控制流完整性的发展历程

武成岗，李建军
中科院计算所

关键字：控制流完整性 控制流劫持 安全保障

1、引言

控制流劫持是一种危害性极大的攻击方式，攻击者能够通过它来获取目标机器的控制权，甚至进行提权操作，对目标机器进行全面控制。当攻击者掌握了被攻击程序的内存错误漏洞后，一般会考虑发起控制流劫持攻击。早期的攻击通常采用代码注入的方式，通过上载一段代码，将控制转向这段代码执行。为了阻止这类攻击，后来的计算机系统中都基本上都部署了DEP(Data Execution Prevention)机制，通过限定内存页不能同时具备写权限和执行权限，来阻止攻击者所上载的代码的执行。为了突破DEP的防御，攻击者又探索出了代码重用攻击方式，他们利用被攻击程序中的代码片段，进行拼接以形成攻击逻辑。代码重用攻击包括Return-to-libc、ROP(Return Oriented Programming)、JOP（Jump Oriented Programming）等。研究表明，当被攻击程序的代码量达到一定规模后，一般能够从被攻击程序中找到图灵完备的代码片段。

为了抵御控制流劫持攻击，加州大学和微软公司于2005年提出了控制流完整性（Control Flow Integrity，CFI）的防御机制。其核心思想是限制程序运行中的控制转移，使之始终处于原有的控制流图所限定的范围内。具体做法是通过分析程序的控制流图，获取间接转移指令（包括间接跳转、间接调用、和函数返回指令）目标的白名单，并在运行过程中，核对间接转移指令的目标是否在白名单中。控制流劫持攻击往往会违背原有的控制流图，CFI使得这种攻击行为难以实现，从而保障软件系统的安全。

CFI从实现角度上，被分为细粒度和粗粒度两种。细粒度CFI严格控制每一个间接转移指令的转移目标，这种精细的检查，在现有的系统环境中，通常会引入很大的开销。而粗粒度CFI则是将一组类似或相近类型的目标归到一起进行检查，以降低开销，但这种方法会导致安全性的下降。

CFI已经被提出十多年的今天，依然有许多研究者在探索新的CFI技术，使其在可接受的开销下能获得高安全性，图1给出了CFI的发展历程，本文将介绍在该历程中的重要的技术创新和相关研究成果。

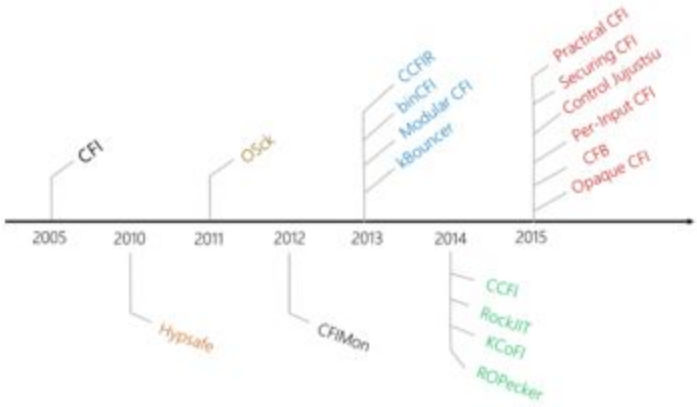


图1. CFI主要技术发展历程

2、控制流完整性的发展历程

2.1 基于插桩技术的CFI

为了有效防御控制流劫持攻击，2005年Martín Abadi和Mihai Budiu,等人提出了控制流

完整性（Control-Flow Integrity，CFI）的思路[1]。程序在其执行过程中，应当遵循程序时预先定义好的控制流图（Control Flow Graph, CFG），以确保程序控制流不被劫持或非法篡改，背离程序编制时所设计的控制流转移关系。CFI在运行时检测程序的控制转移是否在控制流图中，以识别是否遭遇了攻击。具体作法是在控制流转移指令前插入检验代码，来判断目标地址的合法性。这种做法能够对控制流劫持攻击起到防御作用，但是也存在一些问题。

严格意义上的CFI，需要做到对每条间接控制转移都做检查，并确保每条转移指令只能转移到它自身的目标集合。如果要达到更好的防御效果，则要做到上下文敏感的检查。然而，这种检查机制虽然能够最大程度地提升系统的安全性，但其开销过大，难以得到实际部署。

