定的范围内, 来保证程序控制流的完整性。Mashtizadeh 等^[22]提出的 CCFI 会识别所有影响程序控制流的对象, 并且计算这些对象的消息认证码, 从而进行指针加密。通过在使用每个控制流元素之前检查消息认证码, 系统可以保证运行时控制流不被非法篡改。Criswell 等^[23]提出了 KCoFI, 这是一种在操作系统内核上的 CFI, KCoFI 在验证进程控制流完整性的基础上, 加入一个对进程进行动态监视的中间软件层, 以进一步保护一些关键数据结构。然而, 对进程指令进行精细的检查会造成巨大的性能开销。为了降低系统性能开销, 粗粒度的 CFI 选择放宽检查条件, 将同一类转移指令整理到一起进行验证。例如, CCFIR^[24]利用一个专用的“Springborad section”来实现间接分支转移, 没有实现对每一条指令进行精细检查; binCFI^[25]将间接转移指令进行分类, 同一类转移指令归纳到同一目标集合中, 粗化检查粒度, 提高进程运行效率。由于没有对每一条转移指令进行检查, 粗粒度 CFI 会导致系统抵御攻击的能力下降, 攻击者依旧可以针对粗粒度 CFI 进行代码重用攻击。Göktas 等^[26]利用 entry point (EP) gadget 和 call site (CS) gadget 成功实现了对粗粒度 CFI 的攻击; Conti 等^[27]提出了名为 StackDefiler 的系列攻击, 分别利用堆上的漏洞、用户空间和内核空间切换时存在的漏洞以及泄露影子堆栈的地址, 完成了在有 CFI 保护的前提下针对堆栈的攻击, 同时也击败了

DEP 和 ASLR 保护技术; Carlini 等^[28]利用对非控制数据的攻击, 提出了名为 CFB 的攻击技术, 并且证明 CFB 能够对精细的 CFI 进行有效攻击。2015 年, Otgonbaatar 指出^[29], 即便获得了细粒度的控制流完整性, 也存在被攻击的风险, 这使 CFI 技术的防御有效性受到了质疑。

为了减少 CFI 的性能开销, 目前有研究人员开始利用硬件来辅助实现 CFI。Pappas 等^[30]设计了名为 kBouncer 的基于硬件的 CFI, 该技术利用 Last Branch Register 监视进程中的跳转指令, 以防止进程的异常控制流。Qiu 等^[31]提出了利用一种基于轻量级加密的体系结构 (LEA-AES) 对间接跳转指令和返回地址进行加解密, 从而保证了进程控制流的完整可信。Zhang 等^[32]设计了 HCIC 技术, 该技术利用海明距离匹配和线性加解密机制, 通过特定硬件的辅助, 在不对编译器进行修改、不增加新的指令集的条件下, 实现了低开销的 CFI。

CFI 是建立在对被保护程序深入理解基础上的防御技术, 因此往往需要源代码或调试信息。在运行动态监测时, CFI 无法完美刻画进程的 CFG, 因此可能导致限制程序正常运行或允许程序执行非法操作。在运行时 CFI 需要比对跳转指令的目标地址, 导致巨大性能开销。由于存在以上种种问题, 虽然 CFI 已提出十多年时间, 仍然没有在实际系统中得到广泛部署应用。

表 1 给出上述控制流完整性防御技术的对比。

3.2.2 基于主动变换防御思想的运行时防御技术

主动变换防御通过持续地、动态地变化来迷惑攻击者, 从而增加其攻击成本, 降低其攻击成功概率^[33]。基于主动变换防御的运行时防御技术则是在进程运行过程中不断引入随机性和多样性, 打破攻击者构造传统攻击链所依赖的各种有利条件, 加大攻击者攻击成功的概率。

