引用格式:詹静, 陈鹏, 张茜, 等. 代码复用攻防技术演化综述[J]. 北京工业大学学报, 2024, 50(5): 632-650.
ZHAN J, CHEN P, ZHANG Q, et al. Survey on code reuse attack and defense technology evolution[J]. Journal of Beijing University of Technology, 2024, 50(5): 632-650. (in Chinese)

代码复用攻防技术演化综述

詹 静[1,2], 陈 鹏[1], 张 茜[1], 李永震[1], 赵 勇[1]
(1. 北京工业大学信息学部, 北京 100124; 2. 可信计算北京市重点实验室, 北京 100124)

摘 要: 当前代码复用攻击研究多从一种或多种代码复用攻击或防御技术角度进行现状和趋势分析,对影响攻防的关键特征及技术覆盖不全面,对攻防技术对抗演化发展规律分析较少。为解决上述问题,从经典代码复用攻击——返回导向编程(return-oriented programming,ROP)攻击的生命周期入手,归纳影响此类攻击成功与否的关键特征,基于时间线和这些特征,综合衡量安全和性能因素,给出了代码复用攻防技术的发展规律。

关键词: 程序安全; 代码复用攻击; 攻防技术演化; 指令结构特征; 地址随机化; 运行代码特征
中图分类号: TP 311.5 **文献标志码**: A **文章编号**: 0254-0037(2024)05-0632-19
doi: 10.11936/bjutxb2022070008

Survey on Code Reuse Attack and Defense Technology Evolution

ZHAN Jing[1,2], CHEN Peng[1], ZHANG Qian[1], LI Yongzhen[1], ZHAO Yong[1]
(1. Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China;
2. Beijing Key Laboratory of Trusted Computing, Beijing 100124, China)

Abstract: Most current surveys on code reuse attack conclude the status quo and trends from the perspective of one or several attack or defense technologies, lacking analysis of key features related to the attack or defense affects. To solve the above problems, the key characteristics that affect the results of classic code reuse attack were summarized, starting from the life cycle of the classic code reuse attack, i.e., return-oriented programing (ROP) attack. Based on the technology developing timeline and these characteristics, security and performance factors were comprehensively measured, and development rules and trends of code reuse attack and defense confrontation technologies were analyzed and summarized.

Key words: program security; code reuse attack; attack and defense technology evolution; instruction structure characteristics; address randomization; runtime code characteristics

近20年,利用内存中的良性指令片段组合进行攻击的代码复用攻击及防御技术成为研究热点。当前研究通常从单一攻击或防御技术角度阐述代码复用攻击的发展现状,对影响攻防的关键特征及技术覆盖不够全面,对攻防技术对抗演化发展趋势及对实际应用场景的影响分析也较少。

本文针对实现返回导向编程(return-oriented programing,ROP)攻击的4个关键步骤涉及的核心技术,即构建攻击指令片段(gadget)、构建完成恶意功能的gadget链、设计调度攻击所需的内存(主要是栈)布局及利用内存漏洞加载攻击载荷、劫持程序控制流,归纳总结了代码复用攻防的3条攻防演

收稿日期: 2022-07-14; 修回日期: 2022-09-27
基金项目: 国家重点研发计划资助项目(2016YFB0800204)
作者简介: 詹 静(1982—), 女, 讲师, 主要从事系统安全、可信计算方面的研究, E-mail: zhanjing@bjut.edu.cn

化技术路线，并基于时间线分析与这些特征相关的攻防技术演化规律，综合考虑安全和性能因素，分析这些核心技术对现实应用场景的影响及攻防对抗技术的演化发展趋势。

1　代码复用攻击原理及关键特征

早期的内存注入攻击通过常见漏洞（如缓冲区溢出漏洞）使控制流转向攻击者注入进程地址空间动态数据区（如栈、堆）中的恶意代码位置。攻击成功的主要原因在于代码、数据区未严格分离。随后出现的 Linux 不可执行（no-execute，NX）机制和 Windows 数据执行保护（data execution prevention，DEP）机制将进程地址空间中的堆栈数据等标记为不可执行，使注入代码无法得到可执行权限，从而缓解注入攻击，如图 1 中虚线框所示。目前，Linux 通过 - z execstack 与 - z noexecstack 参数禁用/开启 NX 保护机制，Windows 则从 Windows XP SP2 起引入了 DEP 机制，通过设置内存页的属性标记决定该内存页指令是否可执行。图 1 中虚线展示了普通注入攻击被阻止的过程。中央处理器（central processing unit，CPU）一旦检测到攻击代码 shellcode 处于不可执行区域，就会终止当前进程，从而可有效缓解代码注入攻击。

然而，Shacham[1] 在 2007 年提出一种构造巧妙的代码复用攻击——返回导向编程（return-oriented programming，ROP）攻击。ROP 攻击不再复用完整函数，仅利用进程代码段（包括链接库代码）中已有的以 ret 指令结尾的合法指令片段（称为 ROP gadget），就能实现图灵完备的任意攻击功能，是一种典型的代码复用攻击。

图 1 中的实线箭头展示了攻击者利用 ROP 攻击完成减法功能的过程。首先，攻击者在攻击前设计好由 gadget 组成的恶意代码及栈布局（步骤 1、2、3），然后通过漏洞利用（如缓冲区溢出）改变栈布局（步骤 4），使控制流转向预先设计的恶意代码并执行。

ROP 攻击中存在大量以 ret 结尾的指令，容易被检测，并且完全依赖栈进行控制流劫持，随着代码复用攻击技术的发展，出现了多种新型代码复用攻击。林志添[2] 提出一种依赖于堆的 ROP 攻击，利用整形溢出漏洞劫持控制流，将 gadget 的首地址与数据填写到堆空间中，从而发起攻击。文献[3-4]提出

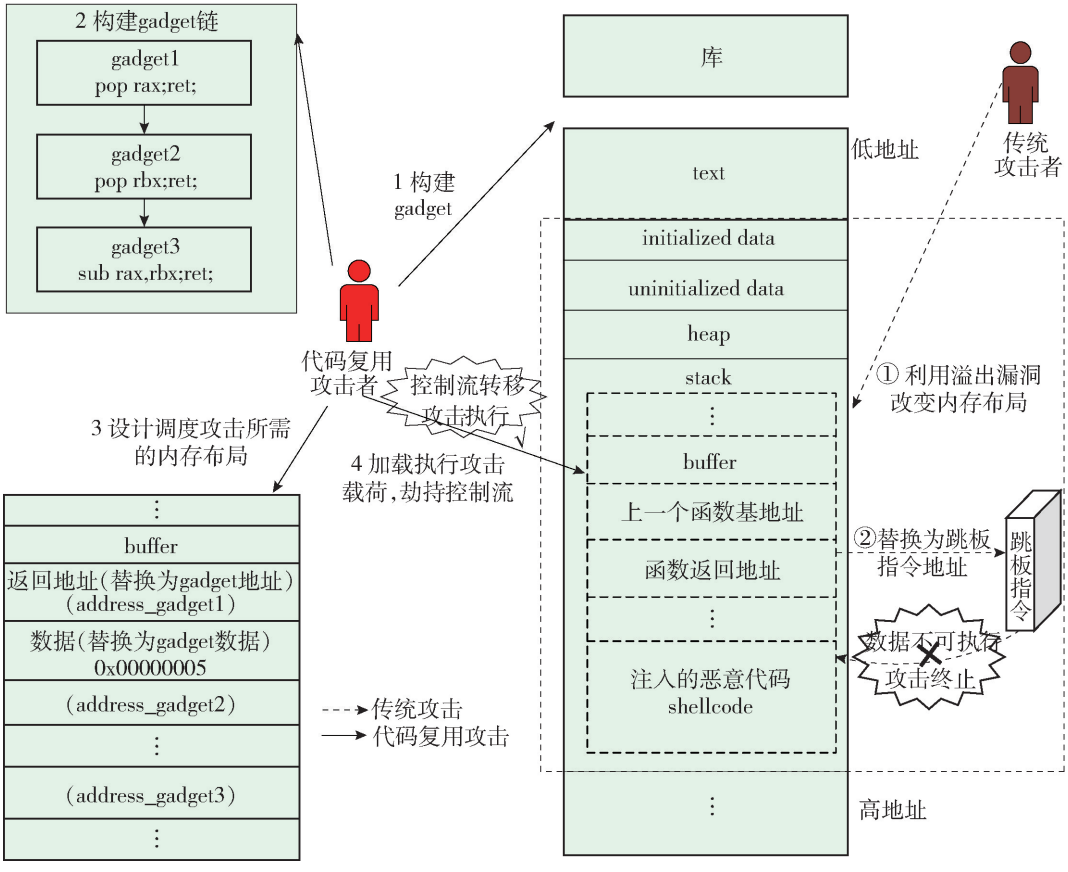

图 1　典型的代码复用攻击：ROP 攻击

Fig. 1　Typical code reuse attack：ROP attack

可以不从栈中获取目标地址的以 jmp 结尾的 gadget,实现了不依赖栈的跳转控制。随后 Bletsch 等[5] 提出了跳转导向编程(jump oriented programing,JOP)攻击,通过在内存任意位置维持的虚表实现对 gadget 首地址与参数的合理布局,构造 gadget 调度器(如 add edx,4;jmp [edx])控制程序指令流,利用以 jmp 结尾的功能 gadget(如 mov [ecx],eax;jmp [edi])将控制再次返回到 dispatcher gadget,使其可以继续执行下一条功能 gadget。因此,JOP 攻击不再基于栈实现程序跳转且可以使用任意寄存器作为程序计数器,随后出现的面向分支指令编程(branch instruction-oriented programming, BIOP)攻击[6]也不再依赖栈传递控制流,而是结合 call 和 ret 结尾的 gadget 和寄存器进行跳转。然而,上述攻击在可用寄存器数量有限时,共用同一个寄存器的多 gadget 会出现数据冲突,因此,此类构造难度较大。面向调用编程(call-oriented programming, COP)攻击[7]部分缓解了寄存器冲突的问题。COP 攻击使用以 call 指令结尾的 gadget 进行控制流跳转,不需要 dispatcher gadget,跳转控制不再由寄存器的值,而是由间接内存位置决定。如此一来,COP 攻击除了要覆写特定间接 call 指令位置,还需要控制栈,通常需要多次漏洞利用才能达到目标。纯面向调用编程(pure-call oriented programming, PCOP)攻击[8]简化了 COP 攻击,改为依赖专门的 trampoline gadget(如 pop eax; popad; cld; call dword [eax-0x18];)进行跳转控制。随后,又出现了面向循环编程(loop-oriented programming, LOP)攻击[9]、面向功能编程(function-oriented programing, FOP)攻击[10]以及面向伪造对象编程(counterfeit object-oriented programming, COOP)攻击[11],分别通过组合由函数、C++虚函数构成的 gadget 链实现攻击,也不再依赖栈,而是通过专用调度指令实现攻击,但上述攻击的本质仍然是通过复用合法指令片段形成攻击代码。

