

基于软硬件结合的控制流完整性保护技术

梁浩, 何本伟, 王庆丰, 程国振, 马海龙
(信息工程大学, 河南 郑州 450001)

摘要: 针对影子栈难以兼顾安全性和性能, 软件实现的前向CFI (control-flow integrity) 技术性能开销过大、难以进行实际部署, 提出基于软硬件结合的CFI (control-flow integrity based on the combination of software and hardware, SHCFI) 技术。通过二进制重写器对程序反编译, 生成中间语言, 并判断转移指令类型。对于后向控制流的保护, 提出基于返回地址加密的平行影子栈方案, 使用随机数对栈上的返回地址进行异或加密, 将加密结果备份到原始栈固定偏移处的影子栈中, 在函数返回时对影子栈中的地址异或解密, 将解密结果作为实际的返回地址。对于前向控制流的保护, 使用硬件ENDBRANCH状态机指令标记间接转移指令的目标地址, 在运行时对目标地址进行合法性检查, 以降低性能开销。实验结果表明, SHCFI加固后的程序能够有效地防御代码重用攻击, 且具有良好的运行开销。

关键词: 软件多样化; 控制流完整性; 软硬件结合; 代码重用攻击

中图分类号: TP311.5

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2024070

Control-flow integrity technology based on the combination of software and hardware

LIANG Hao, HE Benwei, WANG Qingfeng, CHENG Guozhen, MA Hailong
Information Engineering University, Zhengzhou 450001, China

Abstract: For shadow stack, balancing security and performance was difficult, and the forward CFI (control-flow integrity) scheme implemented by software was too expensive and difficult to be deployed in practice. A hybrid CFI scheme based on software and hardware integration, named SHCFI (control-flow integrity based on the combination of software and hardware), was proposed. Decompilation was undergone by the program using a binary rewriter to generate an intermediate representation, and various transfer instruction types were identified. For the protection of backward control flow, a novel parallel shadow stack scheme was introduced, featuring encrypted return addresses. This involved XOR encryption of return addresses on the stack using random numbers, with the encrypted results stored securely in a shadow stack located at a fixed offset on the original stack. During function re-

收稿日期: 2024-02-19; 修回日期: 2024-04-28

通信作者: 王庆丰, wang_qf2024@163.com

基金项目: 国家自然科学基金 (62002383)

Foundation Item: The National Natural Science Foundation of China (62002383)

引用格式: 梁浩, 何本伟, 王庆丰, 等. 基于软硬件结合的控制流完整性保护技术[J]. 网络与信息安全学报, 2024, 10(5): 107-118.

Citation Format: LIANG H, HE B W, WANG Q F, et al. Control-flow integrity technology based on the combination of software and hardware[J]. Chinese Journal of Network and Information Security, 2024, 10(5): 107-118.

turns, addresses in the shadow stack underwent XOR decryption, and the decrypted results served as the actual return addresses. For forward control flow protection, the hardware ENDBRANCH state machine instruction was utilized to mark the target addresses of indirect transfer instructions. At runtime, legitimacy checks were applied to these target addresses, thereby reducing performance overhead. Experimental results demonstrated that programs fortified with SHCFI effectively defended against code reuse attacks, exhibiting favorable runtime overhead.

Keywords: software diversity, control-flow integrity, combination of software and hardware, code-reuse attack

0 引言

软件多样化对具有内存漏洞的程序提供概率保护,其目的是大幅度增加攻击者的攻击成本,使攻击者难以实现攻击或实施大规模攻击。然而攻击者针对单个软件进行内存布局暴力破解时,仍有一定概率绕过软件多样化技术的保护。为了提高单个软件防御代码重用攻击的能力,Abadi等^[1]提出控制流完整性(control flow integrity, CFI)技术,要求程序在执行过程中,每当控制流转移指令执行时,都以一个有效的白名单集合为目标,该目标集合由提前创建的控制流图(control flow graph, CFG)确定,以此提供针对代码重用攻击的通用防御。然而,在对二进制文件进行CFI技术安全加固时,难以绘制精确的控制流图,且运行时需要频繁地对目标地址合法性进行检查,导致CFI技术引入了较高的执行开销,难以对软件进行实际部署。

为了进一步降低CFI技术的性能开销,研究人员从纯软件实现的CFI技术逐步过渡到硬件支持的CFI实现方案^[1-7],如最终分支记录和分支追踪存储等硬件可以记录程序控制流的执行,从而检测程序是否受到了代码重用攻击。Intel提出CET(control-flow enforcement technology)方案^[8],旨在硬件层面对后向和前向控制流劫持做到全面防御。对于后向控制流,CET技术通过硬件维护一个与程序栈类似的影子栈来记录函数调用和返回地址,在执行ret指令时,处理器会检查从影子栈中取的返回地址是否与真实栈中的对应地址相同,以此防止攻击者利用栈溢出等漏洞修改返回地址。对于前向控制流,通过ENDBRANCH状态机启用IBT(indirect branch tracking)校验机制,追踪每个间接转移指令的跳转目标地址并进行合法性验证。虽然这项技术前景广阔,但尚未公布发布日期,因此尚不清楚

CET技术应对代码重用攻击的真实防御能力。基于软件实现的传统影子栈^[9-11]使用线程局部存储(thread local storage, TLS)或专用寄存器等方式隐藏影子栈的栈顶指针,使攻击者无法访问影子栈的内容,影子栈中的返回地址在内存中连续被存储,影子栈指针将在每个函数中被解引用两次,一次在函数序言处压入正确的返回地址,一次在函数结语处弹出影子栈中的返回地址,影子栈频繁的解引用带来了较高的性能损耗。而平行影子栈^[12-13]将影子堆栈项放在与程序堆栈的恒定偏移处,不需要额外的寄存器或内存访问,也不需要维护影子栈指针的移位,具有良好的性能开销,然而攻击者通过伪造栈方式^[14]或信息泄露获取影子栈的偏移量^[15],即可成功发起代码重用攻击。

针对影子栈难以兼顾安全性和性能,软件实现的前向CFI技术性能开销过大、难以进行实际部署,本文提出基于软硬件结合的CFI技术——SHCFI(control-flow integrity based on the combination of software and hardware)技术,主要贡献如下。