基于主动变换防御思想的运行时防御技术的代表是内存布局随机化。随机化思想最早是在

表 1 控制流完整性防御技术对比

分类	名称	技术特点	安全性评价	性能损耗	存在问题
细粒度 CFI	CFI	比对间接转移指令，确保控制流在合理 CFG 中	能有效抵御控制流劫持攻击，但存在因 CFG 构造不完善而被攻击者攻击的风险	SPEC CPU 2000 基准平均开销约 16%	对转移指令进行精细检查带来巨大性能开销
	CCFI	计算影响控制流对象的消息认证码，并加密		SPEC CPU 2006 基准平均开销约 52%	
	KCoFI	验证进程控制流并加强对关键数据结构的保护		LMbench 基准 130%~150%开销	
粗粒度 CFI	CCFIR	利用具体方案，将同类转移指令归纳到一起，减少对转移指令的精细检查	相对于细粒度 CFI，安全性有所下降	SPEC CPU 2006 基准平均开销约 3.68%	粗粒度 CFI 会导致系统抵御攻击的能力下降
	binCFI			SPECCPU2006 基准平均开销约 4.29%	
基于硬件的 CFI	kBouncer	利用寄存器来监视跳转指令，以实现 CFI	能抵御多种溢出攻击，但 LBR 只有 16 条记录，因此攻击者可能寻找更长路径进行攻击	Wine 测试基准，平均开销约为 1%	需要特定硬件支持，同时需要减少漏报和误报。针对 kBouncer，攻击者可能寻找更长路径进行攻击
	LEA-AES	基于 LEA-AES 进行完整性验证	减少攻击者用来组织 ROP 攻击的 gadget 平均约 88.93%	以 12 种开源程序为基准，平均性能开销约 3.19%	
	HCIC	利用 HCIC 技术实现低开销的 CFI	在不泄露密钥的情况下，能抵御 ROP 攻击，漏报率低，无误报	多个测试基准，平均性能开销约 0.95%	

1997 年由 Forrest 等^[34]提出的，通过引入多样性和随机性来提高计算机系统的稳健性。内存布局随机化通过不断改变进程内存布局，实现进程攻击面的随机性变换，从而消除了攻击者的不对称优势，增加了攻击者的攻击难度。此外，由于每一计算机进程在系统中的内存布局不同，运行时内存布局随机化还能阻碍攻击在网络中的扩散。

Pax ASLR^[14]是早期随机化技术的代表，其基本原理如图 3 所示。它会对 ELF 可执行文件地址空间中运行时才映射的动态共享库位置等映射区域进行随机化操作。当攻击者需要知道进程中的 gadget 地址时，由于内存布局被随机打乱，因此加

大了攻击者的攻击难度。同时当进程被编译为位置无关代码时，Pax ASLR 还能随机化代码段的基地址，进一步增加了攻击者攻击成功的难度。

目前，在许多流行的操作系统中部署了 Pax ASLR 技术，以提高操作系统的安全性。然而，Pax ASLR 是一种粗粒度的随机化技术，随机化熵值低并且只随机化段基地址，因此容易遭受暴力攻击和信息泄露攻击^[35]。细粒度随机化技术^[36-42]不仅随机化了段基地址，而且打乱进程段内对象的相对偏移，增大随机化引入的熵值，从而有效抵御暴力攻击和信息泄露攻击。作为一种解决传统粗粒度随机化缺陷的技术，细粒度随机化技术被研究者广泛