由图 1 可以得到代码复用攻击的 4 个关键步骤及相关特征。

1) 构建 gadget

gadget 就是以间接跳转指令(如 call、ret、jmp)结尾的合法指令片段。代码复用攻击的功能完全依赖目标机器中的 gadget 实现,因此,攻击者需要事先获取所需的 gadget。

图 2 展示了通过系统调用实现攻击的不同类型代码复用攻击所需的 gadget。这里的 gadget 用于获

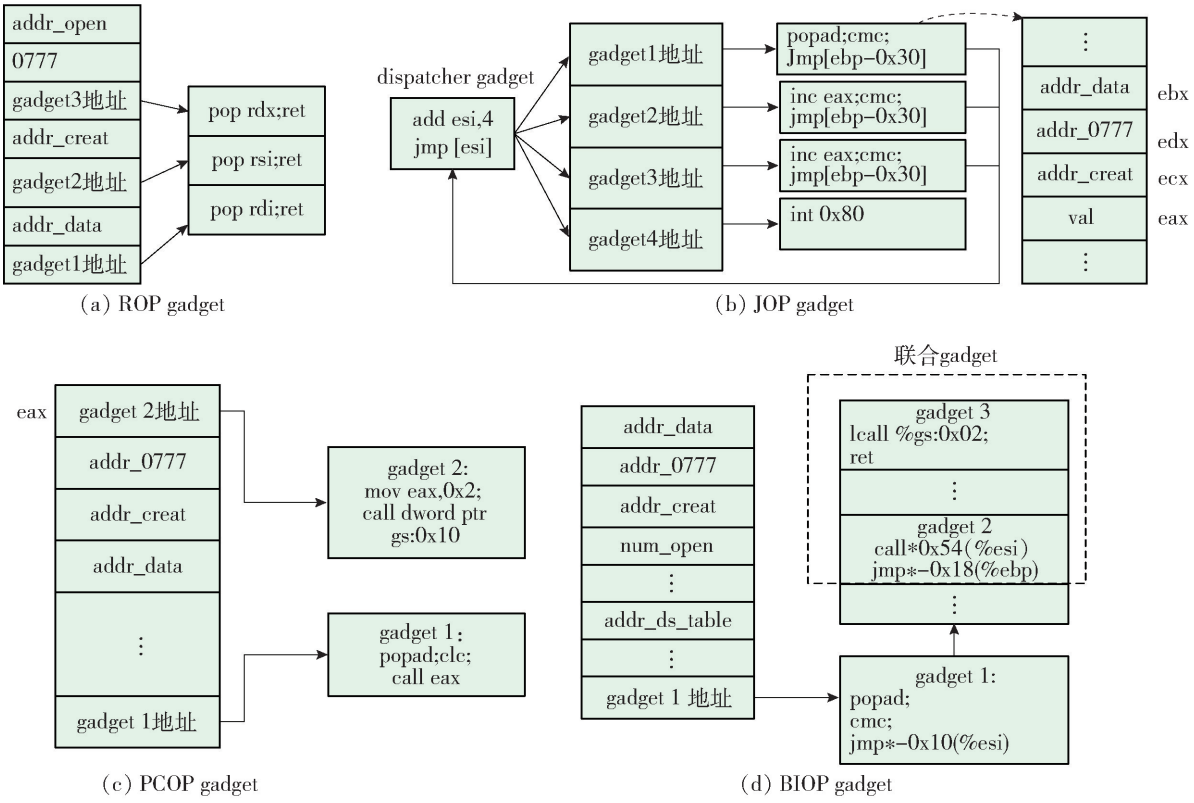

图 2 系统调用参数获取的 gadgets

Fig. 2 gadgets for system call parameters obtainment

取系统调用所需的参数。其中：ROP 攻击用 pop reg + ret 指令从栈中获取参数存放到寄存器 reg，然后跳转到下一个 gadget（获取下一个参数或执行系统调用）；JOP 攻击获取参数的方式更自由，可用 popa + jmp 指令将部署在栈中的数据按照一定顺序取出，并存放到多个寄存器，然后跳转到下一个 gadget，但这会消耗数量有限的寄存器，因此，还需要通过 mov reg1，reg2 + jmp 类指令对寄存器内容进行调整。以 call 结尾的 gadget 构造方式与 JOP 类似，可使用形如 popa + call 或 mov reg1，reg2 + call 的方式构造 gadget。联合 gadget 则通过使用形如 call reg + jmp 与 call % gs：0x10 + ret 的方式联合 jmp、ret 等结尾指令构造联合 gadget，实现系统调用攻击。

所有代码复用攻击都涉及寄存器，为了减少不同 gadget 之间的寄存器使用干扰，攻击者将优先选择指令长度较短的 gadget。

2）构建 gadget 链

攻击功能可能涉及多个命令执行及相关参数获取，因此，需要多个 gadget 组合完成最终攻击。其中，ROP 攻击由以 ret 结尾的 gadget 构成，依赖栈调度 gadget 执行，而以 jmp 和 call 结尾的 gadget 通常依赖专门的调度表构成 gadget 链。

攻击越复杂，gadget 链的长度也就越长，通常大于一个阈值，否则，无法完成特定功能。

3）设计调度攻击所需的内存布局

代码复用攻击需要根据目标机器的内存布局设计攻击所需的 gadget 地址和参数。

在 ROP 攻击中，攻击者将所有的 gadget 地址和 gadget 执行过程中使用的参数构造为攻击载荷，填入栈中，然后，通过栈调度实现恶意代码执行流。对于不完全依赖栈的 JOP、PCOP、BIOP 攻击，攻击者同样需要将 gadget 地址和参数构造为攻击载荷，填充到堆、栈或特定内存区域，通过特定调度代码或者联合栈共同实现恶意代码执行流。由此可见，完成代码复用攻击的关键之一在于获取目标机器上的 gadget 地址，这决定了程序是否会按照攻击者的预期执行。

4）加载执行攻击载荷，劫持控制流

代码复用攻击通常利用现有操作系统和软件中广泛存在的软件漏洞（如缓冲区溢出），将正常程序劫持到首个 gadget，然后依次执行 gadget 链上的指令，达到篡改特殊数据（如栈中的函数返回地址、堆中的函数表地址等）、劫持程序控制流的目的。

综上所述，代码复用攻击是一种利用系统内存中已存在的合法指令构成的新型攻击，没有引入在目标系统中原来不存在的代码，对传统的恶意代码检测与防御机制是一种重大挑战。

2　代码复用攻防技术演化

2.1　攻防特征及技术演化

安全技术的发展实质上是一个攻防对抗发展的过程。一些代码复用研究[12-14]从安全编译、控制流完整性、地址随机化 3 个角度分析现有防御机制。文献[15]从造成内存破坏的原因出发提出缓解措施。文献[16]从攻击改进与防御 2 个角度进行研究。按攻击特点分为利用函数、利用转移控制流、抗地址随机化、动态生成 gadget 这 4 类方式；按防御特点分为控制流完整性与地址随机化 2 类方式。上述研究多从一种或几种攻击或防御技术出发，独立论述某种代码复用攻击或防御进展，对攻击关键阶段技术分析不够完整。

通过对代码复用攻击 4 个关键步骤的分析，可以将代码攻击成功的 3 个本质特征归纳为：代码复用攻击指令的结构特征、攻击代码所需的内存地址空间布局特征以及攻击代码与正常代码不同的运行时特征。其中，结构特征主要是从攻击者事前构造攻击的角度分析得到的，而内存地址空间布局特征和代码运行时特征则是从主要防御者事前预防攻击和事中检测攻击的角度分析得到的。基于上述 3 类本质特征进行代码复用攻防对抗的技术发展时间线如图 3 所示。

2.1.1　基于攻击指令结构特征的攻防技术演化

代码复用攻击包含大量的以间接跳转、调用、返回指令结尾的 gadget，并通过在目标库或可执行文件中搜索链接 gadget 完成任意恶意攻击。与正常代码比较，攻击代码有以下较为明显的结构特征：1）构成攻击的 gadget 结尾必须为 ret、jmp 或 call 指令，gadget 通常会调用特定的内存相关接口（如 virtualalloc 等）实现攻击，因此，gadget 中可能包含这些接口参数信息，gadget 的高位地址也通常在特定目标库地址空间范围内。2）为了避免带来数据冲突等副作用，gadget 指令自身长度通常较短，而完成攻击功能需要多个连续 gadget 组合成 gadget 链，因此，组成攻击代码的 gadget 链长度通常会大于 1 个阈值。

可根据构成攻击的 gadget 特点将攻防技术分为如下 2 类。

1）基于 gadget 指令内容特征分析的攻防技术

为了避免为攻击提供 gadget，一些研究通过修改编译器等方式替换和减少 gadget 所需的指令，缓

图 3 代码复用攻防对抗时间线

Fig. 3 Timeline of code reuse attack/defense confrontation

解攻击,例如,Li 等[17]提出了一种能够去掉 ret 指令的编译器。虽然该方法性能消耗较小,为 3.50% ~ 5.78%,但显然只能防御 ROP 攻击且需要被保护程序源码。G-Free[18]通过修改编译器、消除未对齐的自由分支指令(ret/jmp/call 指令)等多种方式,减少以上述跳转指令结尾的 gadget 数量,总体性能损失小于 5.6%。

ROPminer[19]根据已知恶意代码库中 gadget 内容是否包含特定接口参数和其地址是否处于常见库地址空间等特征,生成隐马尔可夫模型,然后进行 ROP 攻击检测,其准确性取决于已知 gadget 库的完备性。另外,如果 gadget 地址是动态生成的,该方法将会失效。