1) 提出面向二进制代码的控制流完整性保护实现架构,分别通过返回地址加密的平行影子栈方案和硬件ENDBRANCH状态机指令实现对前向、后向控制流的保护。

2) 从性能、安全、文件膨胀率等方面对所提方法进行了全面的测试验证,实验结果表明,SHCFI技术加固后的程序能够有效地防御代码重用攻击,具有良好的运行开销。

1 相关进展

Abadi等^[1]首次提出控制流完整性技术,其核心思想是在程序运行时对间接转移指令进行动态验证,限制程序运行中的控制流转移,使其始终处于程序控制流图所限定的白名单范围内,旨在解决缓冲区溢出等内存漏洞利用攻击中的控制

流劫持问题。该技术主要分为两步：首先分析程序的源代码或二进制代码获取控制流图，得到间接转移指令的目的地址；然后在程序运行时，验证间接转移指令的目的地址是否和控制流图一致。代码重用攻击如 ROP (return-oriented programming)、JOP (jump-oriented programming) 等会劫持程序的正常控制流进行非法转移，导致程序执行不会遵循原有的 CFG，因此 CFI 技术能够有效地防御此类攻击。

控制流完整性防御示例如图 1 所示，图中节点代表程序的基本块，箭头表示执行流的方向。CFI 技术在每个 CFG 节点跳转到其他节点时进行目的地址检查，如节点 E 的跳转目标地址白名单为节点 D 和节点 F，攻击者一旦将控制流重定向到节点 A 或节点 G，CFI 技术发现控制流转移异常就会立即终止程序执行。可以看出，CFI 技术在运行时基于预定义的 CFG 验证程序控制流的完整性，提供了一种通用的代码重用攻击的防御方法。

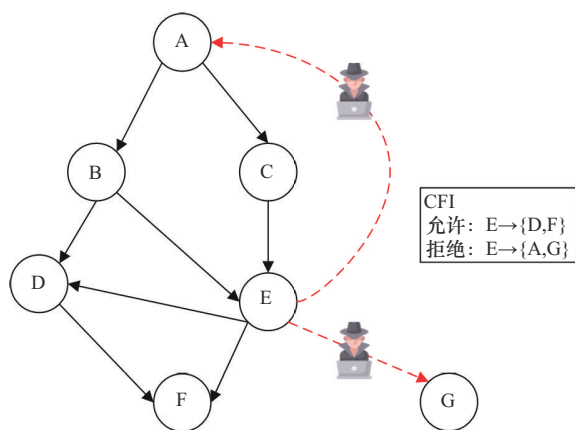


图 1 控制流完整性防御示例

Figure 1 Examples of control flow integrity defenses

基于源码的 CFI 技术可通过源代码获得详细的间接控制流转移信息，绘制精确的控制流图，具有较高的安全性。文献[16]首次将 CFI 技术集成到 LLVM (low level virtual machine) 和 GCC (GNU compiler collection) 两种主流编译器中，实现了对前向控制流间接调用的 CFI 保护方案，在 GCC 编译器中，设计虚函数表进行验证，通过构造正确的虚函数表指针集合，在每个调用点调用之前验证虚函数表指针是否被篡改。在 LLVM 编译器中，设计间接调用检查，为间接调

用目标生成跳转表并在间接调用点前插桩代码来转换函数指针，以此保护间接调用，确保它们指向相应的跳转表条目。文献[17]提出 π CFI，可以强制提取每个具体输入计算的 CFG，相比静态分析得到的全局 CFG，特定输入生成的 CFG 意味着更少的间接分支数量，降低了性能开销。为了防止攻击者任意添加边， π CFI 使用静态计算所有输入的 CFG 来约束运行时可以添加的边。文献[18]提出模块化控制流完整性 (modular control-flow integrity, MCFI)，该方法将应用程序划分为多个模块，每个模块包含相应的代码、数据和辅助信息，帮助该模块与其他模块的链接以及生成该模块的 CFG，模块的代码单独检测控制流的完整性。文献[19]提出操作系统内核控制流完整性 (kernel control-flow integrity, KCoFI) 的保护方案，使用传统的标签匹配方法和影子栈监控间接 jmp/call 指令和 ret 指令的传输目标，同时对上下文切换、中断处理和信号处理程序分派中的内核控制数据进行保护，然而如此精细的保护导致 KCoFI 产生超过 100% 的性能开销。为了降低影子栈的性能开销，文献[12]提出平行影子栈，将影子栈放置在原始调用栈的固定偏移处，使用 pop 指令和 push 指令就可以完成对返回地址的备份和读取，然而攻击者利用栈切换 gadget，如 “xchg %rax, %rsp; ret” 进行栈迁移，同时修改平行影子栈和调用栈上的数据，能够绕过平行影子栈的防御。

由于程序源代码并不容易获得，尤其是众多商业软件并不会公开源代码，基于此，研究人员越来越关注直接对二进制程序进行安全加固的 CFI 技术。文献[20]使用 Super-CFG 算法构造互斥的代码块，该算法将程序的控制流转移分为块间转移和块内转移两种类型，块间转移只允许转移到其他代码块中特定类型的目标，而块内转移只允许控制流转移到当前块内，该方法能够显著减少控制流转移合法目标数量。文献[21]提出一种针对 COTS (commercial off the shelf) 软件的 BinCFI 保护方案，通过二进制重写间接转移指令，将重写指令写入新增代码段中，分别建立指令 ret 和间接指令 jmp/call 的地址转换表，只有当目标地址在地址表中时，才能进行实际的控制传

输。然而 BinCFI 保护方案在启动程序之前需要大量时间进行预处理, 导致该方法与时序关键程序不兼容。为了减少预处理时间, 快速启动程序, 文献[22]没有在线阶段构建控制流图, 仅对只读数据和少量程序代码进行轻量级分析, 并提出代码启发式方法直接从二进制文件中提取控制流元数据, 对前向间接转移的保护策略不允许间接指令 `jmp` 和 `call` 跳到基本块的中间, 对后向间接转移维护一个影子栈以提供上下文敏感的防御。