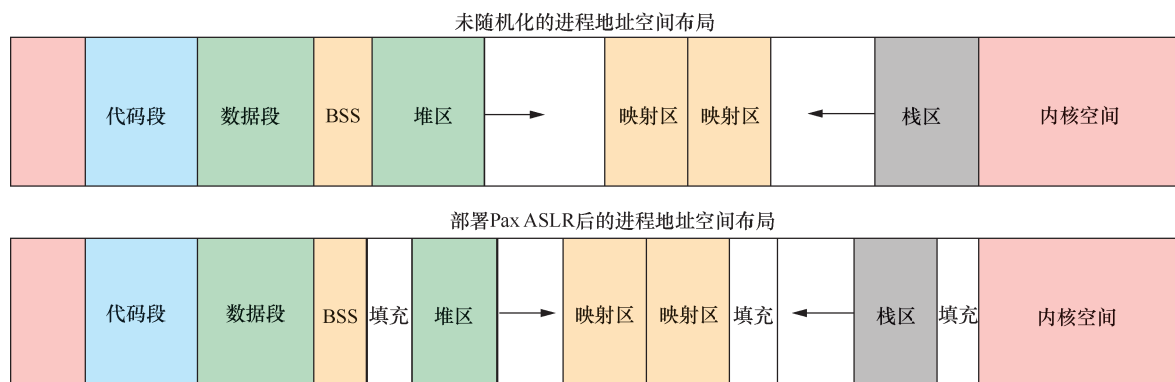


图 3 Pax ASLR 基本原理

采用, 并被认为可以进一步提高进程的安全性。

然而, JIT-ROP 和 BROP 攻击的提出, 使运行前的细粒度随机化技术在面对运行动态攻击时没有了用武之地。针对 JIT-ROP 和 BROP 攻击的一种有效保护方法是在运行时对地址空间进行再随机化操作。运行时再随机化技术通过在进程运行过程中不断对进程地址空间进行再随机化布局, 减小了单一地址空间布局的存在时间, 缩短攻击者的攻击周期, 使攻击者无法掌握足够的知识来完成一次完整的攻击, 运行时随机化示意如图 4 所示。

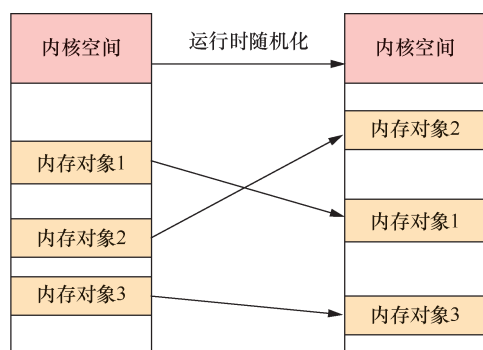


图 4 运行时随机化示意

Giuffrida 等^[42]提出了操作系统中细粒度随机化技术的设计和运行时再随机化的思想, 并在 Minix3 微内核中成功部署了运行时内核级地址空间布局随机化技术。Chen 等^[43]提出了 JIT-ASR 技术, 该技术使用虚拟内存管理, 可以在运行时通过修改代码地址的虚拟页码和页表来不断改变程序的地址空间。侯宇^[44]分析了 ROP 攻击的原理, 并在此基础上设计了一种称为 Chameleon 的防御机制, Chameleon 在虚拟地址转换为物理地址中间加了一个中间层, 从而将取指或访存和进程地址空间随机化有机结合起来。Chen 等^[45]提出了 Remix 技术, 通过 LLVM 编译器框架将代码函数分割成许多基本块, 并对进程地址空间中的函数基本块进行基于时间模型的运行时再随机化。这些运行时随机化技术大都需要修改内核模块来支持进程再随机化操作, 实现难度大, 兼容性有限, 并且都选择基

于时间的再随机化, 会造成一些不必要的性能开销。Hawkins 等^[46]提出的 Mixr 运行时再随机化系统可以在不依赖源代码以及不修改系统内核和加载器等组件的基础上进行, 然而该技术由于在运行时需要反复进行动态链接导致性能开销过大。

上述运行时随机化技术都是基于时间的再随机化模型, 即每隔一段时间进行一次随机化操作^[46]。由于运行时地址空间再随机化操作会给程序性能带来影响。在没有受到危险操作时, 过于频繁的随机化会带来巨大的性能开销, 而随机化频率过低则给攻击者留下攻击的机会。基于时间的再随机化模型无法消除这一矛盾。Bigelow 等^[47]提出了实时再随机化 (TASR) 技术, 在输入/输出系统调用之间引入随机性, 从而使攻击者获得的内存信息无效。TASR 提出了一种将随机化与潜在的运行时信息泄露相结合的方法, 视成对的输入/输出操作为危险操作, 当有成对输入/输出操作时, 则触发随机化。由于只在面临危险操作时进行随机化, TASR 减少了不必要的随机化开销。雷啸^[48]改进了 TASR 技术过于宽泛的危险操作的判断, 提出了安全敏感区的概念, 只有当进程 write 操作访问安全敏感区时才在后续 read 操作前触发随机化, 进一步减少了性能开销。Morton 等^[49]提出了当攻击者在内存中公开了可能有用的可执行 gadget 时, 才触发防御机制, 以确保攻击者不能利用这些字节来劫持运行时进程的控制流。这些基于潜在风险的随机化方案, 减少了不必要的再随机化操作, 对于大部分进程而言, 该方案要优于基于时间的再随机化方案。