2) gadget 及 gadget 链长度特征分析的攻防技术

ROP 攻击利用以 ret 指令结尾的短指令序列 gadget 构造攻击。因此,Chen 等[20]提出了短指令阈值检测方法 DROP 对 ROP 攻击进行防御,通过动态二进制插桩框架 valgrind 拦截 ret 指令,检测每个

gadget 的大小是否小于阈值 5 且 gadget 链长度是否大于阈值 3 来判断是否存在攻击行为。由于采用了动态插桩技术，平均性能损失达到了5.3 倍。

除了 ROP 攻击，以 jmp 及 call 指令结尾的 JOP、COP 等攻击也相继被提出。文献[21-24]研究了针对上述代码复用攻击的 gadget 指令阈值特征。文献[21]中，kBouncer 判断连续 8 个指令数小于 20 的 gadget 是代码复用攻击。文献[22]中，ROPecker 通过正常程序与攻击程序比较，判断连续出现 11 次以上指令条数小于 6 的 gadget 是攻击。文献[23]提出基于多个阈值综合检测的 JOP 攻击评估方法 JOP-alarm，该方法可以基于插桩软件或改造硬件实现。文献[24]提出采用动态插桩工具 Pin 的基于系统调用触发的 ROP 检测方法 ROP-hunt [24]，该方法性能损失约为 1.75 倍。上述基于阈值的防御方法虽能检测出大量代码复用攻击，但仍存在误报率与漏报率，并容易被攻击者精心构造的攻击绕过阈值检测。Goktas 等[25]对基于 gadget 指令特征防御技术的劣势进行了分析，认为阈值的错误设置后果严重，可能会将程序的正常执行标记为攻击或其他情况。

文献[7,26-28]提出了一些加长 gadget 或缩短 gadget 链的改进方案，以规避上述防御机制。文献[7]通过构建指令长度超过 20 的 gadget 破坏阈值检测条件，绕过 kBouncer 和 ROPecker。文献[26]利用函数调用创建较长的不影响攻击代码功能的 lgadget 来破坏阈值检测条件。文献[27]针对 JOP 攻击比 ROP 攻击需要更多 gadget 的特点，提出一种基于修改硬件的 SCRAP 架构，性能损失较小，约为 2%。文献[25]中，通过调用函数的 call 指令创建的 delay gadget 能够被正常指令掩盖，从而绕过阈值检测机制，并能减少使用简单加长 gadget 覆盖大量寄存器和/或内存位置导致程序状态难以控制的问题。文献[28]中，Tiny JOP 则通过尽可能减少攻击过程中连续 gadget 的数量绕过现有的阈值检测方法。由于没有 dispatcher gadget，构造的 gadget 链长度减少，能够执行低于阈值的攻击。类似地，文献[8]提出了 Tiny COP。上述工作都能够绕过 JOP-alarm、SCRAP 等检测机制。

基于上述特征的攻防技术演化过程以时间线的形式表示，如图 4 所示。带方向的箭头表示一种攻击或防御方法出现之后出现了针对性的防御或攻击方法。

总之，通过消除 gadget 结尾跳转指令减少攻击

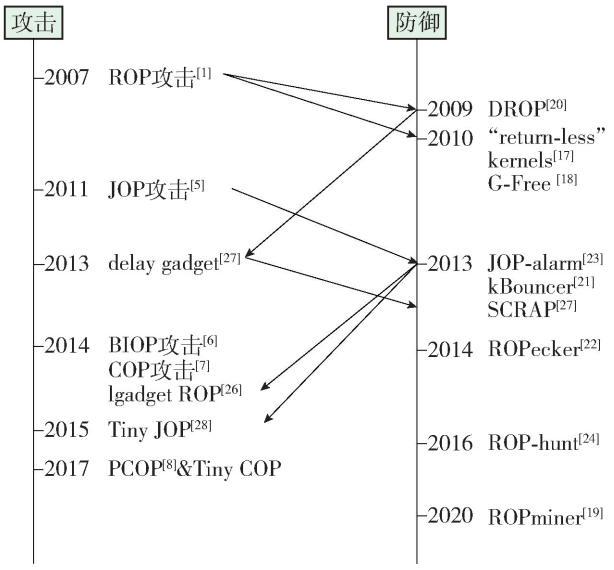

图 4　基于攻击指令结构特征的攻防演化时间线

Fig. 4　Attack/defense evolution timeline based on attack instruction structure characteristics

的方式需要通过改进编译器重新编译源码，难以广泛应用。基于攻击指令内容和指令结构特征的研究主要围绕 gadget 结尾指令、所包含的特定常量信息特征检测、指令长度和连续数量的阈值检测进行。如果攻击者了解防御方的特征和检测阈值，就可以通过精心构造变种 gadget 绕过阈值检测机制；同理，对基于 gadget 特征的防御而言，特征数据是否完备、阈值设置是否合理至关重要，否则，必然发生误报、漏报。后续研究可考虑综合使用多种特征数据，应用机器学习、深度学习方法，进行黑盒学习，更准确地检测代码复用攻击。

2.1.2　基于地址空间布局特征的攻防技术演化

ASLR 机制最早通过对段地址进行随机化增加了攻击者预测 gadget 地址的难度，使其无法了解进程内存空间布局，从而无法编写有效的漏洞利用负载 shellcode。

根据地址空间布局随机化粒度和实时性可将攻防技术分为如下 3 类。

1）面向粗粒度地址随机化的攻防技术

Linux 分区属性交叉（partition attributes across，PaX）实现了 ASLR 机制，通过向用户进程地址空间中的执行区（包括可执行代码等）、映射区（包括堆、动态库等）、用户栈的基地址分别增加偏移变量的方式进行随机化。用户通过内核参数 randomize_va_space 设置 0、1、2 这 3 个级别来控制系统级 ASLR 的禁用、开启以及 ASLR 机制的作用位置。级别 2 在级别 1 的基础上增加了堆的随机化。Linux

还提供了位置无关可执行(position independent executables,PIE)程序机制保证程序每次执行都加载到不同内存位置,但需要重新编译源码[29]。Windows 也从 Vista 开始提供 ASLR,随机化了可移植可执行(portable executable,PE)文件映像、堆、栈等的基址等。由于随机化技术将代码复用攻击所需的 gadget 内存地址进行了随机化转换,传统代码复用攻击将失效。

因为当前操作系统部署的地址随机化只简单将加载模块的基地址随机化,所以模块内部的函数、指令、基本块布局未发生改变。为了绕过地址随机化机制,攻击者通常通过内存泄漏、暴力破解随机值[30-31]的方式泄漏并推测出随机化后的内存布局(例如,通过暴力破解泄漏 libc 库中的某个函数地址,从而推测出 libc 库的基地址偏移量),从而绕过粗粒度 ASLR。

2)面向细粒度地址随机化的攻防技术

尽可能提高随机化粒度,可以增加攻击者根据基地址泄漏信息获取进程空间布局的难度。例如,Kil 等[32]提出了 ASLP 防御方法,通过重写可执行文件和 elf 文件,重新随机化排列代码段函数和数据段变量,实现代码段和数据段函数粒度的随机化,但 ASLP 需要额外的重定向信息,少数没有该信息的库无法使用 ASLP。Pappas 等[33]提出 in-place code 随机化,进一步通过交换寄存器的值生成随机化应用程序,几乎无性能消耗,但只对约 80% 的 gadget 具有防御效果。文献[34]提出的 ILR 则通过更改指令顺序或使用语义相同的等效指令替换原有指令的方式,实现了更细粒度的位置无关的指令级别随机化,但 ILR 需要目标应用在虚拟环境中提前加载,并进行静态分析,这会带来约 13% 的性能消耗。基本块级随机化 binary stirring[35]采用二进制重写工具在每次程序加载到内存时通过重新排列基本块顺序进行细粒度随机化,性能损失约为 1.6%。

上述方法每次都采用同样方式随机化目标程序,而 Gupta 等[36-37]进一步提出基于函数粒度的地址随机化方案 Marlin,通过工具提取函数符号信息,并通过随机化排列函数符号得到随机化的进程地址空间,使代码启动过程平均延迟约 0.53 s,但基本不影响代码运行效率。另外,虽然 Marlin 的随机化粒度不如基本块随机化,但能在每次执行目标代码时随机化,增加了时间随机因素,启动消耗大。虽然细粒度随机化增加了通过一次性泄漏获得内存布局的难度,攻击者仍可以通过获取一个 gadget 地址,或多次泄漏内存实施代码复用攻击[38-39]。

3)实时随机化

Snow 等[39]提出的实时 ROP(just-in-time ROP,JIT-ROP)能够突破细粒度随机化防护机制,即通过泄漏初始代码指针得到和反汇编指针所在的按照 4 kB 对齐的内存代码页,然后利用其中的跳转指令地址泄漏得到更多内存页,反复上述过程,获取接口调用指针和 gadget 指令,最后通过实时编译得到攻击代码。JIT-ROP 攻击过程在不同系统环境下耗时为 2.30 ~ 22.50 s。另外,由于浏览器使用即时编译引擎动态生成和执行机器码,JIT-ROP 攻击也能利用浏览器漏洞在其生成的机器码中查找 gadget,实施攻击。袁平海等[40]指出,主流浏览器动态生成的代码中通常存在一些固定指令序列,不受 ASLR 的影响,令攻击者能生成图灵完备的 gadget 链。实时攻击的核心之一是通过反复泄漏得到足够多的内存布局信息。因此,一些研究提出修改操作系统的内存管理系统[41-42],通过设置“代码 XnR”的方式防止直接代码泄漏,性能损失仅为 2.2% ~ 3.4%。文献[43]中Oxymoron 通过读取、反汇编、转换指令、重新生成可执行文件和共享库,将代码和数据引用替换为标签,消除 jump 直接寻址指令,从而防止直接代码泄漏。此过程需要消耗少量时间转换可执行文件和库,性能损失约为 2.7%。然而,文献[44]指出 Oxymoron 仍然无法阻止攻击者通过泄漏堆栈数据结构中的代码指针进行间接代码泄漏。

许多研究在随机化时效方面展开了防御 JIT-ROP 研究。主要思想是基于特定事件或者在进程生命周期内按照固定的时间间隔重新随机化排列进程内存布局,使攻击者无法在短时间获取足够多的信息构造 payload 或使已经构造的 payload 无效,但高频随机化也会导致较大的性能开销。例如,TASR[45]在每次触发特定 I/O 系统调用后进行不同随机化,性能损失约 2.1%。文献[44]提出执行路径随机化,在返回时随机执行基于相同语义的不同结构的程序副本路径。由于采用了动态插桩技术,性能损失约为 19%。Chen 等[46]提出的 JIT-ASR 通过扩展内核虚拟内存管理模块实现实时随机化,通过修改运行时代码地址的页号和页表,使随机化后的虚拟地址持续改变,性能损失约为 1.2%。文献[47]则结合取指访存和进程地址空间动态随机化使攻击者已经掌握的进程地址空间布局历史信息无效。Hawkins 等[48]提出 Mixr,为现有软件程序和库提供了高效和可灵活选择随机化粒度的运