Chao Zhang 等人指出，CFI未被广泛应用的原因是插桩引入的开销过大，需要额外的信息（比如间接控制转移可能的目标集合）的支持，且不能进行增量式的部署。为此，他们提出了CCFIR[2]（Compact Control Flow Integrity and Randomization）。其策略是对间接调用指令和函数返回指令的目标进行区分，阻止未经验证的返回指令跳转到敏感函数的行为。这些限制囊括了CFI保护的主要思想，且不需要别名分析。出于效率和兼容性的考虑，他们通过一个专用的“Springboard section”来实现间接分支转移，并限定代码对齐，对跳转目标进行限制。经SPECint2000进行测试，其平均/最高性能开销分别为3.6%和8.6%（平均/最高）。为了进一步提升安全性，“Springboard section”在程序启动时随机地变换允许的跳转目标，进一步增加了控制流劫持的难度。CCFIR是一个纯粹的二进制转换程序，不依赖源码或调试信息，所依赖的仅是重定位表中的信息。受保护的代码可以被独立的验证，也可以进行增量式的部署。

Mingwei Zhang 等人提出了一种针对COTS（Commercial-off-the-shelf）程序的CFI机制——binCFI[3]，目标是使CFI直接应用于已有的商用应用上。他们将间接转移指令的操作数分为代码指针、异常处理程序入口、导出符号地址和返回地址等类型，通过精细的静态分析，对不同类型的间接控制转移，收集它们的合法目标集合，如返回指令的合法目标集合就是所有函数调用指令的下一条指令的地址，导出符号地址集合就是ELF文件中.dynamic段中存储的地址。利用这种方法，binCFI可以得到与已有CFI方案相当的安全性。通过新增代码段的方式，不改变原有的代码，达到完全透明的目的。实验表明，论文中提出的方案可以应用于大规模的二进制程序中，拥有不错的安全性。

相对于严格意义上的CFI，CCFIR[2]和binCFI[3]都属于粗粒度CFI机制。所谓粗粒度CFI，就是放宽检查的条件，减少比较目标集合。原来的CFI是一个间接转移对应一个目标集合。放宽后，间接转移目标进行合并，比如把所有的间接调用指令和间接跳转指令的目标集合合成一个目标，所有的函数返回指令的目标集合合成一个。这种方式能够有效地降低CFI引入的开销，但也存在安全性问题。

Enes Göktas在Overcoming CFI[4]中，利用两种特殊的gadget——entry point (EP) gadget和call site (CS) gadget，来绕开粗粒度CFI机制的防御。这两种gadget满足检查的条件，但是其实是并不符合真实的控制流。Enes Göktas对这两种gadget的有效性进行了论述，并利用其构造ROP链、调用库函数，形成实际的攻击。

为了对抗Overcoming CFI对粗粒度CFI的攻击，Ali Jose Mashtizadeh 等人提出了一种通过对代码指针加密，来增强CFI的方法，称为CCFI[5]。粗粒度CFI的问题本质就在于归类使得攻击者有了可乘之机。CCFI就是要对代码指针做更细致地划分，还不能回归到CFI原本定义的那种细粒度的分类。他们将代码指针细分成四类，分别是函数指针类、return指针类（函数返回地址）、方法指针类、虚表指针类（后两种都是针对C++语言特有的类）。每次在保存一个代码指针时，将其运行时信息加密保存。每类指针对应一些具有标识作用的运行时信息。在这些指针解引用（间接转移）时，解密信息并比对。另外为了加快加解密的速度，CCFI使用了处理器中独特的指令“AES-IN”对AES加解密算法加速，同时为了保证密钥的安全，又将密钥保存在处理器中独特的寄存器中。他们修改了开源编译器LLVM，定制了一个编译器，并用其编译了一些程序，在实际的攻击环境中检测其有效性。实验证明，他们的方法在安全性和性能方面取得了很好的折衷。