针对 BROP 攻击, Lu 等^[50]提出的 RuntimeASLR 技术对父进程 fork 出的子进程进行地址空间布局进行再随机化, 从而抵御复制探测攻击。RuntimeASLR 的技术核心是一个自动的、整体的指针跟踪机制, Lu 认为这是未来进行指针识别和运行时动态保护的有用工具。然而, RuntimeASLR

技术只适合 nginx 这类 fork-oriented 程序,并且只支持 x86 体系架构。

除了在运行时对内存布局进行随机置换外,还有研究人员提出加载多个进程映像并在运行时进行随机切换,从而使攻击者获取的运行时内存信息无效。Davi 等^[51]提出了 Isomeron 技术,在一个虚拟地址空间中,Isomeron 同时加载了两份不同的进程映像。在程序运行时,一旦发生了函数调用,就随机决定接下来执行哪个程序副本。Williams-King 等^[52]提出了称为 Shuffler 的防御机制。Shuffler 在几毫秒内对进

程地址空间进行运行时再随机化,并“原子地”从一个随机化的代码副本切换到另一个随机化的代码副本。Williams-King 等还考虑了 Shuffler 本身可能存在的攻击面,通过对自身的地址空间进行再随机化,可以减小防御工具本身攻击面对安全性带来的影响。由于 gadget 的地址在每个副本中不一样,且攻击者也无法预测接下来运行哪个副本。因此这样的方法能有效抵御 JIT-ROP 和 BROP 攻击。

表 2 给出进程运行时再随机化技术的对比。

表 3 给出进程控制流劫持运行时动态防御技术的

表 2 进程运行时再随机化技术对比

分类	名称	技术特点	安全性评价	性能损耗	存在问题
基于时间	JIT-ASR	利用虚拟内存管理实现内存布局随机化	随机化间隔足够小时,能有效抵御控制流劫持攻击	SPEC CPU 2006 基准平均开销约 1.2%	无法消除安全性与性能开销之间的矛盾,Chameleon、Remix、Mixr 开销过高,JIT-ASR 需要额外寄存器辅助实现
	Chame-leon	结合取指异或进行内存布局随机化		SPEC CPU 2006 基准,不同随机化频率,开销不同,范围大致 10%~20%	
	Remix	随机化函数基本块		SPEC CPU 2006 基准,不同随机化频率,开销不同,大部分频率开销小于 5%	
	Mixr	修改二进制文件实现内存随机化		SPEC CPU 2006 基准,平均开销约 60%	
基于风险	TASR	当进程存在风险操作时触发内存布局随机化,雷啸对 TASR 做出了改进	能抵御通过 I/O 进行的攻击,但不能保护动态生成的代码	SPEC CPU 2006 基准平均开销约 2.1%,进程 I/O 密集时超过 50%,雷啸改进后 TASR 部署在 I/O 密集进程中时,开销基本小于 10%	依赖源代码,同时无法有效保护动态生成的代码
	Runtime-ASLR	利用动态信息流追踪,对子进程进行内存随机化操作	能有效抵御 BROP 攻击	SPEC CPU 2006 基准,nginx 启动延迟约 35 s,子进程无性能开销	父进程开销过大,存在一定指针误报
运行路径随机化	Ismeron	加载多个进程映像并在运行时进行随机切换	能有效抵御控制流劫持攻击,尤其是依赖长 ROP 链的攻击	SPEC CPU 2006 基准平均开销约 19%	空间开销大,攻击者可能会攻击执行分配器
	Shuffler			SPEC CPU 2006 基准,每 50 ms 随机化一次,平均开销约 15%	