为了进一步降低 CFI 技术的性能开销, 研究人员从以上纯软件实现的 CFI 技术逐步过渡到硬件支持的 CFI 实现方案, 通过 Intel 处理器的最终分支记录 (last branch recording, LBR) 和分支跟踪存储 (branch tracing store, BTS) 等硬件记录程序控制流的执行记录^[23], 对执行记录进行检查可以确认是否受到了代码重用攻击。文献[24]首次使用 LBR 的分支跟踪功能演示了 CFI 技术, 并在 Windows 7 上实现了 kBouncer。kBouncer 的关键思想是通过 LBR 来检查危险系统调用的控制流路径, 监视已执行的间接分支, 使用启发式方法来区分由 ROP 攻击链引起的执行痕迹和合法的执行痕迹。文献[3]提出 CFIMon 方案, 使用分支跟踪存储 BTS 来检测控制流偏差, BTS 动态收集和分析运行时跟踪的信息, 然后将这些信息存入缓冲区。当系统进行敏感函数调用或缓冲区将要装满时, 触发 CFIMon 指令检查机制以检测历史控制流转移指令是否存在犯规。文献[4]基于 LLVM 编译器实现了上下文敏感的保护方案 CCFI (context-sensitive CFI), 使用消息验证码 (message authentication code, MAC) 对返回地址、函数指针、方法指针和虚表指针 4 类敏感数据进行加密保护。这些数据每次从内存中加载时进行比对检查, 而用于计算这些数据对应 MAC 的随机密钥存储在专用寄存器中, 降低了攻击者通过内存泄露获得密钥的风险。CCFI 利用处理器的加密指令 AES.NI 加快 MAC 的计算速度, 不过平均性能开销仍超过 50%。Intel 对底层处理器架构进行了更改, 推出基于硬件的安全机制 CET^[8], 它由影子栈和间接分支跟踪 (indirect branch tracking, IBT) 两部分组成。对于后向控制流, 影子栈在 CPU 的安全区域存储返回地址

的副本保护 `ret` 指令。对于前向控制流, 在执行间接转移指令时通过 ENDBRANCH 状态机启用 IBT 校验机制, 追踪每个间接转移指令的跳转目标地址从而进行验证。

综上所述, CFI 技术需要对程序每条间接转移的目的地址进行校验, 确保每条转移指令只能转移到控制流图的白名单中。对后向控制流的保护一般采用影子栈方式, 通过检查调用栈上的返回地址与阴影栈上的副本是否匹配以确保执行返回地址的完整性。传统影子栈需要在内存和寄存器之间多次传递数据, 导致性能有较大损耗, 且难以处理 `setjmp/longjmp` 展开多个栈帧的问题, 兼容性较差。平行影子栈具有合理的性能开销, 然而安全取决于其位置的保密性, 攻击者通过暴力破解或者信息泄露推测影子栈在内存空间的位置, 同时覆盖影子栈和原调用栈的返回地址, 可以绕过平行影子栈的防护。而对于前向控制流的保护往往在程序中插入额外的代码进行标签声明和校验, 其安全性取决于生成控制流图的精确性, 且纯软件实现的前向保护方案需要插入较多的保护代码, 并频繁地进行目标地址合法性的验证, 具有较高的性能开销 (在 20% 至 50%)^[1], 难以进行实际部署。目前研究人员致力于研究基于硬件的 CFI 技术, 以最大程度地提高程序运行效率, 在尽量不降低安全性的前提下增强 CFI 技术的适用性和可部署性。

2 SHCFI 技术

SHCFI 技术旨在预防后向控制流转移指令 `ret` 和前向控制流转移指令, 对间接 `jmp/call` 指令提供保护, 以防被攻击者劫持而发起 ROP/JOP 等代码重用攻击。二进制程序控制流保护流程如图 2 所示。

SHCFI 技术首先通过重写器 Egalito 将源程序反编译为中间语言 EIR (egalito intermediate representation), 然后判断间接转移指令类型, 可划分为后向 `ret` 指令和前向间接 `jmp/call` 指令。`ret` 指令用于将程序的控制流返回到先前的位置, 间接 `jmp/call` 指令用于将程序的控制转移到新的位置。对 `ret` 指令的保护通过返回地址加密的平行影子栈方案, 首先将函数返回地址与一个随机数异或加密, 加密后的值备份到平行影子栈中, 当函数

调用结束执行 `ret` 指令返回时，再解密平行影子栈中的返回地址并将解密结果放回到原始栈上，确保返回地址的完整性。对间接 `jmp/call` 指令的保护采用硬件 `ENDBRANCH` 状态机的指令实现保护，依据 Egalito 生成的控制流图，在间接 `jmp/call` 指令的合法转移目标地址处标记 `ENDBRANCH` 状态机识别标签，在程序执行时，验证间接 `jmp/call` 指令的目标地址合法性，防止间接 `jmp/call` 指令跳转到其他非法位置。

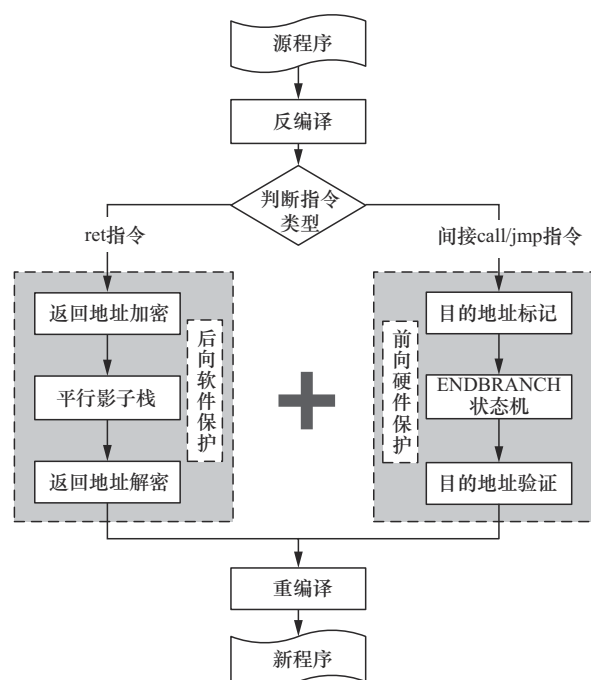


图2 二进制程序控制流完整性保护流程

Figure 2 Control-flow integrity process for binary program

3 SHCFI 实现

3.1 后向控制流保护

对后向控制流的保护主要防止函数返回指令 `ret` 被攻击者篡改。当程序执行到函数调用指令 `call` 时，会把 `call` 指令的下一条指令地址作为本次函数调用的返回地址并进行压栈操作。当被调用函数执行返回指令 `ret` 时，系统会从栈中弹出返回地址给程序计数器，程序继续正常执行。而由于 C/C++ 应用程序栈空间是可写的，攻击者往往通过缓冲区溢出的方式修改返回地址，然后重定向控制流至构建攻击负载以发起代码重用攻击。因此，对后向控制流的保护，主要是保护栈上返回地址的完整性。