行时 ASLR，并且不需要目标程序或库的源码、调试信息，性能损失约为 2%。Ahmed 等[49]进一步通过分析常见工具、应用（含浏览器）和动态库，提出了破坏 gadget 图灵完备性的随机化间隔时间定量分析方法，发现随机化间隔时间最大范围为 1.5～3.5 s，ROP 攻击使用的泄漏指针的地址对 gadget 链的收敛速度有一定的影响。

基于地址随机化技术的攻防演化时间线如图 5 所示，从地址随机化粒度及实时性的角度可以看出代码复用攻击攻防技术的演化过程。

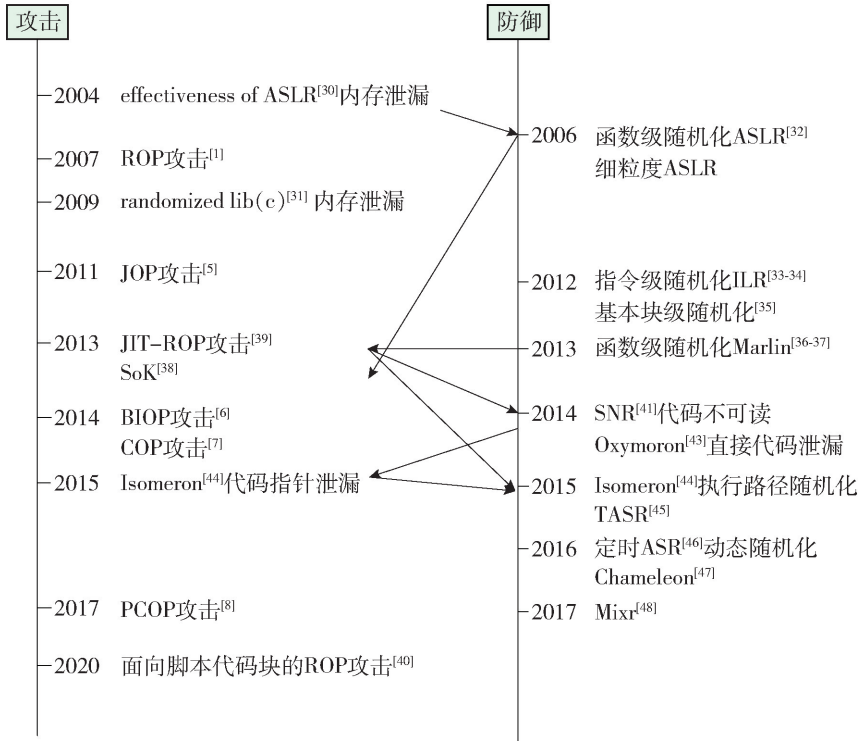

图 5　基于地址随机化技术的攻防演化时间线

Fig. 5　Attack/defense evolution timeline based on address randomization

综上所述，地址随机化防御技术的发展十分充分，防御粒度已到指令级，是一种对各类代码复用攻击都有效的通用防御方法。然而，实时攻击方法（如 JIT-ROP）仍能通过多次内存泄漏攻击仅在程序启动时进行一次随机化的细粒度随机化防御。可执行代码及代码指针隐藏保护能够缓解地址泄漏问题，而细粒度且实时地址随机化能够更有效地阻止 JIT 代码复用攻击，但在增加随机化熵值、扩大随机化范围、增加时间随机化的同时，也带来了大量系统性能开销。

2.1.3　基于运行时代码特征的攻防技术演化

正常代码与复用攻击代码的本质区别在于两者的程序流不同。

基于运行时代码特征的攻防技术可以分为传统粗粒度控制流、现代细粒度控制流和非控制流攻防技术 3 类。

2.1.3.1　粗粒度控制流攻防技术

CFI 方法于 2005 年由 Abadi 等[50]首次提出，通过向函数调用和返回时插入 ID 与 idgain，监视程序运行中间接 jump 和 call 指令以及 ret 指令，在程序运行时判断是否出现不同于提前静态分析得到的图的异常。经典 CFI 的安全性严重依赖预先通过静态分析得到的控制流图的准确度，其是基于 Vulcan 架构通过进行插桩实现的，性能开销较大（约 16%）。因此，2013 年出现了基于简化控制流图进行粗粒度控制流检测的研究。Zhang 等[51]提出性能开销较低的依赖重定位表信息的粗粒度 CFI 的 CCFIR 方法，利用重定位表获取间接分支 call、ret、jmp 的合法目标跳转地址的集合并组成白名单，通过重写二进制文件开辟一个单独的节存放这些合法跳转集合。程序在执行间接跳转指令时判断该指令的目标地址是否是该节内的合法跳转地址，从而检测是否发生控制流劫持攻击。CCFIR 将传统 CFI 策略中的双 ID 扩展为 3 个 ID，进一步区分返回指令目的地址是敏感函数还是非敏感函数，从而有效地避免 ret 跳转到敏感函数执

行。该方法也存在一定问题,其保护的系统共享库依赖特定平台,有一定兼容性问题。Zhang 等[52]提出的 binCFI 仍采用了基于合法跳转目标集合的白名单,但与 CCFIR 不同的是,对合法目标集合按照间接控制指令的操作数进行了进一步细分,粒度更细,安全性也更高,但相应的性能消耗与兼容性问题更大。Huang 等[53]则基于蜜罐思想,提出了一种诱骗检测 ROP 攻击的方法,通过匹配最近分支记录(last branch record,LBR)检测到的代码地址与防御方预设的蜜罐 gadget 地址检测 ROP 攻击。Honeygadget 有 2 个组件:静态处理模块与运行时检查模块。静态处理模块预先定义敏感函数调用列表,记录能够提权的系统调用等核心函数地址。运行时检查模块比较源代码与加入 Honeygadget 的合成代码,确定蜜罐位置,生成地址列表。当程序执行敏感函数时,运行时检查模块暂停程序执行,将 LBR 记录的实际指令地址与蜜罐位置地址列表比较,若匹配则判定为遭受 ROP 攻击。该方法检测速度较快,平均性能损失为 6.8%,但如果攻击者绕开 Honeygadget 的部署位置,该方法将失效。

根据粗粒度控制流相关检查内容可将控制流攻防技术分为 2 类。

1) 栈及间接跳转地址特征的检查

经典 ROP 攻击通过将 gadget 地址与数据合理安排在栈中,使用 ret 指令劫持程序执行流。后续出现的 JOP、PCOP 等攻击通过 jump 及 call 指令进行程序流劫持。可见间接跳转指令的地址是保证程序控制流正常的关键。

相关技术可分为如下 4 类。

① 影子栈

粗粒度 CFI 提出 ret 指令的目标地址应该在某个 call 指令的下一个地址处,与之相比,影子栈实际上是更严格的粗粒度 CFI 策略,要求 ret 指令的目标地址应该与相关的 call 指令一一对应。ROPdefender[54]使用二进制动态插桩技术实现了影子栈,将 call 指令的返回地址同时压入真正的栈和影子栈中,通过在程序运行时比较检测到的 ret 指令与影子栈中的返回地址的值是否相等进行 ROP 攻击检测。因采用了动态插桩技术,性能消耗达到 2 倍。另外,影子栈也存在不能防御 JOP 等变种攻击,并且影子栈自身也缺少保护的问题[55]。文献[56]提出了篡改栈与影子栈中的返回地址绕过 ROPdefender 使用的影子栈的方法。Buddy stacks[57]给出了一种避免维护单独影子栈指针的新型堆栈布局及基于栈的本地线程存储机制,并且能连续重新随机化影子栈中的返回地址,具有支持多线程、高性能、与现有代码兼容、防止信息泄露的优点。

② 栈中返回地址加密

文献[18]提出通过在函数入口点对栈上保存的返回地址加密,在返回时对加密地址解密,可保护栈中的返回地址,但未实现。陈林博等[58]通过二进制动态插桩工具对栈中存放的返回地址进行加密,实现了对返回地址的保护。

③ 栈完整性检查

金丝雀(canary)检查是一种广泛应用的完整性检查方法。文献[59-60]通过在堆栈中的缓冲区与关键数据之间插入和检测 canary 实现对关键数据(如返回地址)的保护。2014 年提出的 BROP[61]攻击给出了一种通过暴力破解 canary 绕过防御机制的方法。该攻击通过测试是否触发能自动重启的目标代码崩溃的方式猜测 canary 的每个 bit 位,并在获得崩溃点返回地址后回传攻击代码,实施经典 ROP 攻击。

④ 基于 jump 及 call 指令间接跳转地址的检查

粗粒度 CFI 针对上述 JOP、BIOP、COP、PCOP 等不依赖栈的攻击也有相关防御方法。例如:检查间接 jump 指令的目标地址是否在另一个函数的起始位置或者在本函数的中间位置,从而防止 jump 越界[62];检查间接 call 指令的目标地址是否在一个函数的开头[63]。

2) 基于 ret/jump/call 跳转指令统计特征检查

除了上述对 ret/jump/call 指令地址进行检查,还出现了针对这些指令统计特征的检查方法,这类方法目前主要基于硬件实现,比早先的粗粒度完整性检查方法的效率高很多。例如:正常程序总是先使用 call 指令调用函数,再通过 ret 指令结束调用,因此,call 指令数量会大于等于 ret 指令,这是一个 call/ret 指令数量统计特征。ROP 通过 ret 指令劫持程序控制流导致 ret 指令远远多于 call 指令,可以利用上述 call/ret 指令数量统计特征检测 ROP 攻击。zero-sum[64]利用上述特征,通过修改编译器,设置线程局部变量,并在函数入口处将变量加 1,在 ret 指令执行时将变量减 1,根据变量值确定是否发生 ROP 攻击。但上述方法需要修改源码,实用性不高。文献[21]利用 LBR 寄存器在 Windows7 上通过仅检测挑选过的关键功能(如特定 Windows 接口)