表 3 进程运行时动态防御技术对比

防御技术名称	思想基础	基本原理	有效性分析	性能损耗	典型技术代表	主要问题
控制流完整性	可信计算、正确性	限制程序中的控制流转移,从而使进程运行时控制流只在其原有控制流图所限定的范围内,保证进程运行过程可信	粗粒度 CFI 抵御攻击的安全性有限,细粒度 CFI 也存在被攻克的可能性,无法阻碍攻击的扩散	性能开销普遍较大	CFI、CCFI、binCFI...	无法完全正确地刻画程序控制流图,性能开销普遍较大,存在被攻击者攻击的风险
运行时再随机化	主动变换防御、概率论	通过不断改变进程内存布局,实现攻击面的动态变换,从而增加攻击者攻击成功的代价和复杂性,降低其攻击成功的概率,提高系统的稳健性	消除攻击者积累的关于软件内存布局的知识,大幅降低攻击者攻击成功的概率,同时能阻碍攻击的扩散	性能开销普遍较大	TASR、Chameleon、Remix、RuntimeASLR...	存在误报可能性,性能开销普遍较大

总体对比。

4 结束语

在信息安全领域中,如何抵御控制流劫持攻击一直是一个热点问题,尤其 Snow 提出 JIT-ROP 攻击之后,以这种新型控制流劫持攻击为基础的攻击模式不断被提出,对计算机进程构成巨大威胁。BROP 攻击的提出更是突破了传统控制流劫持攻击对目标二进制的依赖,实现了“盲的”控制流劫持攻击,使传统的软件静态被动防御机制变得形同虚设。

控制流劫持攻击是计算机软件面临的巨大威胁,随着新型代码重用类攻击的诞生和演化,控制流劫持攻防对抗不断升级。控制流劫持攻击技术的发展也面临着许多挑战和新的发展方向。首先, JIT-ROP 和 BROP 攻击所采用的方法并不高效,需要多次尝试和足够时间来“积累知识”。然而,攻击时间越长,防御者越可能对攻击做出反应。因此快速获取成功完成攻击所需要的信息是未来控制流劫持攻击的研究重点。其次,在二进制文件不可得的情况下进行控制流劫持攻击是一个难点。而 BROP 攻击则揭示了 fork-oriented 应用进程所存在的安全漏洞,为今后攻击二进制信息不可得的应用进程拓宽了思路。最后,随着 ARM 等其他平台的广泛应用,针对这些平台的防御技术不断升级。借鉴现有控制流劫持攻击的研究成果,实现对其他平台的控制流劫持攻击也值得研究人员关注。目前,运行时动态防御技术是抵御 JIT-ROP 和 BROP 攻击的有效手段,笔者依据其背后防御思想的不同分成了基于可信计算和基于主动变换防御思想的运行时防御技术。

基于可信计算的 CFI 技术通过限制程序的控制流来保证程序被正确地执行。然而,细粒度的 CFI 也被研究人员证实存在防御缺陷;进程动态控制流图难以精确刻画;同时 CFI 还存在性能开

销大、依赖调试信息等问题。如何在低性能开销的基础上,构造正确合理的 CFG 是制约 CFI 技术进一步发展的瓶颈,也是 CFI 未来主要的一个研究方向。与此同时,结合硬件来降低 CFI 开销,并提高 CFG 覆盖率和准确率也是一个值得探索的方向。

基于主动变换防御的运行时动态随机化技术,通过在进程内部引入随机性来抵御运行时动态代码重用攻击。运行时再随机化通过不断对进程地址空间进行再随机化布局,改变程序自身的属性,打断了攻击者对进程地址空间的原始认知,使攻击者先前积累的有关进程地址空间的知识无效化,从而抵御代码重用攻击。相比 CFI,运行时随机化能够有效阻碍攻击的扩散,同样也值得深入研究。内存布局随机化技术会使进程出现“带伤运行”的情况,存在造成进程非正常运行的可能性,如何降低随机化操作对进程带来的“伤害”,具有重要的现实意义。此外,运行时内存随机化对进程的保护能力与其随机化粒度之间的关系,随机化最合理的触发时间等问题仍然缺乏相应的研究,这使运行时随机化的发展陷入无法在保护力度和性能损耗之间取得一个最为合理平衡的困境,因此对运行时内存随机化进行费效评估也是一个值得研究的方向。

