

MVX-CFI: 一种实用的软件安全主动防御架构

姚 东¹, 张 铮¹, 张高斐¹, 邬江兴²

¹ 数学工程与先进计算国家重点实验室 郑州 中国 450001

² 国家数字交换系统工程技术研究中心 郑州 中国 450002

摘要 软件安全是网络空间安全中最重要的一环。早期的软件安全解决方案大多是发现安全威胁后再逐一解决的被动防御方案。为了有效应对各类安全威胁,防御方法逐渐从被动过渡到主动。在众多的主动防御方法中,从系统执行架构角度出发构建内生防御能力的软件多变体执行架构技术受到了广泛关注,它通过异构、冗余执行体之间的相对正确性检查发现攻击行为,不依赖于具体安全威胁的特征检测,可实时检测并防御大多数已知、甚至未知安全威胁。然而,该方法面向实际应用部署存在较大的性能瓶颈。控制流完整性(CFI)是一种理想的安全解决方案,但由于其性能损失和兼容性问题也未被广泛采用。本文将两者有效结合提出一种基于多变体执行架构的CFI(MVX-CFI)。MVX-CFI是一种基于执行架构的、动态、透明的CFI实施方法,它能够有效捕获软件整个运行时控制流的走向并发现由攻击等恶意行为引起的非法路径转移。MVX-CFI通过MVX可形式化验证的高可信表决机制在运行时动态建立描述应用程序高频执行路径的控制流子图(Sub-CFG),并作为检测模型正向反馈到MVX用于辅助检测,减少了传统MVX大量重复的表决工作,提高了MVX的执行性能。Sub-CFG具有在线分离软件执行过程中高频路径和低频路径的能力,这一特性为软件预置后门的检测提供了一种思路。实验评估表明,本文的改进方法提高了原架构的执行效率,同时保证了在安全防御方面的有效性。

关键词 多变体执行; 软件安全; 安全架构

中图分类号 TP309 DOI号 10.19363/J.cnki.cn10-1380/tn.2020.07.04

MVX-CFI: a practical active defense framework for software security

YAO Dong¹, ZHANG Zheng¹, ZHANG Gaofei¹, WU Jiangxing²

¹ State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China

² National Digital Switching System Engineering & Technological R&D Center, Zhengzhou 450002, China

Abstract Software security plays an important role in cyberspace security. Software security solutions in the early days are mostly passive defense solutions that are addressed one after the other when security threats are discovered. To deal with various security threats effectively, the defense method gradually changes from passive to active. Among many active defenses, the technique of software multi-variant execution architecture has been widely concerned. It can detect and defend most known attacks, not depending on the feature of specific threats. By checking the relative correctness between heterogeneous and redundant variants, it can find attacks and other abnormal behaviors. However, there is a big performance bottleneck in this method. Control flow integrity (CFI) is another ideal security solution, but it is not widely used in practice because of performance loss and compatibility problems. In this paper, we propose a multi-variant execution framework called MVX-CFI. MVX-CFI is a dynamic and transparent CFI implementation method based on the execution framework. It can effectively capture the control flow of the target software and find illegal path transfer caused by malicious acts such as attacks. MVX-CFI uses MVX formalized high trust voting mechanism to dynamically construct control flow subgraph (Sub CFG) to describe the high-frequency execution path of the target at runtime, and feedback to MVX as a detection model for assisted detection, which reduces a lot of repetitive voting work of traditional MVX and improves the performance. Sub-CFG can separate the high-frequency path and low-frequency path in the process of software execution online, and it provides a probability to detect preset back door in a software. Experimental evaluation shows that the proposed framework can improve the efficiency of the original MVX framework and perform security defense effectively.

Key words multi-variant execution; software security; security architecture

通讯作者: 张铮, 博士, 副教授, Email: ponyzhang@126.com。

本课题得到国家重点研发计划网络空间安全专项(No. 2018YFB0804003, No.2017YFB0803204)资助。

收稿日期: 2020-03-27; 修改日期: 2020-06-02; 定稿日期: 2020-06-19

1 引言

软件面临的众多安全威胁可以大致分为对非控制数据的攻击^[1-4], 以及对控制流劫持的攻击, 其中后者是最常用的攻击方法。在控制流劫持攻击中, 用于指导程序控制流的控制数据被攻击者破坏, 这些数据(如返回地址、间接跳转目标)加载到程序计数器中时, 程序的执行流程将从其设计路径转移到攻击者构造的路径。控制流劫持攻击的实例包括代码注入^[5]、代码重用^[6]、面向返回的编程(ROP)^[7-8]、面向跳转的编程(JOP)^[9]和面向调用的编程(COP)^[10]等。其中, ROP 攻击正日益成为主流, 它比代码注入攻击更具优势, 可以绕过操作系统广泛使用的数据执行预防(DEP)保护。攻击手段的不断演进和升级要求先进的防御手段来应对, 尤其是主动防御。在众多的主动防御方法中, 基于多变体执行架构的主动防御方案因其实时性强、防御范围广等优点而受到了越来越多研究者的关注。该方案的核心思想是构造基于相对正确公理的内生安全免疫机制, 具体来说, 就是对要防御的目标程序使用多样化方法生成若干个功能等价的变体, 每个变体在运行时对外呈现的攻击面存在空间或其他方面的差异, 攻击者每次构造的攻击负载只能在同一时间对其中一个变体有效, 而在其它变体上表现出不同的状态或行为, 进而通过同步的多模表决机制可发现这些由攻击或其他原因引起的异常。

多变体执行架构最初是为提高系统的可靠性、可用性, 以及离线软件调试或性能分析而设计^[11]。后来这一思想才逐渐被研究者用来解决计算机和网络系统中的安全问题。软件多变体执行架构使用软件多样化技术构建功能等价、攻击面相异的变体集合, 运行时对从集合中选择的每个变体赋予相同的输入, 再通过监控器监视每个变体的输出或行为并检测它们之间的分歧。软件多变体执行架构寻求从执行架构的设计产生内生安全机制, 不追求用“加密”的方法来保护程序中某些可以被攻击者利用的信息, 而是确保参与运行的软件变体在漏洞利用方面相互之间存在差异, 这让某些“熵”空间不足、单独使用时易受攻击的多样化技术可以在执行架构下多个变体之间发挥协同防御的作用, 攻击者使用的某个攻击方法必须同时对架构下的所有变体都有效才能达到攻击目的。这样的执行架构极大的增加了黑客的攻击难度, 同时又充分利用了软件多样化技术, 实现了真正意义上的主动防御。尽管软件多变体执行架构早在 2006 年就被提出, 但该方案并未在实际软件

安全防御中被广泛的采纳和部署, 主要原因是其带来的性能损失较大, 部分架构还牵涉到对硬件或操作系统的修改, 部署实施难度较大。

控制流完整性(CFI)^[12]是一种检测程序既定执行流程是否被改变的安全防御方案, 完备的 CFI 安全控制机制可以有效防御所有的控制流劫持攻击, 它能够提供健壮的且可以形式化验证的安全保证, 是理想的安全防御解决方案。与多变体执行架构类似, 它也由于性能和实施难度等问题阻碍了在实际中的应用。