实现了低开销的检测 ROP 攻击的 kBouncer，降低了运行时性能消耗，但有可能漏报。该方案用 LBR 寄存器中记录间接跳转前后程序计数器的值，判断 ret 指令跳转的目标地址的上一条指令是否是 call 指令，从而检测出 call/ret 指令不平衡的 ROP 攻击，但 LBR 也有被污染的可能。文献[22]提出的 ROPecker 扩展了 kBouncer，通过分析程序执行流跳出滑动窗口时 LBR 寄存器中的信息判断是否存在异常，进一步降低了性能消耗。文献[63]提出 CFIMon，利用 Intel 提供的分支跟踪存储(branch tracing store，BTS)及 LBR，基于扩展内核模块实现硬件性能事件分析，监控其是否违反了之前定好的执行流跳转规则，保护程序执行流完整性。该方法有一定检测延迟，性能损失约 6.1%。文献[65]提出 Mibchecker，利用硬件性能管理单元(performance monitor unit，PMU)的事件触发机制，针对每个预测失败的间接分支进行 ROP 攻击检测，规避了历史刷新攻击的可能，同时，提出基于敏感系统调用参数的检测方法来检测短攻击链(gadgets-chain)攻击，Mibchecker 性能开销比较小，仅为 5.7%。HDROP[66]利用 PMU 上的硬件性能计数器(performance monitoring counters，PMC)计算错误预测执行 ret 指令的数目和执行 ret 指令的数目，并与使用人工神经网络预先训练好的检测模型对比，实现函数级 ROP 攻击的检测。其性能损失与插入内核的检测点数量相关，在插入 3 000 个检查点的情况下，性能损失约为 19%。文献[67]发现 ROP gadget 链分布在不同程序段，在执行过程中会不断地在多个页面跳转，导致大量指令转译后备缓冲区未命中(instruction translation lookaside buffer，ITLB)和数据转译后备缓冲区未命中(data translation lookaside buffer，DTLB)，提出 ROPDetector 方法，通过检测一个周期内 ITLB、DTLB 未命中率与错误预测执行 ret 指令占总 ret 指令的比重是否超过阈值判断是否遭受 ROP 攻击，系统性能消耗为 5.05%。HadROP[68]采用支持向量机对 ROP 程序行为相关的硬件性能计数器值集合进行训练，得到了区分 ROP 攻击的分类器，性能损失约为 5%。但 HadROP 提出的定时触发中断检测方式的时间间隔如果太长，可能被攻击者绕过。HBROP[69]在 HDROP 与 HadROP 的基础上，仍以函数为性能事件采集粒度，基于敏感系统调用触发检查，采用聚类算法，通过增加性能事件值的选取实现了更多类别的代码复用攻击检测，减少了误报与漏报。

不过，由于上述粗粒度控制流防御方法仅粗略判断间接控制指令的目标跳转的地址是否符合简单的 call-ret 对应，是否符合跳转地址限制要求或白名单，对上下文不够敏感，不能完全防御与正常程序行为更加接近的特殊类型代码复用攻击。例如：LOP[9]通过精心构造的循环 gadget 和功能 gadget 进行控制流劫持，使得循环 gadget 中的 call 指令与功能 gadget 中的 ret 指令能够一一对应，不仅能绕过粗粒度 CFI，还能绕过影子栈的 call/ret 指令检查。文献[10]提出 FOP，针对 C 程序的代码复用攻击，通过篡改函数指针调度以函数为单位的功能 gadget 进行控制流劫持，不需要修改返回地址，可以绕过影子栈及粗粒度 CFI。COOP[11]通过修改虚函数表将 C++虚函数作为 gadget 调用，从而达到劫持控制流的目的。上述代码复用攻击由于不需要修改栈中的 gadget 返回地址并符合 ret/jump/call 跳转检查规律，能够绕过粗粒度 CFI。

2.1.3.2　现代细粒度控制流攻防技术

经典控制流完整性 CFI 通过检查每个间接跳转地址防止控制流劫持，因为消耗太大难以应用。粗粒度 CFI 虽然消耗小，但缺乏间接跳转地址的上下文信息，容易被绕过。为了解决上述问题，出现了一些新的细粒度控制流检查技术，通过提供更准确的指针分类提供控制流保护，如近年来出现的细粒度 CPI 和 CCFI、基于分支相关完整性的上下文敏感控制流完整性方法(control-flow integrity method based on branch correlation integrity，BCI-CFI)。其中：CPI[70]采用保护所有代码指针(包括函数指针、返回地址指针)的方式保护程序控制流，并使用代码指针隔离(code-pointer separation，CPS)区分敏感数据(代码指针)与常规数据，将敏感数据放到安全区以保证代码指针引用安全。CPI 需要修改编译器对 C/C++代码进行静态分析和插桩检查，带来的性能损失约 8.4%，但其提出的安全区需要硬件保护，否则，仍可能被攻击[71]。CCFI[72]通过对所有控制指针(如返回地址和函数指针)进行标记，记录其消息验证码(message authentication code，MAC)，每次加载时进行检查，从而防止攻击者非法交换或修改返回地址和指针，增强程序运行时控制流完整性，性能损失为 3% ~18%。CCFI 在编译器处理编译对象时，重复使用 macptr 和 checkptr 内联函数进行一些敏感数据变量的记录和检查，从而防止程序数据(如索引)遭到篡改导致的控制流变化。BCI-CFI[73]使用基于分支关联性的上下文敏

感 CFI 方法解决了现有 CFI 策略难以区分等价类的问题,但 BCI-CFI 平均运行时开销为 19.67%,性能消耗仍不可忽略。

然而,细粒度 CFI 的安全性取决于对控制流图检查的准确性,这需要进行正确并完备的指针检查,而这是一个不可判定问题[74]。文献[75]提出的 control jujutsu 利用了当前细粒度 CFI 采用的指针的分析检查正确但不完备的特点,仍然能够通过构造篡改参数的间接调用地址(argument corruptible indirect call site,ACICS)作为 gadget 执行任意代码。文献[76]提出了 cscan 和 cbench,用于测量和检查 CFI 的真实边界和对典型攻击的防御效果,发现当前的 CFI 机制的实际边界与 CFI 的策略确实存在差距。

2.1.3.3 非控制流攻防技术

虽然 CFI 相关机制能够检测出实施控制流劫持的代码复用攻击,但却无法防御新出现的数据攻击[77]。数据导向编程(data-oriented programming,DOP)攻击[78]也通过组合 gadget 实现了代码复用攻击,但只操控程序数据,通过内存错误(如溢出)破坏内存关键数据变量,从而构成新的控制流(即攻击)。文献[78]还提出了 4 类 DOP 防御方法,包括防止内存错误的内存安全机制(如类型安全)、数据流完整性保护、细粒度数据流随机化及硬件和软件错误隔离。但由于正常程序中的数据指针太多,后 3 种防御方法性能消耗会过大,若仅实施部分数据保护,DOP 攻击仍可能成功。文献[79]通过系统化分析 DOP 的防御与攻击,对不同 DOP 攻击进行比较,验证了 DOP 对控制流有一定影响的猜想。对于 DOP 攻击,文献[80]提出了 Shapeshifter,定义一系列安全关键数据对象,将内存上的数据结构实例与变量表示进行随机化处理并自适应重新排列,以此防御 DOP 攻击。Shapeshifter 运行开销约为 20.1%,但无法防御利用目标程序的动态链接库的 DOP 攻击。

C-FLAT[81]是一种基于哈希的控制流认证方法,基于 Arm TrustZone 硬件进行了实现。该方法能够防止通过篡改分支判断条件变量和循环变量进行控制流劫持的 DOP 攻击,时间损耗与控制流事件(如一次分支指令、返回指令的执行)数量成正比。2017 年,运行时控制流认证方案 ATRIUM [82]则在 C-FLAT 的基础上考虑了物理攻击,基于硬件进一步增强了控制流认证过程的安全性,防止检测时到使用时(time-of-check to time-of-use,TOCTOU)攻击,减少检测空间和性能消耗。GACFA[83]则基于多目标优化算法平衡了控制流认证的安全性和效率。虽然 DOP 可以绕过 CFI 防御方法,但并不意味着 DOP 攻击不会对 CFI 防御方法产生任何影响,文献[79]验证了 BOP/DOP 攻击对控制流是有一定影响的,实验结果表明,BOP/DOP 攻击会导致不兼容分支行为与频率异常现象。

本文按时间线总结了基于运行时代码特征的攻防对抗发展过程,如图 6 所示。

综上所述,基于运行时特征的代码复用攻击技术的发展呈现出 3 个特点:第一,对栈的依赖性(如基于栈的经典 ROP 攻击)逐渐降低(如 JOP、BIOP、COP、PCOP 攻击等),代码复用攻击构造难度随之增加。第二,新的代码复用攻击向符合正常程序控制流的方向转变。第三,被攻击对象从控制指令相关逐渐转为与控制流无直接关系的内存关键数据(如 DOP 攻击)相关。

当前基于运行时特征的代码复用攻击防御技术的发展也十分迅速,相关特点包括:第一,从性能消耗较大的经典 CFI 转向性能消耗较小的粗粒度控制流完整性检查和机密性保护,尤其是基于硬件辅助的粗粒度控制流检查能够大幅降低检测性能消耗。第二,控制流保护力度从粗粒度保护转向细粒度。第三,出现了基于机器学习的智能化控制流指令规律检测方法。第四,出现了控制流验证相关的数据流检测方法。

2.2 代码复用攻防技术发展规律

根据代码复用攻击原理及特征分析,主流攻防技术特征可分为 3 个大类 9 个小类,9 个小类中的攻击和防御技术分别记为 A1 ~ A9 和 D1 ~ D9,如表 1 所示。

对攻防技术的比较可以从技术自身和外部评价 2 个方面入手,归纳得到的指标体系如表 2 所示。

不同类别代码复用攻防技术攻防成功率如表 3 所示。从攻防技术自身技术特征可知,由于不同大类攻防技术原理不同,大类之间的攻防技术基本不相容(相互之间无法攻击或防御成功),但基于运行时代码特征的技术例外。例如,粗粒度 CFI 的攻击(A6 和 A7 类攻击)、现代细粒度 CFI 攻击(A8 类攻击)、非控制数据流攻击 DOP 攻击[78](A9 类攻击)都能成功攻击部分或全部第 1、2 类防御(D1 ~ D5)的方法,这是因为第 3 大类攻击中较高级的攻击技术(A7 ~ A9)都需要包含内存泄漏类攻击技术(A4 ~ A5),并且不再依赖简单的间接跳转指令结