最后,现阶段的控制流完整性和地址空间随机化技术大多是针对 x86 平台,在 ARM 等其他平台上鲜有类似研究。然而,如第 2 节所述,控制流劫持攻击在许多平台中都可以实现,并且同样可以达到执行任意恶意代码的目的。因此在 ARM 等其他平台上的或者跨平台的运行时动态保护技术同样亟待研究和解决。

参考文献:

- [1] 邵思豪,高庆,马森,等.缓冲区溢出漏洞分析技术研究进展[J].软件学报,2018,29(5):1179-1198.
SHAO S H, GAO Q, MA S, et al. Progress in research on buffer

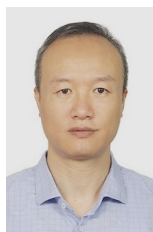
- overflow vulnerability analysis technologies[J]. *Journal of Software*, 2018, 29(5): 1179-1198
- [2] 张超. 针对控制流劫持攻击的软件安全防护技术研究[D]. 北京: 北京大学, 2013.
ZHANG C. Research on software security defense against control-flow hijacking attacks[D]. Beijing: Beijing University, 2013.
- [3] ROEMER R, BUCHANAN E, SHACHAM H, et al. Return-oriented programming: systems, languages, and applications[J]. *ACM Transactions on Information & System Security*, 2012, 15(1): 1-34.
- [4] 乔向东, 郭戎满, 赵勇. 代码复用对抗技术研究进展[J]. *网络与信息安全学报*, 2018, 4(3): 1-12.
QIAO X D, GUO R X, ZHAO Y. Research progress in code reuse attacking and defending[J]. *Chinese Journal of Network and Information Security*, 2018, 4 (3): 1-12.
- [5] DAVI L, SADEGHI A R, WINANDY M. ROPdefender: a detection tool to defend against return-oriented programming attacks[C]//The 6th ACM Symposium on Information, Computer and Communications Security. 2011: 40-51.
- [6] BLETSCH T, JIANG X, FREEH V W, et al. Jump-oriented programming: a new class of code-reuse attack[C]//The 6th ACM Symposium on Information, Computer and Communications Security. 2011: 30-40.
- [7] CHECKOWAY S, DAVI L, DMITRIENKO A, et al. Return-oriented programming without returns[C]//ACM Conference on Computer & Communications Security. 2010: 559-572.
- [8] 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.
- [9] BUCHANAN E, ROEMER R, SHACHAM H, et al. When good instructions go bad: Generalizing return-oriented programming to RISC[C]//The 15th ACM Conference on Computer and Communications Security. 2008: 27-38.
- [10] DAVI L, DMITRIENKO A, SADEGHI A, et al. Return-oriented programming without returns on ARM[R]. Ruhr-University Bochum, 2010.
- [11] KORNAU T. Return oriented programming for the ARM architecture[D]. Ruhr: Ruhr-Universität Bochum. 2010.
- [12] DULLIEN T, PORST S. REIL: a platform-independent intermediate representation of disassembled code for static code analysis[J]. *Cansecwest*, 2009.
- [13] DULLIEN T, KORNAU T, WEINMANN R. A framework for automated architecture independent gadget search[C]//Usenix Conference on Offensive Technologies. 2010: 1.
- [14] PaX team: PaX address space layout randomization (ASLR)[EB].
- [15] SNOW K Z, MONROSE F, DAVI L, et al. Just-in-time code reuse: on the effectiveness of fine-grained address space layout randomization[C]//2013 IEEE Symposium on Security and Privacy. 2013: 574-588.
- [16] BITTAU A, BELAY A, MASHTIZADEH A, et al. Hacking blind[C]//2014 IEEE Symposium on Security and Privacy. 2014: 227-242.