建立在对这两项技术的详细分析基础之上, 本文提出了 MVX-CFI 解决方案, 旨在通过两项技术的合理结合来克服各自在实际应用中存在的问题。MVX-CFI 以控制流为多变体执行架构的监控对象, 相对于以往的架构, 其对程序行为的把控更加清晰、直观, 进一步通过对实际监控路径的选择性优化, 减少了监控对象的数量, 相应地减少了监控的工作量, 降低了性能损耗。MVX-CFI 充分利用 MVX 可形式化验证的高可信表决机制^[13]在运行时动态建立描述应用程序高频执行路径的控制流子图(Sub-CFG), 并作为检测模型正向反馈到 MVX 辅助检测, 减少了传统 MVX 大量重复的表决工作, 提高了 MVX 的执行性能。Sub-CFG 所具有的在线分离软件执行过程中高频路径和低频路径的能力还为软件预置后门的检测提供了一种新的思路。

具体来说, 本文的主要贡献概括如下:

- 首次尝试将软件多变体执行架构与 CFI 相结合, 提出了一种实用的、可实施于 COTS 二进制程序的安全防御机制。
- 充分利用了 MVX 高可信的表决结果, 设计了基于反馈学习的 Sub-CFG 辅助检测模型, 改进了多变体执行架构的性能。
- 利用 Sub-CFG 在运行时区分程序高频和低频执行路径, 提供了一种基于路径执行频率的软件预置后门在线检测方案, 扩展了软件多变体执行架构的安全威胁检测范围。

文章结构安排如下, 第 2 章介绍多变体执行以及控制流完整性检查的相关研究工作; 第 3 章对当前多变体执行架构存在问题进行分析并提出解决方案; 第 4 章详细阐述 MVX-CFI 架构设计与原理; 第 5 章针对 MVX-CFI 架构做出相应实验评估; 第 6 章进行全文总结。

2 相关研究

多变体运行最早出现在用于检测和定位程序错

误的研究中^[11], 具体做法是让 CPU 在检测模式并行执行两个在逻辑上等价的程序(程序中的代码小片段和数据段利用跳转指令进行重新排序), 在线检测控制转换越界和“野”指针使用类错误。Berger 和 Zorn 后来提出了软件多个复制体进行冗余执行的架构^[14], 使用随机化技术为复制体生成不一样的堆对象布局, 监控标准 I/O 发生的错误。

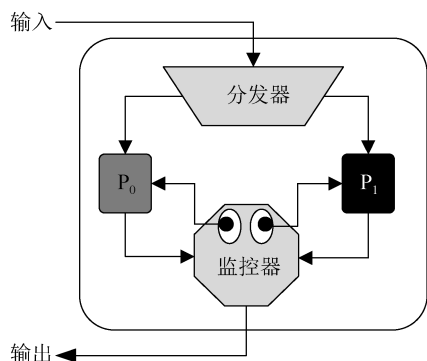


图 1 N-变体系统框架

Figure 1 N-Variant System Framework

2006 年, Cox 提出了一个称为 N-Variant 的较为完整的软件多变体执行架构^[15], 如图 1 所示。为多变体执行架构解决软件安全问题奠定了基础。该架构中的执行体采用了非重叠内存空间的多样化技术, 实现了对依赖于空间信息类攻击的确定性防御。Novark 等人^[16]提出了 Exterminator, 运行时通过表决算法定位内存发生错误的位置和大小, 进而在运行时生成补丁来修正检测到的错误, 这是多变体执行架构中首次引入的动态反馈机制。Aydan 等人^[17]提出了 TightLip, 将拷贝进程置于沙箱, 并且只在处理“敏感数据”时才与原始进程并行工作, 该架构从隔离保护执行体和执行效率方面对原有架构进行了改进, 但沙箱本身又会引入新的攻击面。Weatherwax 分析了构建多变体执行架构过程中易于引入的四类安全漏洞, 基于此提出了一套安全构建多变体执行架构的方法^[18]。Salamat 以尽量不增加原有系统的可信基(Trusted Computing Base, TCB), 即减少额外引入的攻击面为原则提出了 Orchestra 架构^[19], Orchestra 的监控器以独立进程方式运行在用户空间, 且承担了一部分执行体的工作, 解决了多个变体访问竞争资源时产生的不一致问题。Volckaert 等^[20]提出了一个较为完善的架构 GHUMVEE, 并在此基础上对监控机制进行了改进, 提出了 ReMon^[21], ReMon 将监控分为进程内监控与进程外监控, 首次引入了分级监控的设计方法来提高执行效率。Hosek 等^[22]提出了 VARAN, VARAN 对多执行体架构最大的贡献在

于它的事件流异步共享机制。Koning 等^[23]提出了可自定义安全策略的多变体执行架构 MvArmor, 让使用者通过策略设置在执行效率与安全性之间进行折衷, 该架构还可通过环境感知生成变体, 具有一定的灵活性。另外, MvArmor 还使用了基于 Intel CPU VTx 硬件加速的虚拟化系统 Dune^[24], 依靠硬件支持的虚拟化提高监控器的安全性和运行效率。Lu K^[25]提出了多变体执行架构 BUDDY, 它选择 I/O 作为监控对象, 相对于对系统调用的监控, 它减少了表决机制同步比较的次数, 性能损失更小, 不足之处是只能防御信息泄露类攻击。Voulimeneas 首次提出了分布式异构平台多变体架构 DMON^[26], 它将同样的程序分别在使用 Inter 和 ARM CPU 的平台上编译, 然后置于跨平台的同一个多变体架构下执行, 该方案为执行体本身带来了更好的安全性, 但它的实现难度和成本也随之增加。

CFI 是一种通过检查程序运行时执行路径是否符合既定程序流程图(CFG)的安全防御方案。CFI 中的安全策略非常简单: 任何控制流转移路径都不应该偏离其合法路径。为了构建 CFI 策略, 传统的 CFI 首先要从受保护程序中构建一个控制流图(CFG), 然后从中得到每个控制转移的所有合法目标, 在运行时, 内联的强制代码将检查控制流目标是否属于白名单中的目标。Abadi 等人^[12]提出了第一个 CFI 模型, 它使用程序代码中的内联引用监视(IRM)^[27]强制执行 CFI 策略。当给定程序源代码时, IRM 可以通过编译器在每个控制流转移点自动插入, 但是当只给定应用程序二进制文件时, 通常先要解决二进制反汇编和重写两个难题^[28-32]。尽管动态二进制检测(DBI)技术可以动态重写二进制文件而不用反汇编, 但它性能开销很大, 执行速度通常很慢。另外, 静态的二进制重写不可避免地破坏了程序的透明性和完整性, 从而导致改写后的程序与某些操作系统现有安全机制(如系统库保护和远程认证)不兼容。另外一些二进制级 CFI 的研究工作, 例如, BinCFI^[29]、CCFIR^[28]和 BinCC^[33]只执行粗粒度的 CFI 策略, 然而, 高级 ROP 攻击可以绕过这些粗粒度 CFI 的实现^[34], 特别是对于仅使用调用前代码片段的 ROP 攻击能够轻松规避粗粒度 CFI 的检查^[35-36]。为了克服二进制重写和检测的局限性, 最近的研究利用现有的 CPU 硬件特性来帮助 CFI 策略的实施。例如, kBouncer^[37]、ROPecker^[38]、CFIGuard^[39]和 PathArmer^[40]研究中使用分支记录(LBR)跟踪最近执行的间接分支(通常只有 16 个 LBR 条目), 但它易受到历史刷新攻击^[9,35], 这类攻击通过构造恶意负载故意包含虚拟分支指令,