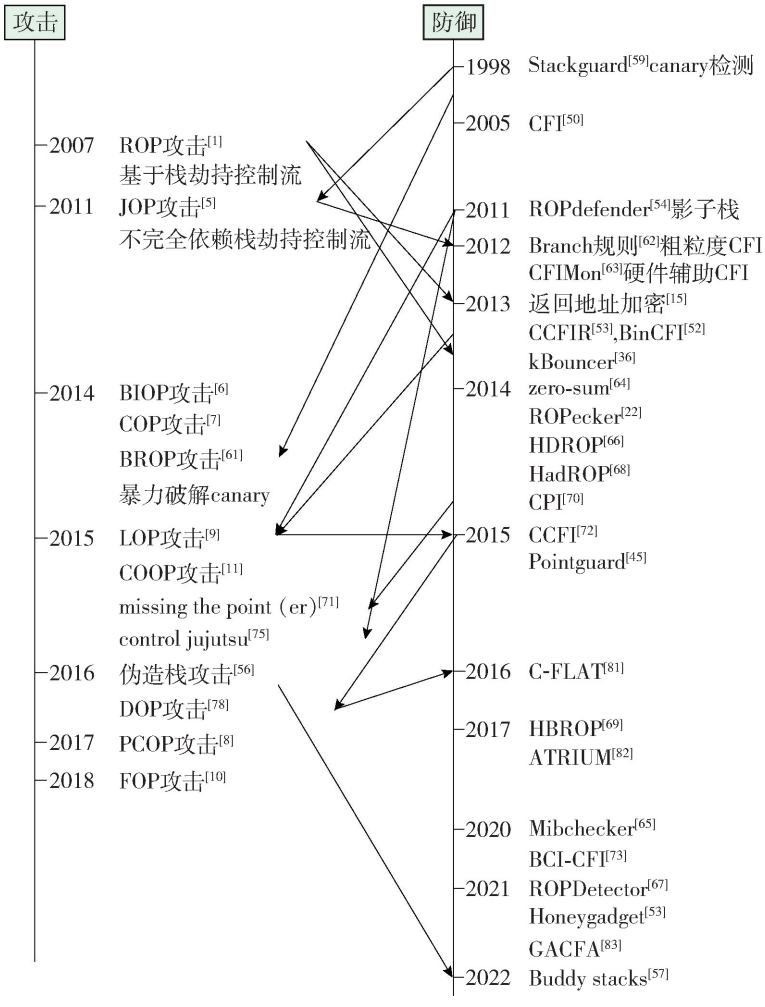

图 6　基于运行时代码特征的攻防演化时间线

Fig. 6　Attack/defense evolution timeline based on runtime code characteristics

表 1　基于特征的代码复用攻防技术分类

Table 1　Feature-based code reuse attack/defense technology classification

攻防技术编号	攻防技术特征大类	攻防技术特征小类
A1/D1	攻击指令结构特征	gadget 结尾指令特征
A2/D2	攻击指令结构特征	gadget 及 gadget 链长度特征
A3/D3	地址空间布局特征	粗粒度地址随机化
A4/D4	地址空间布局特征	细粒度地址随机化
A5/D5	地址空间布局特征	实时地址随机化
A6/D6	运行时代码特征	栈及间接跳转地址特征
A7/D7	运行时代码特征	硬件辅助的间接跳转指令统计特征
A8/D8	运行时代码特征	现代细粒度控制流特征
A9/D9	运行时代码特征	非控制流特征

表 2　代码复用攻防技术评价指标体系

Table 2　Evaluation index system of code reuse attack/defense technology

攻防评价指标	自身评价指标	外部评价指标
攻击技术评价指标	面对不同防御技术的攻击成功率	依赖代码语言
		依赖操作系统
		依赖额外信息
防御技术评价指标	面对不同攻击技术的防御成功率	依赖特定指令结尾攻击
		依赖保护对象源代码
		需要修改硬件/操作系统
		依赖额外信息
		运行性能损失

构特征。另外,数据流代码复用攻击(如 DOP 攻击)实现方式与原有针对控制层面攻击的代码复用攻击不同,因此,基本上能攻击目前所有防御机制;但目前尚未出现较为完善的数据流攻击防护方法,因此,仅能讨论其理论防御成功率。假设数据流攻击防护方法能够防止内存错误(如类型安全)[78],那么,其能防御所有现有针对控制层面攻击的代码复用攻击及部分 DOP 攻击。

表 3 代码复用攻防技术比较

Table 3 Comparison of code reuse attack/defense technology

进攻类别	防御类别									攻击成功率
	D1	D2	D3	D4	D5	D6	D7	D8	D9	
A1	×	×	×	×	×	×	×	×	×	0
A2	×	√	×	×	×	×	×	×	×	0.11
A3	×	×	√	×	×	×	×	×	×	0.11
A4	×	×	√	√	×	×	×	×	×	0.22
A5	×	×	√	√	*	×	×	×	×	0.28
A6	√	√	√	√	×	√	√	×	×	0.67
A7	√	√	√	√	×	√	√	×	×	0.67
A8	√	√	√	√	×	√	√	√	×	0.78
A9	√	√	√	√	√	√	√	√	*	0.94
防御成功率	0.56	0.44	0.22	0.33	0.83	0.56	0.56	0.78	0.99	

注:1. ×表示相应攻击方法无法成功攻击相应防御方法。

2. √表示相应攻击方法能成功攻击相应防御方法。

3. * 表示相应攻击方法能够成功攻击部分防御方法。

基于现有的代码复用攻防技术,本文分析了不同攻防方法的外部指标评价情况,结果如表 4 所示。

表 4 对基于攻击指令结构特征、地址空间布局特征、运行时代码特征的 3 类攻防技术发展进行了分析。从攻击效果上看,A1 ~ A2 类基于攻击指令结构特征的攻击方法没有任何限制,实际上也说明 A1 ~ A2 类方法是基础技术。如今的机器学习能够快速收集攻击的指令特征,从而有效降低这类攻击风险。在A3 ~ A5 类基于地址空间布局特征的攻击方法中,Roglia 等[31]提出攻击方法需要额外信息,如明文过程链接表(procedure link table,PLT)、全局偏移表(global offset table,GOT)信息等,并且已出现有针对性的防御,研究比较充分;在 A6 ~ A9 类基于运行时代码特征的攻击方法中,有些研究[10-11]针对程序编程语言特点进行攻击,因此,只对部分程序有效。从防御效果来看,早期研究主要针对经典的 ROP、JOP 攻击展开,功能上有所限制。另外,防御

表 4 代码复用攻防技术外部评价指标比较

Table 4 Exterior evaluation index comparision of code reuse attack/defense technology

攻击技术编号	攻击技术依赖性	防御技术编号	防御技术依赖性	防御技术运行性能损失
A1	无[1,5-8,21]	D1	需要源码,需要修改内核,只防御 ROP 攻击[17] 需要源码[18] 需要恶意代码库[19]	5.78%[17] 3.1%[18] 0.96 倍[19]
A2	无[8,26-28]	D2	只防御 ROP 攻击[20] 需要修改硬件,只防御 JOP 攻击[23] 需要修改硬件,只防御 JOP 攻击[27] 只防御 ROP 攻击[24]	5.3 倍[20] 2%[23] 2%[27] 1.75 倍[24]

续表 4

攻击技术编号	攻击技术依赖性	防御技术编号	防御技术依赖性	防御技术运行性能损失
A3	无[30] 需要动态链接信息[31]	D3	无[29]	26%[29]
A4	无[38-39]	D4	需要重定向信息[32] 无[35-37] 需要虚拟加载环境[34]	<1%[32] 几乎无损失[33,36-37] 1.6%[35] 13%[34]
A5	无[39,44]	D5	需要修改内核内存管理功能[41] 无[42-44,48,53] 需要源码[45] 需要修改内核内存管理功能[46] 只防御 JIT-ROP 攻击[47]	2.2% ~3.4%[41] 52.1%[42] 2.7%[43] 2.1%[45] 19%[44] 1.2%[46] 14.3% ~18.6%[47] 2%[48] 6.8%[53]
A6	无[9,56] 针对 C 程序[10] 针对 C++程序[11] 针对 Linux[61]	D6	只防御 ROP 攻击[54-55,57-58] 需要源码，只防御 ROP 攻击[59] 无[51-52,60] 需要修改硬件[62] 需要先获取控制流图[50] 需要增加内核扩展组件[63]	2 倍[54] 3.5%[55] 6.65%[57] 2.07 倍[58] 7%[59] 3.279%[60] 2%[62] 16%[50] 3.6%[51] 8.54%[52] 6.1%[63]
A7	无[9] 针对 C 程序[10] 针对 C++程序[11]	D7	只防御 ROP 攻击[64-65,21-22] 需要增加内核扩展组件[63] 只防御内核级 ROP 攻击[66] 只防御 ROP 攻击[67] 只检测 ROP 攻击[68] 只防御 ROP 攻击和 JOP 攻击[69]	1.7%[64] 5.7%[65] 6%[21] 2.6%[22] 6.1%[63] 19%[66] 5.05%[67] 5%[68] 5%[69]
A8	无[17,75]	D8	无[70,72] 只防御部分 DOP 攻击[73,80]	8.4%[70] 3% ~18%[72] 19.67%[73] 20.1%[80]
A9	无[77-78]	D9	需要 Arm TrustZone[81] 需要 RISC-V 架构链接寄存器[82]	19%[81] 1.97%[82]

技术主要分为修改编译器、重写二进制代码及修改硬件和操作系统 3 类方法。在第 1 类基于修改编译器的方法中，一部分需要修改源码，但已有的不开源软件无法应用这类防御方法；另一部分不需要修改源码，通过反汇编、修改编译器进行静态插桩以增加攻击检测指令，对程序运行性能影响普遍较小，在

10%以下,只有个别研究[65]因检测点过多导致性能损耗较大(19%)。在第 2 类基于二进制代码重写的方法中,基于动态插桩的程序分析检查工作对性能影响很大,甚至达到几倍性能损失,其他基于静态代码改写的方法性能损失均在 10%以下,只有个别研究[34]因其他原因,如需要加载虚拟执行环境,导致性能损失达到 13%。在第 3 类方法中,由于采用了额外硬件或修改了操作系统核心功能,可能导致兼容性问题,但性能损耗普遍不大,在 10%以下。