CFI 技术对后向控制流的保护一般是借助影子栈完成，其基本思想是为每个函数维护一个影子栈，与程序运行时调用栈的作用类似，但影子栈只保存函数返回地址，并且是被隐藏起来的，攻击者不知道其具体位置，无法做出修改。当函数 A 跳转到调用函数 B 时，会将函数 A 的返回地址压入原始调用栈和影子栈中，而函数 B 执行完后，有两种方式确保返回地址的完整性：一是验证方式，将对应的返回地址从影子栈中弹出并与实际的返回地址比较，不仅能保护返回地址，还可以检查栈上的局部变量是否被修改；二是覆写方式，不经过比较，直接将影子栈中的返回地址压入栈中，降低了性能损耗。由于篡改栈上局部变量的操作一般只发生在前向控制流攻击行为中，本节仅对返回地址进行保护。为了增加返回地址验证带来的性能开销，本文选择覆写方式保护返回地址。

SHCFI 技术采用返回地址加密的平行影子栈的方式，通过对返回地址加密并将加密结果存储在原始栈的固定偏移处，如偏移量为 `0x400 000`，该位置为重写器在程序运行时预留的空白区域，仅通过压栈和入栈操作就可以完成对返回地址的备份。为了防止攻击者通过信息泄露或暴力破解方式获取影子栈的偏移量，同时修改原始栈和影子栈中的返回地址实施攻击，SHCFI 技术还对 `ret` 地址进行异或加密。平行影子栈示意如图 3 所示，当程序指令的译指结果是 `call` 指令时，SHCFI 会从线程局部存储偏移量 `0x28` 字节处取一随机值 `%fs: 0x 28`，即 `canary`。由于 `canary` 存储在每个线程独立拥有的内存空间中，只有程序本身才知道 `canary` 值的具体地址，攻击者无法直接通过读取内存的方式获取 `canary` 值。将 `canary` 与栈上的返回地址异或，并将异或结果压进原始栈中，然后通过 `pop` 指令对加密后的返回地址备份到偏移量为 `0x400 000` 的影子栈中。当译指结果是 `ret` 指令时，首先从影子栈中取出加密后的返回地址再次和 `canary` 进行异或，解密出真正的返回地址，并将解密结果压进原始栈中作为真正的返回地址，这样程序就可以正常返回到调用之前的状态。

图 4 为 SHCFI 的后向控制流保护汇编示意，在函数序言处，首先将随机值 `%fs: 0x28` 存放至寄

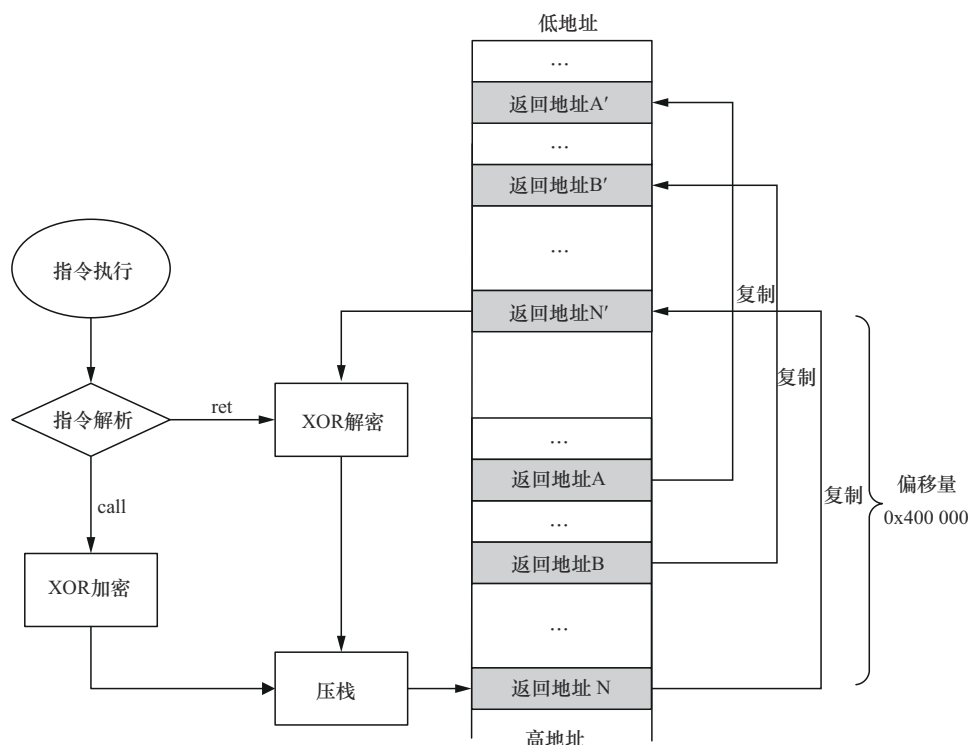


图3 平行影子栈示意

Figure 3 Schematic diagram of the parallel shadow stack

寄存器r11中，r11被称为易失性寄存器，用于保存不需要在各个调用之间保留的临时数据；然后将寄存器r11的值与返回地址进行XOR加密。需要注意的是，此时(%rsp)里面存储的是返回地址，然后将加密后的地址存储到原始栈中。通过指令“pop 0x400 000 (%rsp)”将加密后的返回地址备份到影子栈中。为了不改变原始栈的布局，将栈指针rsp减小8 byte。在函数结语处，再一次利用canary和备份的加密返回地址异或，将解密后的地址压进栈中，作为ret指令返回的真正地址。

SHCFI技术通过返回地址加密方式对返回地址提供了更安全的保护，具体而言，SHCFI技术对后向控制流的保护有以下优点。一是不需要修改寄存器，传统影子栈在内存和寄存器之间多次传递数据，通过mov指令和lea指令完成返回地址的备份，引入了较高的性能开销。SHCFI技术通过pop和push指令完成对返回地址的操作，虽然需要两倍的栈内存开销，但不需要保存或恢复任何寄存器，降低了性能损耗。二是对setjmp和longjmp等异常控制流的处理，不同于call和ret一对一的常规匹配，setjmp和longjmp函数可以

```
//函数序言
55                push  %rbp
48 89 e5           mov   %rsp, %rbp

.....

//函数结语
5d                pop   %rbp
e3                ret
```