这样 LBR 条目就会被这些刻意构造的虚拟指令扰乱。CFIMon^[41]使用了分支跟踪存储(BTS), 它与 LBR 不同的是可以记录所有先前的间接分支, 但这也带来了更高的性能开销。基于编译器的方法^[2,7,35,42]通过生成更精确的向后(例如, 影子堆栈)和向前(例如, 间接调用检查)控制来实现更细粒度的 CFI 保护, 但该类方法不利于对共享库的检测, 也不适用于 COTS 应用程序。体系结构方法^[1,40,43,44]需要对处理器体系结构或 ISA 进行更改, 不易部署。

3 问题分析和解决方案

软件多变体执行架构是一种新兴的、具有内生安全机制的主动防御型执行架构; CFI 是一种理想的软件安全解决方案。在第二节中我们分析了二者各自在独立应用中遇到的问题, 本节试图通过对这些问题的分析将二者进行有效地结合, 克服各自单独使用中的不足, 强化自身优势, 实现两项技术的“双赢”组合。

从软件多变体执行架构研究发展的过程可以抽象出如图 2 所示的通用多变体执行架构。软件在执行过程中产生的系统调用由输入分发代理分发到各个变体, 在各个变体的运行过程中对系统调用的参数、执行结果等要素进行监控, 然后在预定的检查点处对监控结果进行一致性裁决。

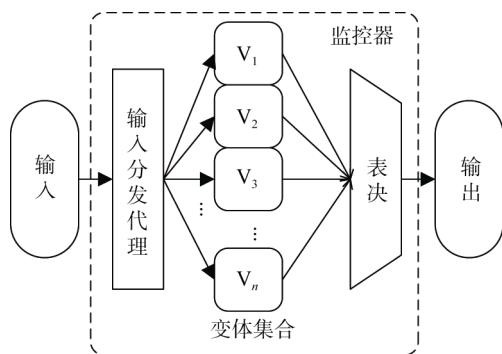


图 2 通用的多变体执行架构

Figure 2 General Multi-Variant Execution Architecture

通过对现有软件多变体通用执行架构的分析可以发现, 影响软件多变体执行架构执行性能的因素主要来源于三个方面: 第一, 监控, 因为在监控点要严格保持变体执行的同步, 所以监控对象的数量越多, 即监控粒度越小, 其工作量越大; 第二, 裁决, 裁决是多变体执行架构实现检测的核心, 原则上有多少监控对象就要执行多少次裁决操作, 另外, 裁

决操作在多数情况下还要在监控点依据裁决策略提取出需要比对的其他特征参数, 过多的工作量是整个执行架构的性能瓶颈; 第三, 缺乏反馈机制, 软件在执行过程中通常包含大量的重复操作, 在多变体执行架构下, 这些重复的操作也会不断被重复地监测。

3.1 CFI 对软件多变体执行架构的改进

3.1.1 监控对象

对于监控对象, 大多数软件多变体执行架构都选择系统调用, 理由是在使用页面级内存保护的现代操作系统中, 每个应用程序都被限制在自己的地址空间内, 因此, 应用程序必须使用系统调用的方式与系统交互, 同样的道理也适用于攻击者利用漏洞进行攻击, 如果攻击的最终目标是破坏目标系统, 或使用系统提供的功能, 那么攻击的有效载荷必须经过系统调用来与系统交互^[18-19]。这种实现在实际应用中有几个弊端: 一是软件的执行经常会反复、大量调用系统中共享的库, 此时的监控会陷入对共享库的执行监控, 不利于对共享库之外软件本身行为的监控和分析; 二是一些操作系统有诸如 VDSO(Virtual Dynamically-linked Shared Object)的实现, 软件对它的使用是不经过系统调用的; 三是某些研究中为了减少监控和裁决的工作量采用了选择性系统调用监控策略, 这种选择受主观经验影响较大, 容易错漏对某些安全威胁的监控。

监控对象的选择应该与攻击密切相关。攻击者对软件的攻击通常从寻找、利用软件漏洞开始, 如栈上的溢出漏洞可以让攻击者改变函数的返回地址, 堆上的缓冲区溢出漏洞可以让攻击者覆盖堆上的数据对象中的函数指针^[45], 整数溢出也可以被用来覆盖堆上的函数指针^[46]。还有一些高级的攻击技术, 如使用后释放(Use-after-free)可以替换对象的虚函数表, 将正常的函数调用地址篡改后指向攻击者需要执行的地方, 如 shellcode。面向返回的攻击技术(return-to-lib、ROP 等)从已有的库或可执行文件中提取指令片段, 构建恶意功能, 但它首先要对返回地址或代码片段中的跳转地址进行修改来实现。由以上分析可以看出攻击者对软件的攻击最直接的反映就是改变了软件的即有执行路径, 即实施了控制流劫持。因此, 要检测对软件的攻击只要对控制流进行监控即可, 即将控制流作为软件多变体执行架构的监控对象。以此为依据, 确定了选择 CFI 与多变体执行架构结合的可行性。

解决了监控对象的选择问题, 另一个问题就是对监控对象的优化。由于软件多变体执行架构在监

控点要对所有变体进行严格的同步执行检查, 之后再开始基于多模裁决的一致性检测, 因此监控对象的数量直接决定了架构的执行效率。另外, 在实际的攻击中, 攻击者能够改变的控制流对象只是整个控制流对象中的某些部分。对监控对象的优化可以进一步准确锁定监控目标, 同时提高整个执行架构的效率。对控制流的监控即监控控制流上所有的转移点(转移指令), 程序中与控制流相关的转移指令有如下几种: (1) 直接跳转/调用指令; (2) 间接跳转/调用指令; (3) 返回指令; (4) 异常处理指令。是否需要对所有这些转移指令进行监控呢? 事实上, 目前流行的操作系统自身都内嵌了一些安全机制对运行其上的程序实施基本的保护, 如数据执行保护(DEP)和内存写保护等, 这些机制的使用本身就确保了它们所保护的控制流对象在运行时无法被攻击者篡改, 因为攻击者对整个控制流路径中能够进行改变的节点只有间接跳转和堆栈中的返回地址, 而正常程序中的间接跳转指令频率不高, 跳转目标正常且有规律^[47], 因此可以对监控对象做出进一步的优化。