综上所述,代码复用攻防技术研究热点已经从攻击指令结构特征、基于地址空间布局特征转向基于运行时代码特征的研究,并且基于运行时代码特征的研究取得了当前本领域的最佳进展,特别是基于数据流特征的攻防研究。与其他领域相比,基于数据流特征的攻防研究效果提升更加显著,目前处于顶端水平,但其发展时间还不长,说明该类研究目前还未充分展开,仍是本领域研究热点。最后,当前攻防技术在效果和性能上的限制也进一步促进了研究发展,例如:D6 ~ D9 类基于运行时代码特征防御方法经历了从高性能损耗的经典 CFI 向基于硬件辅助的粗粒度 CFI 的发展,目前的细粒度 CFI 检测性能还有进一步优化空间,DOP 防御技术也面临性能损耗过高的问题,还需要进一步加强研究。无论哪类攻击,防御都有独有的特征表示,故而针对多变的攻击,擅长特征提取的机器学习不失为一种好的选择。

3 结论

1)由于 gadget 指令相关特征是攻击代码的本质特征,基于 gadget 指令特征的检测方法当然适用于所有安全级别的应用场景,是一种常规防御手段。此类方法的检测阈值较难准确设置,不少研究采用基于大样本量的机器学习和深度学习方法提高检测准确性,但涉及攻防检测的不正常(恶意)样本数据往往存在比例失衡的问题,需要研究小样本自动生成方法。未来的代码复用攻击将继续研究消除自身的特殊指令结构特征、逃避检测的方法;未来代码复用攻击防御很可能向数据挖掘方向转化,以尽可能减少误报漏报问题。

2)由于代码复用攻击形式是内存攻击,基于内存布局随机化防御是一种通用的防御方法,适用于所有应用场景。目前,粗粒度地址随机化技术已经广泛应用于各类操作系统中,细粒度随机化研究已达到指令级别,安全性较高,但性能损失也有所提高,未作为常规防御手段使用。攻击技术在泄漏随机化内存布局,实时编译攻击方面获得了很大进展。未来代码复用攻击防御将向减少性能损失方向发展,而未来的代码复用攻击需要继续研究更巧妙的内存泄漏方式。

3)由于代码复用攻击的最终目标是实现控制流劫持,控制流劫持及保护也是一种通用的攻防方法。当前已经出现了不少自动化攻击工具,大大降低了攻击难度,而粗粒度栈保护功能已被不少操作系统默认实施,但细粒度控制流保护机制对系统性能影响较大,在如无线网络、物联网、雾计算等资源受限场景下难以应用。另外,还出现了基于非控制数据的数据导向攻击 DOP,能够规避现有的控制流保护机制,但实现难度也较大。由此可见,未来的代码复用攻击防御将向采用硬件、减少性能损失的控制流保护方向发展;未来的代码复用攻击则可能向非控制数据流攻击的方向发展。

参考文献:

[1] SHACHAM H. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86) [C]// Proceedings of 14th ACM Conference on Computer and Communications Security. New York: ACM, 2007: 552-561.

[2] 林志添. 一种不完全依赖栈的 ROP 攻击技术的研究[D]. 南京: 南京大学, 2015.
LIN Z T. Abandoning the reliance on the stack: a new ROP attack technique[D]. Nanjing: Nanjing University, 2015. (in Chinese)

[3] CHECKOWAY S, DAVI L, DMITRIENKO A, et al. Return-oriented programming without returns [C] // Proceedings of the 17th ACM Conference on Computer and Communications Security. New York: ACM, 2010: 559-572.

[4] CHEN P, XING X, MAO B, et al. Return-oriented rootkit without returns (on the x86) [C] // Information and Communications Security: 12th International Conference. Berlin: Springer, 2010: 340-354.

[5] BLETSCH T, JIANG X X, FREEH V W, et al. Jump-oriented programming: a new class of code-reuse attack [C] // Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. New York: ACM, 2011: 30-40.

[6] 邢骁, 陈平, 丁文彪, 等. BIOP: 自动构造增强型 ROP 攻击[J]. 计算机学报, 2014, 37(5): 1111-1123.
XING X, CHEN P, DING W B, et al. BIOP: automatic

construction of enhanced ROP attacks[J]. Chinese Journal of Computers, 2014, 37(5): 1111-1123. (in Chinese)

[7] CARLINI N, WAGNER D. ROP is still dangerous: breaking modern defenses[C]// Proceedings of the 23rd USENIX Security Symposium. Berkeley, CA: USENIX Association, 2014: 385-399.

[8] SADEGHI A, NIKSEFAT S, ROSTAMIPOUR M. Pure-Call Oriented Programming (PCOP): chaining the gadgets using call instructions[J]. Journal of Computer Virology and Hacking Techniques, 2018, 14: 139-156.

[9] LAN B C, LI Y, SUN H, et al. Loop-oriented programming: a new code reuse attack to bypass modern defenses[C]// 2015 IEEE Trustcom/BigDataSE/ISPA. Piscataway, NJ: IEEE, 2015: 190-197.

[10] GUO Y J, CHEN L W, SHI G. Function-oriented programming: a new class of code reuse attack in C applications [C]// 2018 IEEE Conference on Communications and Network Security. Piscataway, NJ: IEEE, 2018: 1-9.

[11] SCHUSTER F, TENDYCK T, LIEBCHEN C, et al. Counterfeit object-oriented programming: on the difficulty of preventing code reuse attacks in C++ applications [C]// 2015 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2015: 745-762.

[12] PRANDINI M, RAMILLI M. Return-oriented programming[J]. IEEE Security & Privacy, 2012, 10(6): 84-87.

[13] 金红. ROP 防御研究现状[J]. 计算机安全, 2013(5): 77-81.
JIN H. The evolution of ROP and its defense research [J]. Computer Security, 2013(5): 77-81. (in Chinese)

[14] RUAN Y F, KALYANASUNDARAM S, ZOU X K. Survey of return-oriented programming defense mechanisms[J]. Security and Communication Networks, 2016, 9(10): 1247-1265.

[15] TSOUTSOS N G, MANIATAKOS M. Anatomy of memory corruption attacks and mitigations in embedded systems[J]. IEEE Embedded Systems Letters, 2018, 10(3): 95-98.

[16] 彭国军, 梁玉, 张焕国, 等. 软件二进制代码重用技术综述[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]. Journal of Software, 2017, 28(8): 2026- 2045. (in Chinese)

[17] LI J K, WANG Z, JIANG X X, et al. Defeating return-oriented rootkits with "return-less" kernels [C]// Proceedings of the 5th European Conference on Computer Systems. New York: ACM, 2010: 195-208.

[18] ONARLIOGLU K, BILGE L, LANZI A, et al. G-Free: defeating return-oriented programming through gadget-less binaries[C]// Proceedings of the 26th Annual Computer Security Applications Conference. New York: ACM, 2010: 49-58.

[19] USUI T, IKUSE T, OTSUKI Y, et al. ROPminer: learning-based static detection of ROP chain considering linkability of ROP gadgets [J]. IEICE Transactions on Information and Systems, 2020, 103(7): 1476-1492.

[20] CHEN P, XIAO H, SHEN X B, et al. DROP: detecting return-oriented programming malicious code [C]// Information Systems Security: 5th International Conference. Berlin: Springer, 2009: 163-177.

[21] PAPPAS V, POLYCHRONAKIS M, KEROMYTIS A D. Transparent ROP exploit mitigation using indirect branch tracing [C]// Proceedings of the 22nd USENIX Conference on Security Symposium. Berkeley, CA: USENIX Association, 2013: 447-462.

[22] CHENG Y Q, ZHOU Z W, YU M, et al. ROPecker: a generic and practical approach for defending against ROP attack[C]// NDSS Sysmposium 2014: Proceedings of the 21st Network and Distributed System Security Symposium. Resten: Internet Society, 2014: 1-14.

[23] YAO F, CHEN J, VENKATARAMANI G. JOP-alarm: detecting jump-oriented programming-based anomalies in applications [C]// 2013 IEEE 31st International Conference on Computer Design. Piscataway, NJ: IEEE, 2013: 467-470.

[24] SI L, YU J, LUO L, et al. ROP-hunt: detecting return-oriented programming attacks in applications [C]// Security, Privacy and Anonymity in Computation, Communication and Storage: 9th International Conference. Berlin: Springer, 2016: 131-144.

[25] GOKTAS E, ATHANASOPOULOS E, POLYCHRONAKIS M, et al. Size does matter: why using gadget-chain length to prevent code-reuse attacks is hard [C]// Proceedings of 23rd USENIX Security Symposium. Berkeley, CA: USENIX Association, 2014: 417-432.

[26] 曹嘉欣. 基于长指令序列的 ROP 攻击方案的研究[D]. 南京: 南京大学, 2014.
CAO J X. Research on ROP exploits based on long instruction sequence [D]. Nanjing: Nanjing University, 2014. (in Chinese)

[27] KAYAALP M, SCHMITT T, NOMANI J, et al. SCRAP: architecture for signature-based protection from code reuse attacks[C]// 2013 IEEE 19th International Symposium on High Performance Computer Architecture. Piscataway,

NJ: IEEE, 2013: 258-269.

[28] SADEGHI A, AMINMANSOUR F, SHAHRIARI H R. Tiny jump-oriented programming attack (a class of code reuse attacks) [C] // 2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology. Piscataway, NJ: IEEE, 2015: 52-57.

[29] PAYER M. Too much PIE is bad for performance [R]. Zurich: ETH Zurich, 2012.

[30] SHACHAM H, PAGE M, PFAFF B, et al. On the effectiveness of address-space randomization [C] // Proceedings of the 11th ACM Conference on Computer and Communications Security. New York: ACM, 2004: 298-307.

[31] ROGLIA G F, MARTIGNONI L, PALEARI R, et al. Surgically returning to randomized lib (c) [C] // Computer Security Applications Conference. Piscataway, NJ: IEEE, 2009: 60-69.

[32] 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. Piscataway, NJ: IEEE, 2006: 339-348.

[33] PAPPAS V, POLYCHRONAKIS M, KEROMYTIS A D. Smashing the gadgets: hindering return-oriented programming using in-place code randomization [C] // 2012 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2012: 601-615.

[34] DAVIDSON J W, HALL M, CO M, et al. ILR: where'd my gadgets go? [C] //2012 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2012: 571-585.

[35] WARTELL R, MOHAN V, HAMLEN K W, et al. Binary stirring: self-randomizing instruction addresses of legacy x86 binary code [C] // Proceedings of the 2012 ACM Conference on Computer and Communications Security. New York: ACM, 2012: 157-168.