(a) 程序原始汇编代码

```
//函数序言
64 4c 8b 1c 25 28 00 00 00  mov  %fs:0x28, %r11
4c 31 1c 24           xor   %r11, (%rsp)
8f 84 24 00 00 40 00    pop   0x400000(%rsp)
48 2b 24 25 08 00 00 00  sub   0x8, %rsp
55                push  %rbp
48 89 e5           mov   %rsp, %rbp

.....

//函数结语
5d                pop   %rbp
64 4c 8b 1c 25 28 00 00 00  mov  %fs:0x28, %r11
4c 31 9c 24 00 00 40 00    xor   %r11, 0x400000(%rsp)
ff b4 24 00 00 40 00    push  0x400000(%rsp)
e3                ret
```

(b) SHCFI安全保护后汇编代码

图4 SHCFI的后向控制流保护汇编示意

Figure 4 The schematic of the assembly for backward control flow protection of SHCFI

展开多个堆栈帧，传统影子栈需要从影子栈中不断弹出返回地址，直到找到匹配项，增加了程序的运行时间。所提方法使用平行影子栈，影子栈位置始终在栈指针固定偏移处，会自动处理栈指

针的异常更改。三是能够防御信息泄露攻击，攻击者往往会通过信息泄露或暴力破解的方式计算影子栈的偏移量，同时修改原始栈和影子栈的内容，此时攻击者就可以进行正常的ROP攻击。然而栈中的地址已经被加密处理，如果程序被调用函数进行返回操作，影子栈中返回地址已经被篡改攻击为攻击负载地址，该地址与经过异或解密后的结果将导致程序控制流执行到一个未定义的虚拟存储区域中，此时程序会崩溃。

3.2 前向控制流保护

前向控制流主要保护的是间接转移指令，有间接call和间接jmp两种。间接转移指令使用数据寻址方式间接获得转移地址，如执行完“callq *%rax”这条指令之后，指令寄存器的值就被替换为rax寄存器的值，跳转目标的地址是程序动态运行时从寄存器读取的^[24]，这些目标地址是可变化的，因此攻击者可以篡改寄存器的值或内存中的数据，实施JOP等类型的代码重用攻击。

SHCFI技术对二进制程序实现了类似于CET技术对前向控制流的保护，以解决CET技术不能对二进制代码进行保护的问题。SHCFI技术借助ENDBRANCH状态机指令对间接jmp/call的目的地址进行监控，CPU在用户态和内核态分别设有一个ENDBRANCH状态机（如图5所示），状态机共有两个状态，分别为IDLE和WAIT_FOR_ENDBRANCH。状态机最开始处于IDLE状态，当程序执行间接转移指令jmp/call时，状态机由初始状态IDLE状态转变为WAIT_FOR_ENDBRANCH状态。如果一条指令不是endbr32/64，会触发#CP异常。只有在下一条指令是endbr32/64的情况下，状态机回到IDLE状态，程序才能正常运行。另外，在不支持ENDBRANCH状态机的处理器上，endbr32/64被视为NOP（no operation）指令，不会影响程序的执行状态，这允许该方法转换的二进制程序可以在不支持ENDBRANCH状态机的处理器上执行，具有较好的兼容性。

前向控制流汇编代码变化如图6所示。当程序执行指令“callq *%rax”间接调用函数时，程序将寄存器rax的值与0xfa1e0ff3进行比较（0xfa1e0ff3表示的是状态机指令endbr64，标记

在函数入口处），如果比较结果一致，程序正常执行，否则程序跳转到控制流报错函数control-flow error，触发程序崩溃。SHCFI技术确保间接调用和跳转只能重定向到以endbr64指令开头的函数，这并不能确保调用正确的函数，如果攻击者更改控制流以跳转到另一个同样开头的函数endbr64，状态机将不会发现错误并继续执行程序。相较SHCFI技术对后向控制流的保护，对前向控制流仅采取粗粒度的保护，将所有间接跳转目标视为一个集合，这仍然大大缩小了攻击面，且大多数JOP/ROP攻击针对的是函数体内的代码片段，很难从函数序言处截取代码片段进行攻击。

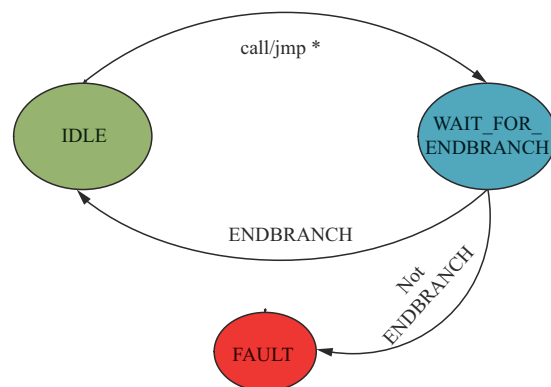


图5 ENDBRANCH状态机

Figure 5 ENDBRANCH state machine

//原汇编		
41 ff d3	...	callq *%rax
...	...	
55		push %rbp
48 89 e5		mov %rsp,%rbp
...	...	

(a) 程序原始汇编代码

//新汇编指令		
49 89 d3	...	mov %rax,%r11
41 81 3b f3 0f 1e fa		cmpl \$0xfa1e0ff3,(%r11)
0f 85 60 fe ff ff		jne 40000000 <control-flow error>
41 ff d3		callq *%r11
...	...	
f3 0f 1e fa		endbr64
55		push %rbp
48 89 e5		mov %rsp,%rbp
...	...	
<control-flow error>:		ud2
0f 0b		
...	...	