Lucas Davi 等人在Stitching the Gadgets[6]中详细分析了不同的CFI解决方案包括kBouncer、ROPecker、binCFI、ROPGuard和Microsoft EMET4.1，并指出它们的安全性不容乐观。他们首先重新思考了不同的粗粒度的CFI的假设，对kBouncer、ROPecker、binCFI、ROPGuard和Microsoft EMET4.1进行了详尽的安全性分析，特别地，整合了这些现有粗粒度CFI，形成了一个安全策略更为严格的CFI系统。在这个更为严格的条件下，完

成了图灵完整性的验证。为了更好地展示攻击能力，他们在Adobe Reader和mPlayer上进行了实验，证明了粗粒度的CFI无法达到预期的安全效果。论文[7]也探索对CFI进行攻击的手段。多数防御人员认为，通过影子栈（Shadow Stack）来检测函数返回目标，再加上DEP和ASLR的保护，栈应该会变得非常安全，从而将重心转向堆数据的安全保护，然而，作者发现，栈还是会受到攻击的。他们提出了三种攻击攻击方法：一是利用堆上的漏洞来破坏栈上的callee-saved寄存器保存区域，使得callee-saved寄存器被劫持；二是利用用户空间和内核之间进行上下文切换的问题，来劫持sysenter指令，使控制跳转到攻击者想跳转的位置；三是通过泄露主栈的地址来泄露出来shadow stack的地址，进而进行攻击。

以往CFI方案的问题是只实施了控制流不敏感策略。上下文信息的缺少不可避免地降低了CFI的防御能力，从而给了攻击者利用空间。上下文敏感的CFI（Context sensitive CFI）是有望解决这一问题的方法，它依赖于上下文敏感的静态分析，将CFI不变量和CFG中的控制流路径联系到一起，运行时在执行路径上强制执行这些不变量。CCFI早在CFI提出之时已被认可，但因其在现实应用中不实际而被迅速废弃。Victor van der Veen 等人提出的Practical Context-Sensitive CFI[8]，展示了一个可用于现实应用程序的高效、可靠、实用的上下文敏感CFI方案：PathArmor。PathArmor依赖于商用硬件的支持，高效可靠地监控通往敏感函数（可用于实施控制流转移攻击）的执行路径；使用仔细优化的二进制插桩设计，在被监控路径上强制执行上下文敏感CFI的不变量。PathArmor的路径不变量是按需在CFG上，通过一定范围内的上下文敏感静态分析得到的，而且路径验证步骤使用了caching来提高验证效率。路径验证本身也是非常高效的，因为所有的CFI检查都在敏感函数点处集中完成。

John Criswell将CFI做到操作系统内核中，使之免受经典的控制流劫持、return2user和代码段修改攻击，该系统称为KCoFI[9]。他们在基于标签的控制流间接转移保护的基础上，加入一个运行时监控的软件层，负责保护一些关键的操作系统数据结构和监控操作系统进行的所有底层状态操作。并且还通过形式化方法，证明了一个子集的正确性。证明涵盖了页表管理、异常处理、上下文切换和信号分派等操作。实验表明，KCoFI阻止了攻击者将控制流向FreeBSD Kernel中的任何gadget的转移。与未修改的内核相比，KCoFI在运行server测试集时具有较小的开销，但运行文件系统操作密集型的测试集时，开销较大。

2.2 利用硬件来提升CFI的效率

为了能够在性能和防御方面取得更好的效果，一些研究着手于利用现有的硬件机制，来降低CFI的开销。

Vasilis Pappas提出利用硬件性能计数器，在运行时观察执行流的思路，该方法被称为kBouncer[10]。他们利用LBR（Last Branch Register）来捕获最近的16次跳转信息。具体做法是在敏感系统调用处，对捕获的16次跳转进行安全性判断，即return指令需要跳转到调用点的后继位置，indirect-call指令的目标是函数入口，其余跳转指令目标基本块长度不能全部少于20条指令。为了避免攻击者利用库函数调用来完成攻击，文章在所有的库函数调用点，来进行上述合法性检查。为了验证kBouncer的防御效果，作者对IE浏览器、Adobe Flash Player和Adobe Reader进行了实验（利用已知安全漏洞，组织ROP payload攻击这三种应用），实验结果表明该方法能够有效缓解ROP攻击。同时，该方法的性能开销低于~4%。