- [17] COWAN C, PU C, MAIER D, et al. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks[C]//USENIX Security Symposium. 1998, 98: 63-78.
- [18] TSAI T, SINGH N. Libsafe: transparent system-wide protection against buffer overflow attacks[C]//International Conference on Dependable Systems and Networks. IEEE, 2002: 541.
- [19] COWAN C, BARRINGER M, BEATTIE S, et al. FormatGuard: automatic protection from printf format string vulnerabilities[C]//USENIX Security Symposium. 2001, 91.
- [20] MARTÍN A, BUDIÚ M, ERLINGSSON, ÚLFAR, et al. Control-flow integrity[C]//ACM Conference on Computer & Communications Security. 2005: 340.
- [21] 武成岗, 李建军. 控制流完整性的发展历程[J]. *中国教育网络*, 2016 (4): 52-55.
WU C G, LI J J. The evolution of control flow integrity[J]. *China Education Network*. 2016(4): 52-55.
- [22] MASHTIZADEH A J, BITTAU A, MAZIERES D, et al. Cryptographically enforced control flow integrity[J]. *arXiv preprint arXiv:1408.1451*, 2014.
- [23] CRISWELL J, DAUTENHAHN N, ADVE V. KCoFI: complete control-flow integrity for commodity operating system kernels[C]//2014 IEEE Symposium on Security and Privacy. 2014: 292-307.
- [24] ZHANG C, WEI T, CHEN Z, et al. Practical control flow integrity and randomization for binary executables[J]. *IEEE Symposium on Security & Privacy*, 2013: 559-573.
- [25] ZHANG M, SEKAR R. Control Flow Integrity for {COTS} Binaries[C]//Presented as part of the 22nd Security Symposium. 2013: 337-352.
- [26] GÖKTAS E, ATHANASOPOULOS E, BOS H, et al. Out of control: overcoming control-flow integrity[J]. *Security & Privacy*, 2014: 575-589.
- [27] CONTI M, CRANE S, DAVI L, et al. Losing control: on the effectiveness of control-flow integrity under stack attacks[C]//The 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015: 952-963.
- [28] CARLINI N, BARRESI N, MATHIAS P, et al. Control-flow bending: on the effectiveness of control-flow integrity[C]//Usenix Conference on Security Symposium. 2015: 161-176.
- [29] OTGONBAATAR U. Evaluating modern defenses against control flow hijacking[D]. Massachusetts: Massachusetts Institute of Technology. 2015.
- [30] PAPPAS V, POLYCHRONAKIS M, KEROMYTIS A D. Transparent {ROP} exploit mitigation using indirect branch tracing[C]//Presented as Part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13). 2013: 447-462.
- [31] QIU P, LYU Y, ZHANG J, et al. Control flow integrity based on lightweight encryption architecture[J]. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018, 37(7): 1358-1369.
- [32] ZHANG J, QI B, QIN Z, et al. HCIC: Hardware-assisted control-flow integrity checking[J]. *IEEE Internet of Things Journal*, 2018, 6(1): 458-471.
- [33] 蔡桂林, 王宝生, 王天佐, 等. 移动目标防御技术研究进展[J]. *计算机研究与发展*, 2016, 53 (5): 968-987.