直接跳转/调用指令的跳转目标是在代码中固定的, 操作系统为其提供了数据执行保护, 攻击者无法对其进行篡改。间接跳转/调用指令的跳转目标通常是运行时从内存中读取的, 是有可能被攻击者通过漏洞进行篡改的。返回指令的目标位于程序的执行栈, 极易受到攻击者的篡改。异常处理指令通常都是由操作系统来调用, 不会成为间接跳转的目标。综上, 可以将优化后的监控对象限定为间接跳转/调用指令和返回指令。

3.1.2 裁决

裁决操作是整个软件多变体执行架构的另一个主要性能瓶颈, 它的工作量来源于每一次在监控点的对监控对象的比较和分析工作。要提高其性能, 最直接的策略就是减少裁决的工作量。裁决的工作量来源于监控对象的规模(数量)和裁决的工作机制两个环节。

就控制流作为监控对象而言, 对多变体执行架构裁决工作量的改进可以依据以下两点进行:

1) 减少监控对象的数量。这一点通过上一小节对监控对象的优化达成。

2) 减少裁决操作的工作量。假设裁决机制本身没有漏洞使攻击者能完成逃逸, 裁决机制的形式化验证^[13]以很大的概率保证了其输出结果的正确性, 即裁决输出的控制流是正常且未受篡改的, 可以用于动态构建目标程序 CFG, 而 CFG 又是实施传统 CFI 的前提条件, 因此经裁决输出在线形成的 CFG

又可以反馈用于 CFI 检测。这样一来, 原本在一般软件多变体执行架构中的裁决环节就可以分为两步来执行, 即基于 CFG 反馈检测模型的二级检测机制。反馈检测模型在多模型裁决之前对主变体预先实施检测, 接管了之前大量的重复检测工作, 只有无法通过反馈检测模型的控制流转移才由后端的多模裁决进行检测。

3.1.3 基于 CFG 的反馈检测

作为一种主动防御方案, 现有的软件多变体执行架构缺乏动态调整的特性, 即缺乏信息反馈对架构在执行过程中进行动态调整以应对攻击和优化性能的过程。在以往的相关研究中, 只有少量研究使用了非一致裁决信息对执行架构进行终止、执行体替换等负反馈响应, 缺少使用一致性裁决信息进行正向反馈的研究。究其原因, 在以系统调用为监控对象的多变体执行架构中, 使用一致性裁决结果进行正反馈存在较大的难度。首先, 系统调用的数量巨大, 在线存储和分析困难; 其次, 无论是攻击等恶意操作还是正常的软件行为反映在系统调用上是一个序列行为, 而检测序列的定义又需要攻击的先验知识, 这不符合多变体执行架构的主动防御初衷, 且检测需要对序列的精确提取, 错、漏报率高。在多变体执行架构中以控制流为监控对象克服了建立正反馈检测时存在的问题。控制流本身具有描述程序执行行为的特性, 与 CFG 强相关, 提供了建立正反馈检测模型的途径, 可以用来提高软件多变体执行架构的检测效率, 降低架构的性能损耗。

3.2 软件多变体执行架构对 CFI 的改进

控制流完整性(CFI)本身作为一种理想的软件安全解决方案, 在单独使用中也面临着一些难以解决的问题:

1) 构建 CFG。对控制流的追踪必须足够精细才能消除尽可能多的攻击, 它的基础来源于对 CFG 的精确构建, 而静态生成这样的 CFG 是很难完成的^[38]。理论上来说, 从程序源代码或二进制文件中静态提取的 CFG 是与程序控制流高度近似的, 提取过程需要对程序中的每个调用或跳转进行精确的指针分析, 然而, 这种类型的分析实际上很难精确, 这主要是由于 C 语言等底层编程语言的动态特性所致, 因为很多指令目标是在运行时才能计算得到的。另外, 同一个函数可能有多个合法的调用位置, 对应于 CFG 中的多个边, 但在程序运行时, 在任何执行状态下只应允许经过其中一个边, 所以仅依赖静态代码分析无法解决这些问题。

2) 执行效率。现有的 CFI 解决方案大多需要事

先构建完整的 CFG, 而静态构建的 CFG 往往规模较大, 在实际应用中附加的运行时开销也较大。

3) 透明性。为了配合 CFG 模型的检测, 现有的控制流检测方案需要事先在对象程序内部嵌入辅助检测点, 这样不仅破坏了应用程序的透明性, 还无法很好地兼容现有的安全机制。

在软件多变体执行架构下, CFI 应用中的三个主要问题得到了解决。

1) 带有正向反馈辅助检测的多变体执行架构对控制流的检测不依赖于事先构建 CFG, 在正反馈检测模型建立之前, 所有的检测工作都由判决器的一致性多模裁决来完成, 而且经裁决的控制流在正确性方面是有理论保证的。依赖经多模裁决后的控制流, 可以在程序执行期间动态的建立 CFG。与静态建立的 CFG 相比, 它更直观地代表了程序运行期间的实际行为(路径), 因此规模更小、准确度更高。这个动态建立的 CFG 就是用于反馈辅助检测的检测模型。尽管与静态构建的 CFG 相比它缺乏完整度, 但在加入正反馈检测模型后具有两级检测机制的软件多变体执行架构下, 当动态建立的 CFG 检测模型无法完成检测时, 在调度策略的控制下, 控制流还会经过一致性多模裁决, 确保了检测的完整性。

2) 对执行效率的提高从两个方面得到了解决, 第一, 减少了 CFG 静态构建环节的大量工作; 第二, 动态建立的 CFG 路径少(只包含实际执行路径), 规模小, 因此存储空间小, 检测效率高。

3) CFI 检测策略通过软件多变体架构的实施不需向程序内部嵌入 IRM, 保证了程序的透明性, 以及与其它安全手段的兼容性。

4 基于软件多变体执行架构的 CFI——MVX-CFI

基于以上分析, 本文提出了一种新的结合软件多变体执行架构与 CFI 的安全防御解决方案 MVX-CFI, 它能够同时克服各自单独使用时存在的问题, 利于实际部署应用。MVX-CFI 的准确性来自于多个执行体互相监督的相对正确公理, 攻击者在一个执行体上对控制流的改变无法以相同的效果作用于其它执行体; MVX-CFI 是高效的, CFI 通过 MVX 实现了动态应用, 减少了静态环节, MVX 通过 CFI 及其带来的高效、合理反馈减少了监控和裁决的工作量; MVX-CFI 是透明的, 可直接应用于二进制程序, 并兼容现有安全防御措施。