(b) 程序新汇编代码

图6 前向控制流汇编代码变化示意

Figure 6 Schematic of forward control flow assembly code changes

4 实验结果与分析

实验从转换前后程序的性能、实际防御效果以及文件膨胀率等方面对 SHCFI 技术进行了测试, 实验使用 SPEC CPU2006^[25]基准测试集中的 C/C++ 程序进行多样化转换, 源程序生成用的是 O2 级别优化, 实验设置如表 1 所示。

表 1 实验设置

Table 1 Experimental settings

软/硬件	参数
操作系统	64 位 Ubuntu 18.04
处理器	Inter Core i7-8700
内核版本	Linux 5.4.0-42-generic
内存	8 GB

4.1 性能测试

在性能测试中分别分析 SHCFI 技术的前向和后向保护机制和总的性能开销, 性能测试结果如图 7 所示。

图 7 柱状图表示 SHCFI 的后向、前向和总的性能开销, 折线图表示间接转移指令的执行次数。性能开销主要受间接转移指令的执行次数影响, 无论是前向转移还是后向转移指令, 执行次数越多, 程序中检查指令执行次数就越

多, 从而导致了较高的性能损耗。如程序 445.gobmk, 程序中有大量的 ret 指令执行, 导致后向控制流保护的性能开销增加 7.79%, 相对程序 470.lbm, ret 指令执行次数较少, 相应的性能开销较低, 仅有 0.26%。而程序 483.xalacbm 存在大量的虚拟函数调用情况, 导致前向控制流保护性能开销增加 9.41%。总体来说, SHCFI 技术的后向控制流保护平均性能开销增加 2.69%, 前向控制流保护开销增加 0.82%, 而总的性能开销增加 3.9%, 相较传统影子栈 9.7% 的运行开销^[11], SHCFI 具有较好的性能开销。

4.2 安全测试

本节对 SHCFI 技术的安全测试分为两个部分, 后向保护测试和前向保护测试。后向保护测试通过实例 ROP 攻击分析 SHCFI 技术保护程序的安全性, 前向保护测试采用定量分析评估对间接分支指令的利用难度。

4.2.1 后向保护测试

本节主要测试 SHCFI 技术对后向控制流攻击实际防御效果, 漏洞程序部分源代码如图 8 所示。使用含有缓冲区溢出的漏洞程序进行测试, 触发溢出漏洞的代码分别位于第 6 行、第 11 行。为了确保实验的真实性, 漏洞程序编译时开启了 ASLR (address space layout randomization) 和

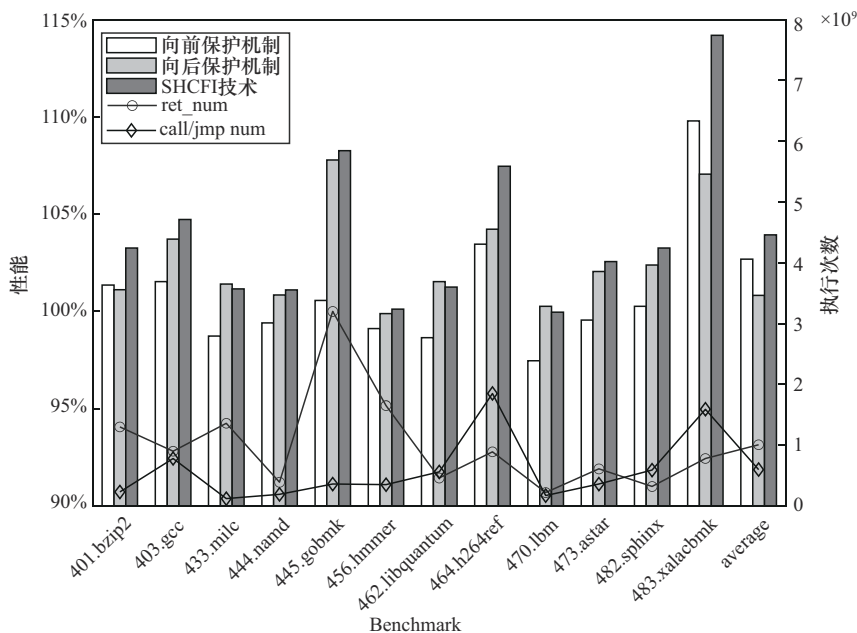


图 7 性能测试结果

Figure 7 Performance test results

NX (non-executable) 保护, 并去掉栈金丝雀保护^[26], 避免影响 SHCFI 技术对后向控制流的保护效果。由于开启了 ASLR 保护, 攻击者无法获得库函数 `system` 的具体地址。若要通过 ROP 攻击获得系统权限, 需要攻击者利用源程序中 `write@plt()` 函数将 `write` 函数在内存中的地址 `write.got` 打印出来, ASLR 技术并不改变函数 `write` 和 `system` 的相对偏移量, 借助 `libc.so` 基地址就可以计算出 `system` 和所需参数 `“/bin/sh”` 在内存中的实际地址, 然后构建 ROP 攻击负载实施攻击。