- CAI G L, WANG B S, WANG T Z, et al. Research and development of moving target defense technology[J]. Journal of Computer Research and Development, 2016, 53 (5): 968-987.
- [34] FORREST S, SOMAYAJI A, ACKLEY D H. Building diverse computer systems[C]//The Sixth Workshop on Hot Topics in Operating Systems (Cat. No. 97TB100133). 1997: 67-72.
- [35] SHACHAM H, PAGE M, PFAFF B, et al. On the effectiveness of address-space randomization[C]//The 11th ACM Conference on Computer and Communications Security. 2004: 298-307.
- [36] KIL C, JUN J, BOOKHOLT C, et al. Address space layout permutation (ASLP): towards fine-grained randomization of commodity software[C]//2006 22nd Annual Computer Security Applications Conference (ACSAC'06). 2006: 339-348.
- [37] HISER J, NGUYEN-TUONG A, Co M, et al. ILR: Where'd my gadgets go[C]//2012 IEEE Symposium on Security and Privacy. 2012: 571-585.
- [38] IYER V, KANITKAR A, DASGUPTA P, et al. Preventing overflow attacks by memory randomization[C]//2010 IEEE 21st International Symposium on Software Reliability Engineering. 2010: 339-347.
- [39] BHATKAR S, DUVARNEY D C, SEKAR R. Address obfuscation: an efficient approach to combat a broad range of memory error exploits[C]//USENIX Security Symposium. 2003, 12(2): 291-301.
- [40] BHATKAR S, DUVARNEY D C, SEKAR R. efficient techniques for comprehensive protection from memory error exploits[C]//USENIX Security Symposium. 2005: 17.
- [41] BACKES M AND NURNBERGER S. Oxymoron: making fine-grained memory randomization practical by allowing code sharing[C]//Usenix Conference on Security Symposium. 2014: 433-447.
- [42] GIUFFRIDA C, KUIJSTEN A, TANENBAUM A S. Enhanced operating system security through efficient and fine-grained address space randomization[C]//Presented as Part of the 21st {USENIX} Security Symposium ({USENIX} Security 12). 2012: 475-490.
- [43] CHEN X, XUE R, Wu C. Timely address space rerandomization for resisting code reuse attacks[J]. Concurrency and Computation: Practice and Experience, 2017, 29(16): e3965.
- [44] 侯宇. 基于动态随机化和只可执行内存的 JIT-ROP 防御研究[D]. 南京: 南京大学, 2016.
HOU Y. Defence against JIT-ROP based on dynamic randomization and executable only memory[D]. Nanjing: Nanjing University, 2016
- [45] CHEN Y, WANG Z, WHALLEY D, et al. Remix: on-demand live randomization[C]//The sixth ACM Conference on Data and Application Security and Privacy. 2016: 50-61.
- [46] HAWKINS W, NGUYEN-TUONG A, HISER J D, et al. Mixr: flexible runtime rerandomization for binaries[C]//The 2017 Workshop on Moving Target Defense. 2017: 27-37.
- [47] BIGELOW D, HOBSON T, RUDD R, et al. Timely rerandomization for mitigating memory disclosures[C]//ACM Sigsac Conference on Computer & Communications Security, 2015: 268-279.
- [48] 雷啸. 内存信息泄露的运行中随机化防御方法的研究与改进[D]. 南京: 南京大学, 2017.
LEI X. Research and improvement of runtime randomization defense method against memory information leakage[D]. Nanjing: Nanjing University. 2017.
- [49] MORTON M, KOO H, LI F, et al. Defeating zombie gadgets by re-randomizing code upon disclosure[C]//International Symposium on Engineering Secure Software and Systems. 2017: 143-160.
- [50] LU K, NURNBERGER S, BACKES M, et al. How to make ASLR win the clone wars: runtime re-randomization[C]//Network & Distributed System Security Symposium. 2016.
- [51] DAVI L, LIEBCHEN C, SADEGHI A R, et al. Isomeron: code randomization resilient to (just-in-time) return-oriented programming[C]//Network & Distributed System Security Symposium. 2015.
- [52] WILLIAMS-KING D, GOBIESKI G, WILLIAMS-KING K, et al. Shuffler: fast and deployable continuous code re-randomization[C]//12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016: 367-382.

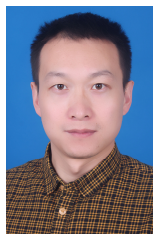
[作者简介]



王丰峰 (1994-), 男, 江苏昆山人, 陆军工程大学硕士生, 主要研究方向为系统安全。



张涛 (1973-), 男, 甘肃泾川人, 博士, 陆军工程大学教授, 主要研究方向为操作系统安全、数据库安全、智能终端软件安全等。



徐伟光 (1984-), 男, 安徽宿州人, 博士, 陆军工程大学讲师, 主要研究方向为移动目标防御、密码学、区块链等。



孙蒙 (1984-), 男, 山东齐河人, 博士, 陆军工程大学副教授, 主要研究方向为网络安全与机器学习、智能语音处理等。