4.1 MVX-CFI 执行架构总览

结合以上分析, 我们设计了如图 3 所示的 MVX-CFI 执行架构。

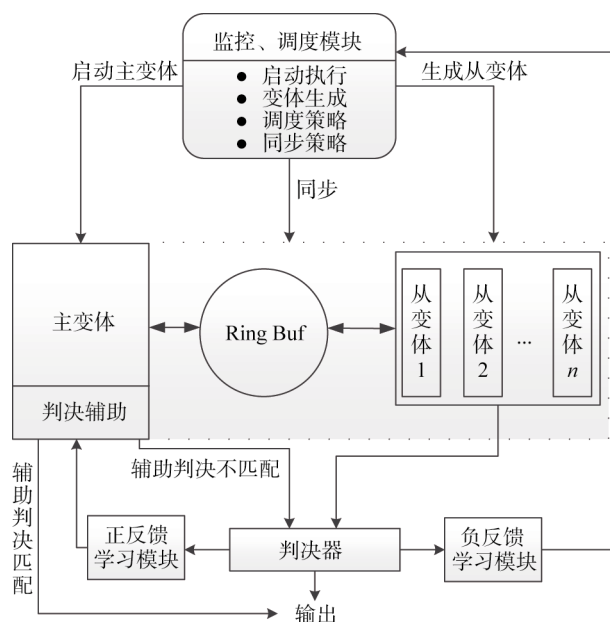


图 3 MVX-CFI 执行架构

Figure 3 MVX-CFI Execution Architecture

MVX-CFI 的工作流程是: 当目标程序作为主变体启动时, 变体生成器使用变体生成策略为每个子变体生成一个目标程序的实例并将它们置于监控模块的监控之下。同步策略通过共享某些竞争类资源的访问和执行结果保证变体执行的一致性和透明性。裁决模块对控制流对象进行裁决, 一致的裁决结果被传递给正反馈学习模块, 正反馈学习模块利用该数据在线生成 CFG。在应用于较大规模的程序时, 为了限制 CFG 规模随目标程序规模增长, 以 CFG 中路径执行频率为依据对其进行进一步优化, 得到 CFG 中相对执行频率较高的子图——Sub-CFG。之后, Sub-CFG 加入到对主变体的在线监测, 此时, 只有在控制流无法经过 Sub-CFG 的检测时才送至后端裁决模块进行多模裁决。负反馈模块的工作是在产生裁决异常时将相关的信息传递到监控、调度模块, 由该模块进行下一步的进程终止、清洗和替换等工作。

4.2 正反馈检测模型(CFG)的动态构建

如前所述, 构建正反馈模型的目的是为了在软件多变体架构执行的过程中加速它的裁决过程。MVX-CFI 的监控和裁决对象为软件控制流, 自然反馈检测模型的作用对象也是软件控制流, CFG 就是最直接有效的控制流模型。MVX-CFI 对控制流多模

裁决的一致性结果给出了构建 CFG 时准确的控制流信息, 同时还具有高可信和高可靠的特性。

程序的结构可以用各种方式来描述。源代码准确地描述了它在运行时对输入如何响应。从源代码静态生成的控制流图只是同一结构的不同视图。在运行时, 二进制代码定义了布局在内存中的程序将执行的操作, 此时动态构建的 CFG 记录了二进制代码基本块的常见执行路径。这些基本块由相邻地址的指令序列组成, 不包括监控对象所定义的转移指令。由于编译完成的程序是确定的, 基本块在内存中的数量、位置和执行顺序也是固定的, 这就是程序在运行时具有的唯一结构。

为了减少裁决的工作量, MVX-CFI 对控制流的检测只针对那些运行时可能被攻击者篡改的转移点, 因此我们在构建反馈检测模型时动态建立的 CFG 是软件完整 CFG 的一个子集, 这也是可将之用于反馈检测的前提。用一个有穷状态自动机 M 来描述静态构建的 CFG, $M = (K, \Sigma, f, S, Z)$, 其中 K 代表 CFG 中的结点集, 即基本代码块, Σ 是程序中所有跳转构成的有穷集, 自动机里的有向边对应 CFG 中的有向边, 自动机的状态转换函数 f 对应 CFG 中的程序控制流向, 它是 $K \times \Sigma \rightarrow K$ 上的映射, 自动机的初始状态 S 对应 CFG 中的首结点, 终止状态 Z 对应 CFG 中的终止结点。用 M' 表示动态构建的 CFG, $M' = (K', \Sigma', f', S', Z')$, M' 中的结点集 K' 只包含 K 中间接跳转可达到点, 忽略了那些确定的、由直接调用到达的点; Σ' 是 Σ 中去除了直接跳转后的子集, 即只包含间接跳转; f' 代表了映射 $K' \times \Sigma' \rightarrow K'$, 即将 f 中所有由直接跳转构成的确定性路径去除后的子集, 所以有 $K' \subset K, \Sigma' \subset \Sigma, f' \subset f, S' = S, Z' = Z$ 。可见, 动态 CFG 的构建过程相当于对 M , 即静态 CFG 的化简, 它只是去除了原来 CFG 中的确定路径, 因此, 两者是等价的, 即 $L(M) = L(M')$ 。

动态 CFG 的构建由于不需要静态反汇编, 所以只需要两个步骤:

1) 划分基本块。基本块是组成控制流图的原子节点, 完整的基本块包含一组顺序执行的语句序列, 有且仅有一个入口指令和出口指令。控制流只能从入口指令进入基本块, 从出口指令离开基本块。由于在 MVX-CFI 架构下动态建立 CFG 只用到经多模一致性裁决输出的间接转移指令, 所以这里的基本块实际上只有入口地址和出口地址来表示, 忽略了中间所有确定性的成分。这里基本块入口指令的范围为: 每个间接调用函数入口的第

一条指令; 所有间接跳转指令的目标指令; 紧跟在间接跳转指令后的指令。出口为下一个入口指令或程序结束指令。

2) 连接基本块。控制流图中的边表示基本块之间的跳转关系, 在这里的 CFG 中, 它实际上就是不断由多模裁决送到的一致性结果——间接跳转关系。

由于 CFG 反馈检测模型是在线动态建立的, 所以几乎不可能在软件运行期间得到该软件完整的 CFG, 即便是一直运行到软件结束也不能保证它在运行过程中执行了所有可能的路径。另一方面, CFG 反馈检测模型只是在软件执行过程中用于辅助检测, 不需要建立完整的 CFG, 它完成不了的工作将由 MVX-CFI 架构中的多模裁决继续完成。因此, 需要为 CFG 的动态建立过程设置一个停止条件。简便起见, 在原型系统中设置一个固定时长 $timespan$ 为停止条件, 如 10 分钟, 在 $timespan$ 时长范围内若再无新增节点则结束 CFG 构建。

4.3 控制流子图(Sub-CFG)