Yueqiang Chen等人设计了一种与kBouncer类似的方法，称作ROPecker[11]，也是利用LBR捕获程序控制流的方式进行ROP攻击监测。但不同之处在于判断是否遭受ROP攻击的逻辑和触发监测的时机。1、判断逻辑：在运行时检测过去（利用LBR）和未来的执行流（模拟执行）中是否存在长gadget链（5个比较短的gadget），若存在，则认为这是一次ROP攻击。Gadget信息是通过静态分析二进制程序和共享库得到的。2、运行时监测是事件驱动的，具体时机是调用敏感系统调用和执行流跳出滑动窗口触发异常。ROPecker设计了一个滑动窗口，因为代码本身具有时间和空间的局部性，但是gadget链却是散列的，利用这一特性，系统保证该窗口内的gadget数目不足以构成一次ROP攻击，窗口内的代码设置可执行权限，窗口外的代码不可执行，当执行流跳出滑动窗口时，便会触发异常，进行运行时检测。该方法利用代码本身具有的时间和空间局部性，针对gadget链是散列的前提，提出了滑动窗口机制，使用事件驱动的检测方法，具有较高的准确性和高效性。为了验证该方法的安全性，ROPecker选取了有栈溢出的真实世界应用（Linux Hex-editer）进行攻防演练。实验结果证明，ROPecker能够有效的阻止ROP攻击。同时，SPEC

CPU2006 benchmark显示了该方法的开销非常低（~2%）。

Yubin Xia等人设计的CFIMon[12]，也是采用性能计数器来捕获程序执行流，并进行合法性判断。但他们采用的是BTB（Branch Trace Buffer），来捕获受保护程序运行过程中所有跳转指令的信息。BTB与LBR不同之处在于，BTB可以把程序整个执行过程中所有的跳转指令的历史信息都记录下来，LBR只能记录16条。但是BTB需要CPU向指定的一个缓冲区内写入跳转信息，当缓冲区满时，CPU会触发异常交给操作系统处理（将缓冲区内容写入文件中），LBR是循环的寄存器。使用BTB的程序性能明显比LBR性能低。CFIMon检查BTB的时机在两个阶段：一是当缓冲区满时，操作系统将所有历史信息写入另一个进程，由另一个进程进行合法性判断；二是当受保护进程执行敏感系统调用时，另一个进程也进行历史信息的合法性判断。合法性判断主要检查间接控制转移的跳转目标是合法目标集合内。如果所有间接控制转移的历史跳转目标在合法目标集合中，认为当前受保护进程没有收到攻击；如果有至少一个间接控制转移的历史跳转目标在合法目标集合中，那么认为受保护进程受到攻击。合法目标的集合是在线下通过静态分析获得的，并且存储在检查进程中。

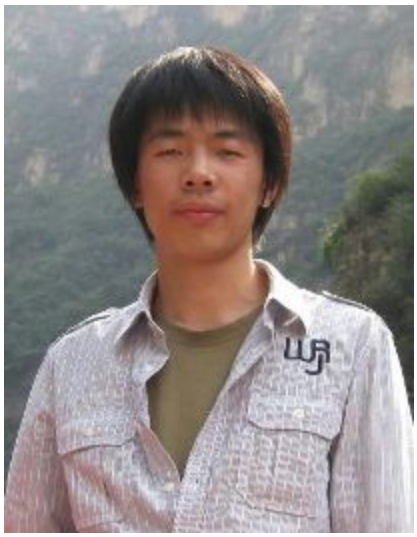
3、总结与展望

大多数的攻击都依赖于某种形式的控制劫持来重定向程序执行，CFI（Control Flow Integrity）是一种运行时强制执行的技术，用以抵御代码注入、代码重用攻击，也不易受信息泄露攻击。CFI在运行时强制执行预期的控制流转移，不允许执行应用程序控制流图（Control Flow Graph, CFG）中未出现的转移。精确的CFI会引入大量开销，这一点激励了更为实际的、粗粒度的CFI变体的发展；粗粒度CFI通过放宽限制条件来降低开销，却也给攻击者提供足够的利用空间。另外，CFI的作用也仅仅是用于阻止攻击者对控制流的劫持，有研究表明（如Control-Flow Bending，CFB[13] ），攻击者可以仅仅通过控制函数调用的参数，完成攻击者的目的，如执行任意代码、执行受限制的代码、和信息泄露。