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 void vulnerable() {
6     char buf[128];
7     read(STDIN_FILENO, buf, 256);
8 }
9
10 int main(int argc, char** argv) {
11     vulnerable();
12     write(STDOUT_FILENO, "vulnerable is here\n", 19);
13 }

```

图8 漏洞程序部分源代码

Figure 8 Vulnerability program part source code

获得 `write` 函数运行时的地址所构建的 `gadgets` 攻击链在栈上的位置, ROP 攻击防御示意如图9所示。在内存中搜索到 `gadget1 “pop %rdi; ret”`、`gadget2 “pop %rsi; pop %r15; ret”` 和 `gadget3 “pop %rdx; ret”`, 将 `write` 函数所需参数 `0x1`、`got_write()` 地址和 `0x8` 依次赋值给寄存器 `rdi`、`rsi` 和 `rdx`, 如此就可以利用函数 `write` 在控制台输出其 `got` 表地址, 然后可以计算得出 `system` 函数和字符串 `“/bin/sh”` 的地址, 从而绕过 ASLR 的防御, 重新构建 `gadgets` 攻击链即可发起代码重用攻击。

使用 SHCFI 技术对后向控制流保护后, 栈上的原放置 `vulnerable` 函数返回地址变为和其和系统生成的随机数异或后的值, 在实施攻击时, 同样使用 `gadget1 “pop rdi; ret”` 的地址将异或值覆盖, 然而在执行 `ret` 指令前, SHCFI 技术会取出加密后的异或值与随机值进行异或操作, 生成正确的返回地址压入栈中, 导致攻击失败。如果同时篡改影子栈和原始调用栈中的返回地址, 则

此时解密后的值会指向未知方向, 导致程序崩溃, 无法获得 `write` 的 `got` 表地址, 更无法实施代码重用攻击。

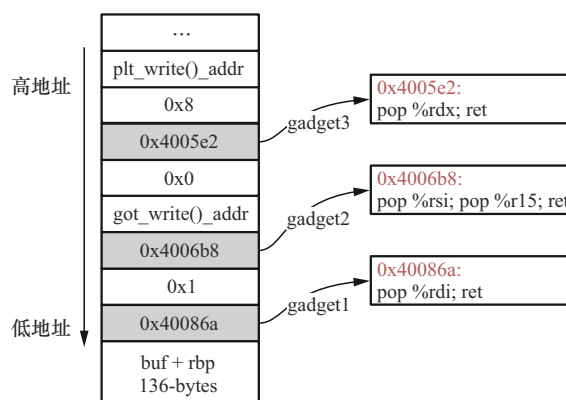


图9 ROP 攻击防御示意

Figure 9 Schematic of ROP attack defense

另外, 本文使用了危害性依次增加的4种攻击类型, 对比所提方法与 Dang 等^[12]归纳的传统影子栈、平行影子栈和随机偏移平行影子栈的防御能力。连续写入表示程序有一个缓冲区溢出漏洞, 而任意写入表示攻击者可以控制栈的任意数据, 信息泄露表示攻击者可以获取栈布局、函数返回地址和平行影子栈的偏移量。

SHCFI 后向控制流保护能力分析如表2所示。

表2 SHCFI 后向控制流保护能力分析

Table 2 Analysis of SHCFI backward control flow protection capability

攻击类型	传统影子栈	平行影子栈	平行影子栈 (随机偏移)	SHCFI
连续写入, 无信息泄露	√	√	√	√
连续写入, 有信息泄露	√	√	√	√
任意写入, 无信息泄露	√	×	√	√
任意写入, 有信息泄露	√	×	×	√

由于攻击者不能通过信息泄露获取传统影子栈隐藏的栈顶指针, 使传统影子栈能够防御4种攻击类型。此外, 每次加载时随机偏移的影子栈在没有信息泄露的情况下, 可以防御攻击者任意写入, 而普通的平行影子栈难以防御攻击者任意写入。SHCFI 技术和传统影子栈具有同样的安全性, 即使攻击者通过信息泄露获取栈偏移量, 能够对

影子栈上的地址任意写入,也无法绕过 SHCFI 的防御,因为栈上的返回地址已经过异或加密,在函数结语处经过解密才能将控制流转移到正确位置。

4.2.2 前向保护测试

Zhang 等^[21]提出平均间接目标减少量 (average indirect target reduction, AIR) 指标来评估对代码重用攻击的防御能力,定义 AIR 为在程序中所有的间接控制流 (indirect control flow, ICF) 传输过程中转移目标的减少比例, AIR 的值越大,说明间接指令可转移的目标地址越少,其对应 CFI 技术的安全性越好。AIR 的定义如下。

$$AIR = \frac{1}{n} \sum_{j=1}^n \left(1 - \frac{|T_j|}{S} \right) \quad (1)$$

其中, $j=1, \dots, n$ 表示程序中所有的 ICF 传输, S 是原始程序中未被保护的 ICF 目标集合, T_j 是使用 CFI 技术之后剩余的 ICF 目标集合。

间接 jmp/call 指令平均间接目标减少量如图 10 所示,间接指令 call 和 jmp AIR 分别为 86.59% 和 90.4%,提高了攻击者利用间接指令构建攻击负载的难度。

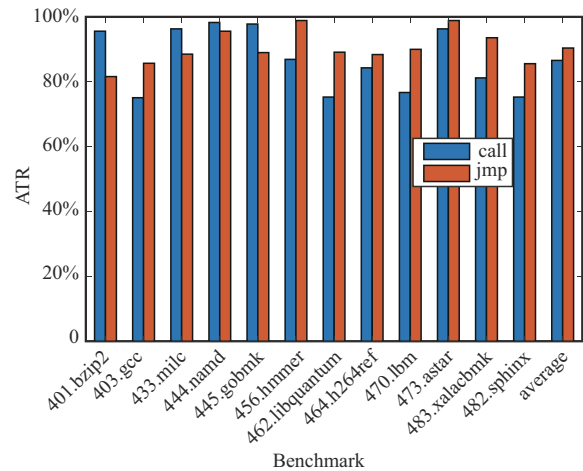


图 10 间接 jmp/call 指令平均间接目标减少量

Figure 10 Average indirect target reduction for indirect jmp/call instructions

4.3 文件膨胀率

表 3 为 SHCFI 部署后文件膨胀率的变化。将表 3 中对应的膨胀率求平均,得到后向与前向的控制流保护对文件的平均膨胀率分别为 15.1% 和 12.8%,总的文件大小增加 24.6%。一般而言,程序本身越小,文件膨胀率越大,如程序 470.lbm 仅有 22.1 kB,而多样化后膨胀率达到 79.64%,程序 462.libquantum 和 473.astar 同样是这种情况。然而,程序 483.xalancbmk 内存大于 445.gobmk,

表 3 SHCFI 部署后文件膨胀率的变化

Table 3 Changes in file expansion rate after SHCFI deployment

程序	文件大小/kB	ret		call/jmp		总膨胀率
		数量/个	膨胀率	数量/个	膨胀率	
401.bzip2	73.7	96	14.05%	23/25	11.05%	23.15%
403.gcc	3773.8	6399	12.77%	446/539	7.18%	16.24%
433.milc	148.0	226	18.92%	7/64	16.92%	29.79%
444.namd	333.1	95	6.14%	17/32	4.91%	12.58%
445.gobmk	4457.5	2765	4.17%	47/62	3.54%	6.12%
456.hmmmer	341.6	626	11.49%	13/95	6.29%	15.32%
462.libquantum	51.1	103	26.78%	3/27	14.78%	38.46%
464.h264ref	594.5	624	6.12%	356/55	11.75%	12.23%
470.lbm	22.1	21	34.19%	3/20	42.23%	79.64%
473.astar	56.2	118	24.75%	4/27	16.75%	32.52%
482.sphinx3	217.9	384	15.23%	11/62	13.35%	16.34%
483.xalancbmk	6175.1	11158	6.94%	132/68	4.77%	12.82%

445.gobmk 需要保护的间接控制流指令较少, 导致文件总膨胀率不足 483.xalancbmk 的一半。可见, 文件的膨胀率主要取决于 SHCFI 技术保护前后代码段指令的数量以及文件本身大小。

5 结束语

本文针对影子栈难以兼顾安全性和性能, 软件实现的前向 CFI 技术性能开销过大、难以进行实际部署, 提出并实现了一种软硬件结合的控制流完整性方法, 分别通过返回地址加密的平行影子栈方案和硬件 ENDBRANCH 状态机指令实现了对前向、后向控制流的保护。通过性能、代码重用攻击防御效果及文件膨胀率等测试结果表明, 该方法能够有效地防御代码重用攻击, 具有良好的运行开销。

参考文献:

- [1] ABADI M, BUDI M H, ERLINGSSON Ú, et al. Control-flow integrity principles, implementations, and applications[J]. ACM Transactions on Information and System Security, 2009, 13(1): 4.
- [2] PAPPAS V, POLYCHRONAKIS M, KEROMYTIS A D. Transparent ROP exploit mitigation using indirect branch tracing[C]//Proceedings of the 22nd USENIX conference on Security. New York: ACM, 2013: 447-462.
- [3] XIA Y B, LIU Y T, CHEN H B, et al. CFIMon: detecting violation of control flow integrity using performance counters[C]//Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012). Piscataway: IEEE Press, 2012: 1-12.
- [4] 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.
- [5] FENG L, HUANG J, HU J, et al. FastCFI: real-time control-flow integrity using FPGA without code instrumentation[J]. ACM Transactions on Design Automation of Electronic Systems, 2021, 26(5): 39.