在 MVX-CFI 架构下软件运行时动态得到的 CFG 代表了软件在执行时的一种实际路径视图, 虽然与静态分析得到的全路径视图相比在规模上要小很多, 但实际应用中, 有时也会面对具有更大规模和复杂性的程序, 在这种情况下, 即使是动态建立的运行时 CFG 也会在规模上迅速增长, 此时需要对 CFG 的规模进行进一步的优化。

在程序的实际运行中, 某些基本块经常被执行, 而有一些基本块很少被执行, 例如特定的菜单项、配置选项或协议附加组件等, 也就是说, 在 CFG 中有些路径经过的频率很高, 有些路径则很少经过。因此我们可以根据路径的执行频率来剔除 CFG 中那些很少经过的路径和相应地基本块, 让用于辅助检测的 CFG 只包含软件运行时经常重复执行的“热”区域, 而将对执行频率相对较低的路径的检测交由多变体执行架构后端的多模裁决来完成。基于这一思想, 我们对 CFG 进行进一步优化, 得到它的子图——Sub-CFG。其创建过程如图 4 所示。

原始 CFG 中的节点代表基本功能块, 边代表了间接控制流的转移, 边的权重代表了该路径执行的次数, Sub-CFG 构建时选择保留权重值超过阈值的边。

由于程序在运行时大多数时间都集中在某些功能模块形成的高频路径, 所以 Sub-CFG 的规模小于原始 CFG, 作为辅助检测模型不会消耗过多的系统资源。

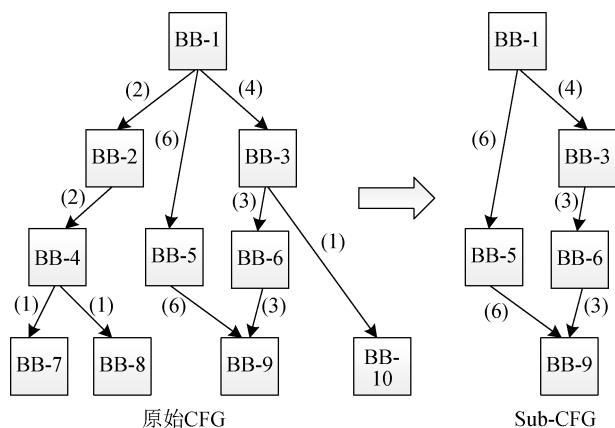


图 4 构建 Sub-CFG 的过程

Figure 4 The process of building Sub-CFG

4.4 MVX-CFI 的非法路径转移检测原理

在子变体生成并纳入监控之后, 监控模块使用 `ptrace`^[48] 的单步执行功能在预设的同步点(监控点)对 `eip` 指针进行追踪, 由于变体在内存空间上的布局是异构的, 所以不能对间接跳转指令和调用指令的参数进行直接比对, 而是要看参数指向位置代表的内容是否一致, 如对 `call` 指令要检查其函数指针是否指向同一个函数, 对 `ret` 指令则检查其结果是否为 `call` 指令函数调用前 `eip` 的下一条指令。

如图 5 所示, 变体生成时使用了多种空间布局随机化技术, 攻击所依赖的绝对地址和相对地址在不同变体空间中都发生了变化。假设该程序存在栈溢出漏洞, 攻击者可以越过变量 `A` 的边界去覆盖变体 1(主变体)栈上的返回地址 `ret`, 而同样的负载在作用于变体 2 和变体 3 时都无法准确覆盖 `ret`, 进而在对控制流的多模裁决到达该点时, 裁决模块在三个变体中取出的 `eip` 会对应不同的内容, 产生不一致的裁决结果。攻击者要成功实施攻击必须构造一个有效负载同时三个变体中同时生效, 这极大的增加了攻击的难度。除非攻击者可以一次试错成功, 否则

变体1	变体2	变体3
...
ret 0x6015f00	ret 0x7015f00	ret 0x8015f00
C	C	C
...
B	B	B
...
A	A	A
...
...
...
...

图 5 变体空间布局

Figure 5 The layout of variant

在每次有不一致发生时调度策略都会对变体进行清洗和替换, 让攻击者对目标的认知无法持续有效。

4.5 基于 MVX-CFI 的软件预置后门检测原理

预置后门作为一种完成特定功能的模块, 通常在开发时由开发人员隐藏在正常的软件代码当中, 在软件运行时由特定的条件或特定的远程指令来触发。假设在软件正常的运行过程中预置后门很少被触发, 也就是说相对于软件的正常功能, 它的使用频率相对较低, 则在 Sub-CFG 依赖于高频执行路径的构建过程中, 基本不会涉及预置后门的执行路径。MVX-CFI 在加入 Sub-CFG 辅助检测后实施的是一种混合检测机制, 根据代码的执行频率分离出的 Sub-CFG 在检测过程中可以有效地将程序执行过程中的高频执行部分和低频执行部分分离。基于这种分离以及对预置后门执行规律的假设, 就可以在 MVX-CFI 的多模裁决机制中加入对后门感知的规则(如敏感系统调用、行为特征等), 实现在线软件预置后门检测。

5 实验评估

5.1 性能测试

由于目前我们还没有条件将 MVX-CFI 与其他多变体执行架构进行横向对比实验, 所以我们使用 SPEC CINT2006 基准对辅助裁决加入 MVX-CFI 前后的执行性能进行了对比。实验平台基于虚拟机搭建, 硬件环境: CPU Intel i7-6700 4.00GHz, 内存 12G, 硬盘 40G; 软件环境: Ubuntu18.04-amd64, 编译环境为 gcc, g++ & gfortran 4.1.2, gcc 和 g++ 的优化参数使用 O2。测试结果如表 1 所示(seconds 表示执行时间; ratio 表示执行性能, 数值越大, 性能越快)。

直观地看, 加入辅助判决之后 MVX-CFI 的执行效率只有小幅提升, 这主要是由于后者在运行初期需要执行与前者相同的多模裁决工作, 此外还要执行 CFG 和 Sub-CFG 的构建, 对表决工作量的减少只有在 Sub-CFG 建立之后才能体现出来, 由于测试集运行时间所限, 这个优势没有很明显的反映在测试结果中, 但总的来说, 在有限的测试时间内, 后者的执行效率依然优于前者。在实际应用中随着程序运行时间的增长, 后者的优势会更加明显。

从测试成绩上也能看出, 影响 MVC-CFI 多变体执行架构效率的主要因素是程序中存在的跳转数量, 当跳转数量较多时, 判决比较的工作量随之增加, 相应地会影响执行效率, 如测试用例 445.gobmk, 它是一个 C 语言实现的围棋程序, 在执行期间的大量