[36] GUPTA A, KERR S, KIRKPATRICK M S, et al. Marlin: a fine grained randomization approach to defend against ROP attacks [C] // Network and System Security: 7th International Conference. Berlin: Springer, 2013: 293-306.

[37] GUPTA A, HABIBI J, KIRKPATRICK M S, et al. Marlin: mitigating code reuse attacks using code randomization [J]. IEEE Transactions on Dependable and Secure Computing, 2014, 12(3): 326-337.

[38] SZEKERES L, PAYER M, WEI T, et al. SoK: eternal war in memory [C] // 2013 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2013: 48-62.

[39] 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. Piscataway, NJ: IEEE, 2013: 574-588.

[40] 袁平海, 曾庆凯, 张云剑, 等. 攻击网页浏览器: 面向脚本代码块的 ROP Gadget 注入[J]. 软件学报, 2020, 31(2): 247-265.
YUAN P H, ZENG Q K, ZHANG Y J, et al. Attack Web browser: ROP gadget injection by using JavaScript code blocks [J]. Journal of Software, 2020, 31(2): 247-265. (in Chinese)

[41] BACKES M, HOLZ T, KOLLENDA B, et al. You can run but you can't read: preventing disclosure exploits in executable code [C] // Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2014: 1342-1353.

[42] 王烨, 李清宝, 曾光裕, 等. 基于代码防泄漏的代码复用攻击防御技术[J]. 计算机研究与发展, 2016, 53(10): 2277-2287.
WANG Y, LI Q B, ZENG G Y, et al. A code reuse attack protection technique based on code anti-leakage [J]. Journal of Computer Research and Development, 2016, 53(10): 2277-2287. (in Chinese)

[43] BACKES M, NÜRNBERGER S. Oxymoron-making fine-grained memory randomization practical by allowing code sharing [C] // Proceedings of the 23rd USENIX Conference on Security Symposium. Berkeley, CA: USENIX Association, 2014: 433-447.

[44] DAVI L, LIEBCHEN C, SADEGHI A R, et al. Isomeron: code randomization resilient to (just-in-time) return-oriented programming [C] // Network and Distributed System Security Symposium. Resten: Internet Society, 2015: 1-15.

[45] BIGELOW D, HOBSON T, RUDD R, et al. Timely rerandomization for mitigating memory disclosures [C] // Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2015: 268-279.

[46] CHEN X Q, XUE R, WU C K. Timely address space rerandomization for resisting code reuse attacks [J]. Concurrency and Computation: Practice and Experience, 2017, 29(16): e3965.

[47] 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. (in Chinese)

[48] HAWKINS W, NGUYEN-TUONG A, HISER J D, et al.

Mixr: flexible runtime rerandomization for binaries[C]// Proceedings of the Workshop on Moving Target Defense. New York: ACM, 2017: 27-37.

[49] AHMED S, XIAO Y, SNOW K Z, et al. Methodologies for quantifying (re-)randomization security and timing under JIT-ROP [C] // Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2020: 1803-1820.

[50] ABADI M, BUDIU M, ERLINGSSON U, et al. Control-flow integrity [C] // Proceedings of the 12th ACM Conference on Computer and Communications Security. New York: ACM, 2005: 340-353.

[51] ZHANG C, WEI T, CHEN Z F, et al. Practical control flow integrity and randomization for binary executables [C]// 2013 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2013: 559-573.

[52] ZHANG M W, SEKAR R. Control flow integrity for COTS binaries [C] // Proceedings of 22nd USENIX Security Symposium. Berkeley, CA: USENIX Association, 2013: 337-352.

[53] HUANG X, YAN F, ZHANG L Q, et al. Honeygadget: a deception based approach for detecting code reuse attacks [J]. Information Systems Frontiers, 2021, 23(2): 269-283.

[54] DAVI L, SADEGHI A R, WINANDY M. ROPdefender: a detection tool to defend against return-oriented programming attacks[C]// Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. New York: ACM, 2011: 40-51.

[55] DANG T H Y, MANIATIS P, WAGNER D. The performance cost of shadow stacks and stack canaries[C]// Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. New York: ACM, 2015: 555-566.

[56] 黄韬. 一种绕过平行影子栈的 ROP 攻击方法的设计与实现[D]. 南京: 南京大学. 2016.
HUANG T. The design and implementation of a ROP exploit schema bypassing parallel shadow stack [D]. Nanjing: Nanjing University, 2016. (in Chinese)

[57] ZOU C W, WANG X D, GAO Y Q, et al. Buddy stacks: protecting return addresses with efficient thread-local storage and runtime re-randomization [J]. ACM Transactions on Software Engineering and Methodology, 2022, 32(2): 35e.

[58] 陈林博, 江建慧, 张丹青. 利用返回地址保护机制防御代码复用类攻击[J]. 计算机科学, 2013, 40(9): 93-98, 102.
CHEN L B, JIANG J H, ZHANG D Q. Prevention of code reuse attacks through return address protection[J]. Computer Science, 2013, 40(9): 93-98, 102. (in Chinese)

[59] COWAN C, PU C, MAIER D, et al. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks[C]// Proceedings of the 7th Conference on USENIX Security Symposium. Berkeley, CA: USENIX Association, 1998: 63-78.

[60] 朱君. 基于 Canary 的增强型栈保护技术研究[D]. 南京: 南京大学, 2017.
ZHU J. Research on the technology of enhanced canary-based protections [D]. Nanjing: Nanjing University, 2017. (in Chinese)

[61] BITTAU A, BELAY A, MASHTIZADEH A, et al. Hacking blind[C]// 2014 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2014: 227-242.

[62] KAYAALP M, OZSOY M, ABU-GHAZALEH N, et al. Branch regulation: low-overhead protection from code reuse attacks[C]// 39th Annual International Symposium on Computer Architecture. Piscataway, NJ: IEEE, 2012: 94-105.

[63] XIA Y B, LIU Y T, CHEN H B, et al. CFIMon: detecting violation of control flow integrity using performance counters [C] // IEEE/IFIP International Conference on Dependable Systems and Networks. Piscataway, NJ: IEEE, 2012: 1-12.

[64] KIM J, KIM I, MIN C, et al. Zero-sum defender: fast and space-efficient defense against return-oriented programming attacks [J]. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2014, 97(1): 303-305.

[65] 李威威, 马越, 王俊杰, 等. 基于硬件分支信息的 ROP 攻击检测方法[J]. 软件学报, 2020, 31(11): 3588-3602.
LI W W, MA Y, WANG J J, et al. ROP attack detection approach based on hardware branch information [J]. Journal of Software, 2020, 31 (11): 3588-3602. (in Chinese)

[66] ZHOU H W, WU X, SHI W C, et al. HDROP: detecting ROP attacks using performance monitoring counters [C] // Information Security Practice and Experience: 10th International Conference. Berlin: Springer, 2014: 172-186.

[67] 牛伟纳, 赵成洋, 张小松, 等. ROPDetector: 一种基于硬件性能计数器的 ROP 攻击实时检测方法[J]. 计算机学报, 2021, 44(4): 761-772.
NIU W N, ZHAO C Y, ZHANG X S, et al.

ROPDetector: a real-time detection method of ROP attack based on hardware performance counter [J]. Chinese Journal of Computers, 2021, 44(4): 761-772. (in Chinese)

[68] PFAFF D, HACK S, HAMMER C. Learning how to prevent return-oriented programming efficiently [C]// Engineering Security Software and Systems: 7th International Symposium. Berlin: Springer, 2015: 68-85.

[69] 严飞, 彭慧容, 何凡, 等. HBROP: 基于硬件性能计数器的函数级 ROP 检测[J]. 武汉大学学报(理学版), 2017, 63(2): 109-116.
YAN F, PENG H R, HE F, et al. HBROP: HPC-based function-level approach to detect ROP attack [J]. Journal of Wuhan University (Natural Science Edition), 2017, 63(2): 109-116. (in Chinese)

[70] KUZNETZOV V, SZEKERES L, PAYER M, et al. Code-pointer integrity [C]// Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 147-163.

[71] EVANS I, FINGERET S, GONZALEZ J, et al. Missing the point (er): on the effectiveness of code pointer integrity[C]// 2015 IEEE Symposium on Security and Privacy(SP). Piscataway, NJ: IEEE, 2015: 781-796.

[72] MASHTIZADEH A J, BITTAU A, BONEH D, et al. CCFI: cryptographically enforced control flow integrity [C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2015: 941-951.

[73] WANG Y, LI Q B, CHEN Z F, et al. BCI-CFI: a context-sensitive control-flow integrity method based on branch correlation integrity[J]. Information and Software Technology, 2021, 136: 106572.

[74] RAMALINGAM G. The undecidability of aliasing[J]. ACM Transactions on Programming Languages and Systems, 1994, 16(5): 1467-1471

[75] EVANS I, LONG F, OTGONBAATAR U, et al. Control jujutsu: on the weaknesses of fine-grained control flow integrity[C]// Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2015: 901-913.

[76] LI Y, WANG M Z, ZHANG C, et al. Finding cracks in shields: on the security of control flow integrity mechanisms[C]//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2020: 1821-1835.

[77] CHEN S, XU J, SEZER E C, et al. Non-control-data attacks are realistic threats[C]//Proceedings of the 14th Conference on USENIX Security Symposium. Berkeley, CA: USENIX Association, 2005: 177-191.

[78] HU H, SHINDE S, ADRIAN S, et al. Data-oriented programming: on the expressiveness of non-control data attacks[C]// 2016 IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE, 2016: 969-986.

[79] CHENG L, AHMED S, LILJESTRAND H, et al. Exploitation techniques for data-oriented attacks with existing and potential defense approaches [J]. ACM Transactions on Privacy and Security, 2021, 24(4): 26.

[80] WANG Y, LI Q B, CHEN Z F, et al. Shapeshifter: intelligence-driven data plane randomization resilient to data-oriented programming attacks [J]. Computers & Security, 2020, 89: 101679.

[81] ABERA T, ASOKAN N, DAVI L, et al. C-FLAT: control-flow attestation for embedded systems software[C]// Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2016: 743-754.

[82] ZEITOUNI S, DESSOUKY G, ARIAS O, et al. ATRIUM: runtime attestation resilient under memory attacks[C]// 2017 IEEE/ACM International Conference on Computer-Aided Design. New York: ACM, 2017: 384-391.

[83] ZHAN J, LI Y Z, LIU Y F, et al. NSGA-Ⅱ-based granularity-adaptive control-flow attestation[J]. Security and Communication Networks, 2021, 2021: 1-16.

(责任编辑 梁 洁)