从2005年CFI技术被提出，CFI技术已经历经了10多年的发展。近两年涌现出的大量CFI相关的研究工作，极大推进了CFI技术的发展。但是，CFI技术未来仍有很大的发展空间，仍然需要研究人员继续深入研究，实现一种高可靠、低开销的CFI技术。

作者

武成岗，中国科学院计算技术研究所正高级工程师、博士生导师，主要研究领域：动态编译、软件安全、程序分析等。



李建军，中国科学院计算技术研究所副研究员，主要研究领域：动态程序分析、软件安全保障等。

参考文献

[1]. Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2005. Control-flow integrity. In*Proceedings of the 12th ACM conference on Computer and communications security* (CCS '05). ACM, New York, NY, USA, 340-353.

[2]. Chao Zhang, Tao Wei, Zhaofeng Chen, Lei Duan, Laszlo Szekeres, Stephen McCamant, Dawn Song, and Wei Zou. 2013. Practical Control Flow Integrity and Randomization for Binary Executables. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy* (SP '13). IEEE Computer Society, Washington, DC, USA, 559-573. DOI=http://dx.doi.org/10.1109/SP.2013.44

[3]. Mingwei Zhang and R. Sekar. 2013. Control flow integrity for COTS binaries. In *Proceedings of the 22nd USENIX conference on Security* (SEC'13). USENIX Association, Berkeley, CA, USA, 337-352.

[4]. Enes Göktas, Elias Athanasopoulos, Herbert Bos, and Georgios Portokalidis. 2014. Out of Control: Overcoming Control-Flow Integrity. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy* (SP '14). IEEE Computer Society, Washington, DC, USA, 575-589.

[5]. Ali Jose Mashtizadeh, Andrea Bittau, Dan Boneh, and David Mazières. 2015. CCFI: Cryptographically Enforced Control Flow Integrity. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (CCS '15). ACM, New York, NY, USA, 941-951.

[6]. Lucas Davi, Ahmad-Reza Sadeghi, Daniel Lehmann, and Fabian Monrose. 2014. Stitching the gadgets: on the ineffectiveness of coarse-grained control-flow integrity protection. In *Proceedings of the 23rd USENIX conference on Security Symposium* (SEC'14). USENIX Association, Berkeley, CA, USA, 401-416.

[7]. Mauro Conti, Stephen Crane, Lucas Davi, Michael Franz, Per Larsen, Marco Negro, Christopher Liebchen, Mohaned Qunaibit, and Ahmad-Reza Sadeghi. 2015. Losing Control: On the Effectiveness of Control-Flow Integrity under Stack Attacks. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, 952-963.

[8]. Victor van der Veen, Dennis Andriesse, Enes Göktaş, Ben Gras, Lionel Sambuc, Asia Slowinska, Herbert Bos, and Cristiano Giuffrida. 2015. Practical Context-Sensitive CFI. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, 927-940.

[9]. John Criswell, Nathan Dautenhahn, and Vikram Adve. 2014. KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy* (SP '14). IEEE Computer Society, Washington, DC, USA, 292-307.

[10]. Vasilis Pappas, Michalis Polychronakis, and Angelos D. Keromytis. 2013. Transparent ROP exploit mitigation using indirect branch tracing. In Proceedings of the 22nd USENIX conference on Security (SEC'13). USENIX Association, Berkeley, CA, USA, 447-462.

[11]. Yueqiang Cheng, Zongwei Zhou, Miao Yu, Xuhua Ding and Robert H. Deng. 2014. ROPecker: A Generic and Practical Approach for Defending Against ROP Attacks. In *Proceedings of* the 2014 Network and Distributed System Security Symposium (NDSS '14).

[12]. Yubin Xia, Yutao Liu, Haibo Chen, and Binyu Zang. 2012. CFIMon: Detecting violation of control flow integrity using performance counters. In *Proceedings of the 2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (DSN '12). IEEE Computer Society, Washington, DC, USA, 1-12.

[13]. Carlini N, Barresi A, Payer M, Wagner D and Gross TR. 2015. Control-Flow Bending: On the Effectiveness of Control-Flow Integrity. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., USA