决策会产生大量跳转操作, 因此它在多变体架构下的执行效率远不如另一个主要用于实现计算功能的测试用例 462.libquantum。

表 1 MVX-CFI 性能对比测试结果

Table 1 MVX-CFI Performance Comparison Test

测试集	MVX-CFI		MVX-CFI (Sub-CFG 辅助)	
	Seconds	Ratio	Seconds	Ratio
400.perlbench	199	50.2	196	51.0
401.bzip2	331	30.2	319	31.3
403.gcc	186	53.8	183	54.6
429.mcf	283	35.3	265	37.7
445.gobmk	345	29.0	343	29.2
456.hmmmer	269	37.1	267	37.5
458.sjeng	380	26.3	373	26.8
462.libquantum	251	39.8	224	44.6
464.h264ref	354	28.2	351	28.5
471.omnetpp	309	32.4	300	33.3
473.astar	323	31.0	317	31.5
483.xalancbmk	166	60.2	161	62.1

5.2 安全防御有效性测试

为了检测 MVX-CFI 在防御漏洞攻击方面的实际工作情况, 我们选用了两种类型的漏洞来进行验证, 分别是基于栈的缓冲区溢出漏洞(CVE-2013-2028)和基于堆的缓冲区溢出漏洞(CVE-2014-0160)。

针对 CVE-2013-2028, 攻击发起时会首先通过堆栈读取来泄露堆栈 canary 以及存在漏洞函数的栈帧返回地址, 基于该地址计算 Nginx 二进制文件的基址, 进一步的建立 ROP 攻击链, 从而获取 shell。

针对 CVE-2014-0160, 通过发送可控制的畸形数据给服务端, 进一步覆盖堆上的缓冲区, 从而导致大量的敏感信息返回给客户端。

在 MVX-CFI 架构下测试时分两种情况进行:

1) 漏洞在第一次发送数据时触发;

2) 先进行若干次正常的数据发送, 再发送可以触发漏洞的数据。

这是为了验证 MVX-CFI 在多模表决机制下和在辅助表决机制下都能有效拦截针对漏洞的利用。经过实验, 两种情况下 MVX-CFI 对溢出漏洞都进行了有效防御。由于原理验证系统中还没有加入负反馈处理的相关功能, 所以在发现问题后只是简单地对进程进行了终止, 结果如图 6 所示。

```
MVX-CFI - monitoring.....
MVX-CFI - =====
MVX-CFI - warning: call arguments mismatch
MVX-CFI - state: abnormal
MVX-CFI - monitor is terminated.
root@ubuntu:/home#
```

图 6 安全防御有效性测试

Figure 6 Security Defense Effectiveness Test

6 结论

本文对多变体执行架构和 CFI 两种主动防御技术面向实际部署应用时存在的问题进行了分析, 在此基础上对它们进行了有效结合, 提出了一种新的软件多变体执行架构——MVX-CFI。新的架构克服了两种技术单独应用时存在的问题, 引入的辅助表决机制提高了架构的性能, 同时还为软件预置后门的检测提供了新的思路。

存在不足的是, 在原理验证系统中, 我们对主要功能进行了实现和验证, 还有大量的细节需要在未来的研究工作中进一步完善。由于缺乏对软件预置后门共性特征的研究, 所以我们在多模检测规则中还没有加入相应的检测功能。对于提出的检测原理的验证有待在后续的工作中完成。

致谢 本文工作受到国家重点研发计划网络空间安全专项(No.2018YFB0804003, No.2017YFB0803204)资助。

参考文献

- [1] S. Chen, J. Xu, E. C. Sezer, et al. Non-control-data attacks are realistic threats[C]. *The 14th Conference on USENIX Security Symposium (SEC'05)*, 2005:265-275.
- [2] N. Carlini, A. Barresi, M. Payer, et al. Control-flow bending: On the effectiveness of control-flow integrity[C]. *USENIX Security Symposium (SEC'15)*, 2015: 161-176.
- [3] H. Hu, Z. L. Chua, S. Adrian, et al. Automatic generation of data-oriented exploits[C]. *USENIX Security Symposium (SEC'15)*, 2015: 177-192.
- [4] H. Hu, S. Shinde, S. Adrian, et al. Data-oriented programming: On the expressiveness of non-control data attacks[C]. *IEEE Symposium on Security and Privacy (SP'16)*, 2016: 969-986.
- [5] Levy E. Smashing the Stack for Fun and Profit[J]. *Phrack Magazine*, 1996, 8(49): 1-25.
- [6] R. Wojtczuk. The advanced return-into-lib (c) exploits: Pax case study[J]. *Phrack Magazine*, 2001, 0x0b(0x3a): 0x04 -0x0e.