- [6] LI J F, CHEN L W, SHI G, et al. ABCFI: fast and lightweight fine-grained hardware-assisted control-flow integrity[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(11): 3165-3176.
- [7] ZHANG J L, QI B H, QIN Z, et al. HCIC: hardware-assisted control-flow integrity checking[J]. IEEE Internet of Things Journal, 2019, 6(1): 458-471.
- [8] SHANBHOGE V, GUPTA D, SAHITA R. Security analysis of processor instruction set architecture for enforcing control-flow integrity [C]//Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy. New York: ACM, 2019: 1-11.
- [9] ZOU C W, XUE J L. Burn after reading: a shadow stack with microsecond-level runtime rerandomization for protecting return addresses[C]//Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE). Piscataway: IEEE Press, 2020: 258-270.
- [10] LI J F, CHEN L W, XU Q Z, et al. Zipper stack: shadow stacks without shadow[C]//Proceedings of the Computer Security - ESORICS 2020: 25th European Symposium on Research in Computer Security. New York: ACM, 2020: 338-358.
- [11] ZOU C W, WANG X D, GAO Y Q, et al. Buddy stacks: protecting return addresses with efficient thread-local storage and runtime rerandomization[J]. ACM Transactions on Software Engineering and Methodology, 2022, 31(2): 35e.
- [12] 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.
- [13] BUROW N, ZHANG X P, PAYER M. SoK: shining light on shadow stacks[C]//Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP). Piscataway: IEEE Press, 2019: 985-999.
- [14] 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.
- [15] HUANG T. The design and implementation of A ROP exploit schema bypassing parallel shadow stack[D]. Nanjing: Nanjing University, 2016.
- [16] LILJESTRAND H, NYMAN T, GUNN L J, et al. PACStack: an authenticated call stack[EB]. arXiv Preprint arXiv: 1905.10242, 2019.
- [17] TICE C, ROEDER T, COLLINGBOURNE P, et al. Enforcing forward-edge control-flow integrity in GCC & LLVM[C]//Proceedings of the 23rd USENIX Conference on Security Symposium. New York: ACM, 2014: 941-955.
- [18] NIU B, TAN G. Per-input control-flow integrity[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2015: 914-926.
- [19] NIU B, TAN G. Modular control-flow integrity[C]//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2014: 577-587.
- [20] CRISWELL J, DAUTENHAHN N, ADVE V. KCoFI: complete control-flow integrity for commodity operating system kernels[C]//Proceedings of the 2014 IEEE Symposium on Security and Privacy. Piscataway: IEEE Press, 2014: 292-307.
- [21] 王明华, 尹恒, 苏璞睿, 等. 二进制代码块: 面向二进制程序的细粒度控制流完整性校验方法[J]. 信息安全学报, 2016, 1(2): 61-72.
- [22] WANG M H, YIN H, SU P R, et al. Binary code continent: finer-grained control flow integrity for stripped binaries[J]. Journal of

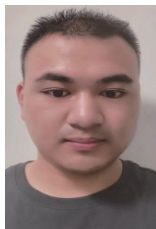
Cyber Security, 2016, 1(2): 61-72.

- [21] ZHANG M W, SEKAR R. Control flow and code integrity for COTS binaries: an effective defense against real-world ROP attacks [C]//Proceedings of the 31st Annual Computer Security Applications Conference. New York: ACM, 2015: 91-100.
- [22] HE W J, DAS S, ZHANG W, et al. BBB-CFI: lightweight CFI approach against code-reuse attacks using basic block information[J]. ACM Transactions on Embedded Computing Systems, 2020, 19(1): 7.
- [23] 张正, 薛静锋, 张静慈, 等. 进程控制流完整性保护技术综述[J]. 软件学报, 2023, 34(1): 489-508.
- ZHANG Z, XUE J F, ZHANG J C, et al. Survey on control-flow integrity techniques[J]. Journal of Software, 2023, 34(1): 489-508.
- [24] 柳童, 史岗, 孟丹. 代码重用攻击与防御机制综述[J]. 信息安全学报, 2016, 1(2): 15-27.
- LIU T, SHI G, MENG D. A survey of code reuse attack and defense mechanisms[J]. Journal of Cyber Security, 2016, 1(2): 15-27.
- [25] HENNING J L. SPEC CPU2006 benchmark descriptions[J]. ACM SIGARCH Computer Architecture News, 2006, 34(4): 1-17.
- [26] CHEN G, JIN H, ZOU D Q, et al. SafeStack: automatically patching stack-based buffer overflow vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing, 2013, 10(6): 368-379.

[作者简介]



梁浩 (1987-), 男, 河南郑州人, 信息工程大学副研究员, 主要研究方向为网络空间主动防御、内生安全。



何本伟 (1998-), 男, 安徽滁州人, 信息工程大学硕士生, 主要研究方向为网络空间安全、软件多样化。



王庆丰 (1995-), 男, 河南周口人, 信息工程大学助理研究员, 主要研究方向为拟态防御、软件多样化。



程国振 (1986-), 男, 山东菏泽人, 信息工程大学副教授, 主要研究方向为网络主动防御、软件多样化。



马海龙 (1980-), 男, 山东沂水人, 信息工程大学副研究员, 主要研究方向为网络空间内生安全、网络威胁智能感知、新型网络体系。