- [7] S. Checkoway, L. Davi, A. Dmitrienko, et al. Return-oriented programming without returns[C]. *The 17th ACM Conference on Computer and Communications Security (CCS'10)*, 2010: 559-572.
- [8] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)[C]. *The 14th ACM Conference on Computer and Communications Security (CCS'07)*, 2007: 552-561.
- [9] F. Schuster, T. Tendyck, J. Powny, et al. Evaluating the effectiveness of current anti-ROP defenses[C]. *The 17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID'14)*, 2014: 88-108.
- [10] T. Bletsch, X. Jiang, V. Freeh. Mitigating code-reuse attacks with control-flow locking[C]. *The 27th Annual Computer Security Applications Conference (ACSAC'11)*, 2011:353-362.
- [11] K. C. Knowlton. A Combination Hardware-Software Debugging System[J]. *IEEE Transactions on Computers*. 1968, 100(1): 84-86.
- [12] M. Abadi, M. Budiu, U. Erlingsson, et al. Control-flow integrity[C]. *The 12th ACM Conference on Computer and Communications Security (CCS'05)*, 2005: 340-353.
- [13] H. Hu, J. Wu, Z. Wang et al. Mimic defense: a designed-in cybersecurity defense framework[C]. *IET Information Security*. 2018,12(3): 226-237.
- [14] Berger E D, Zorn B G. DieHard: Probabilistic Memory Safety for Unsafe Languages[C]. *Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation-PLDI '06*, 2006: 158-168.
- [15] B. Cox, D. Evans, A. Filipi, et al. N-Variant Systems: A Secretless Framework for Security through Diversity[C]. *USENIX Security Symposium (SEC'06)*, 2006: 105-120.
- [16] G. Novark, E. D. Berger, B. G. Zorn. Exterminator: Automatically Correcting Memory Errors with High Probability[J]. *Communications of the ACM*, 2008, 51(12): 87-95.
- [17] A. R. Yumerefendi, B. Mickle, L. P. Cox. TightLip: Keeping Applications from Spilling the Beans[C]. *The 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI'07)*, 2007:568-578.
- [18] E. Weatherwax, J. Knight, A. Nguyen-Tuong. A Model of Secretless Security in N-Variant Systems[C]. *Workshop on Compiler and Architectural Techniques for Application Reliability and Security (CATARS'09)*, 2009:568-572.
- [19] Salamat B, Jackson T, Gal A, et al. Orchestra: Intrusion Detection Using Parallel Execution and Monitoring of Program Variants in User-space[C]. *Proceedings of the fourth ACM european conference on Computer systems - EuroSys '09*, 2009: 33-49.
- [20] Volckaert S, de Sutter B, de Baets T, et al. GHUMVEE: Efficient, Effective, and Flexible Replication[M]. *Foundations and Practice of Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013: 261-277.
- [21] S. Volckaert, B. Coppens, A. Voulimeneas, et al. Secure and Efficient Application Monitoring and Replication[C]. *2016 USENIX Annual Technical Conference (ATC'16)*, 2016: 167-179.
- [22] P. Hosek, C. Cadar. VARAN the Unbelievable: An Efficient N-version Execution Framework[C]. *ACM Special Interest Group on Programming Languages (SIGPLAN'15)*, 2015: 339-353.
- [23] K. Koning, H. Bos, C. Giuffrida. Secure and Efficient Multi-variant Execution Using Hardware-assisted Process Virtualization[C]. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'16)*, 2016: 431-442.
- [24] A. Belay, A. Bittau, A. Mashtizadeh, et al. Dune: Safe User-level Access to Privileged CPU Features[C]. *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12)*, 2012: 335-348.
- [25] Lu K J, Xu M, Song C Y, et al. Stopping Memory Disclosures via Diversification and Replicated Execution[J]. *IEEE Transactions on Dependable and Secure Computing*, 2018: 1.
- [26] A. Voulimeneas, D. Song, F. Parzefall, et al. DMON: A Distributed Heterogeneous N-Variant System[DB/OL]. arXiv preprint arXiv:1903.03643, 2019.
- [27] U. Erlingsson. The Inlined Reference Monitor Approach to Security Policy Enforcement [D]. Cornell University, Ithaca, 2004.
- [28] C. Zhang, T. Wei, Z. Chen, et al. Practical control flow integrity and randomization for binary executables[C]. *2013 IEEE Symposium on Security and Privacy (S&P'13)*, 2013: 559-573.
- [29] M. Zhang, R. Sekar. Control flow integrity for cots binaries[C]. *Presented as part of the 22nd USENIX Security Symposium (SEC'13)*, 2013: 337-352.
- [30] Qiao R, Zhang M W, Sekar R. A Principled Approach for ROP Defense[C]. *Proceedings of the 31st Annual Computer Security Applications Conference on - ACSAC 2015*, 2015: 101-110.
- [31] V. Mohan, P. Larsen, S. Brunthaler, et al. Opaque control-flow integrity[C]. *Symposium on Network and Distributed System Security (NDSS'15)*, 2015: 27-30.
- [32] V. van der Veen, E. Göktas, M. Contag, et al. A tough call: Mitigating advanced code-reuse attacks at the binary level[C]. *2016 IEEE Symposium on Security and Privacy (S&P'16)*, 2016: 934-953.
- [33] Wang M H, Yin H, Bhaskar A V, et al. Binary Code Continent: Finer-Grained Control Flow Integrity for Stripped Binaries[C]. *Proceedings of the 31st Annual Computer Security Applications Conference on - ACSAC 2015*, 2015: 331-340.
- [34] L. Davi, A.R. Sadeghi, D. Lehmann, et al. Stitching the gadgets: On the ineffectiveness of coarse-grained control-flow integrity protection[C]. *The 23rd USENIX Conference on Security Symposium (SEC'14)*, 2014: 401-416.

- [35] N. Carlini, D. Wagner. ROP is still dangerous: Breaking modern defenses[C]. *The 23rd USENIX Conference on Security Symposium (SEC'14)*, 2014: 385-399.
- [36] E. Gökta, E. Athanasopoulos, M. Polychronakis, et al. Size does matter: Why using gadget-chain length to prevent code-reuse attacks is hard[C]. *23rd USENIX Security Symposium (SEC'14)*, 2014: 417-432.
- [37] V. Pappas, M. Polychronakis, A. D. Keromytis. Transparent ROP exploit mitigation using indirect branch tracing[C]. *The 22nd USENIX Conference on Security (SEC'13)*, 2013: 447-462.
- [38] Cheng Y Q, Zhou Z W, Yu M, et al. ROPecker: A Generic and Practical Approach for Defending Against ROP Attacks[C]. *2014 Network and Distributed System Security Symposium*, 2014: 265-276.
- [39] Yuan P H, Zeng Q K, Ding X H. Hardware-Assisted Fine-Grained Code-Reuse Attack Detection[M]. *Research in Attacks, Intrusions, and Defenses*. Cham: Springer International Publishing, 2015: 66-85.
- [40] van der Veen V, Andriesse D, Göktaş E, et al. Practical Context-Sensitive CFI[C]. *The 22nd ACM SIGSAC Conference on Computer and Communications Security-CCS '15*, 2015: 927-940.
- [41] Y. Xia, Y. Liu, H. Chen, et al. CFIMon: Detecting violation of control flow integrity using performance counters[C]. *The 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'12)*, 2012: 1-12.
- [42] E. Bosman, H. Bos. Framing signals — return to portable exploits. (working title, subject to change.)[C]. *2014 IEEE Symposium on Security and Privacy (S&P'14)*, 2014:869-873.
- [43] Chen P, Xiao H, Shen X B, et al. DROP: Detecting Return-Oriented Programming Malicious Code[M]. *Information Systems Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009: 163-177.
- [44] C. Cowan, C. Pu, D. Maier, et al. Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks[C]. *USENIX security symposium (SEC'98)*, 1998: 63-78.
- [45] OpenBSD's IPv6 mbufs remote kernel buffer overflow. <https://www.coresecurity.com/content/open-bsd-advisorie>, 2007.
- [46] M. Daniel, J. Honoroff, C. Miller. Engineering Heap Overflow Exploits with JavaScript[C]. *WOOT*, 2008: 1-6.
- [47] P. Chen. Research on the Attack and Defense Techniques of Code Reuse[D]. Nanjing University, 2012.
- [48] S. Krahmer, Control Flow Integrity with ptrace (). <https://www.mendeley.com/catalogue/e5d96206-a0db-3401-a3ab-aa9709b00a73/>, 2006.



姚东 于 2013 年在信息工程大学计算机科学与技术专业获得硕士学位, 现在信息工程大学计算机科学与技术专业攻读博士学位, 研究领域为网络安全。Email: dojn_dd@163.com



张铮 于 2006 年在信息工程大学计算机科学与技术专业获得博士学位。现任数学工程与先进计算国家重点实验室副教授。研究领域为网络安全、先进计算。研究兴趣包括: 主动防御技术、高性能计算。Email: ponyzhang@126.com



张高斐 于 2018 年在郑州轻工业大学网络工程专业获得学士学位。现在信息工程大学计算机科学与技术专业攻读硕士学位。研究领域为网络安全。研究兴趣包括: 主动防御技术、网络体系结构。Email: gaofei_zhang@163.com



邬江兴 现任国家数字交换系统工程技术研究中心主任, 教授, 博导。研究领域为信息通信网络、网络安全。研究兴趣包括: 主动防御、交换技术与宽带信息网络、高效能计算。Email: 17034203@qq.com