启发式检测之外,还对触发检测的时机进行了有效选择——基于敏感系统调用触发异常检测,从而很好地控制了性能开销.但这些代码重用攻击检测方案仅针对大多数通用的攻击有效,对于针对性攻击,基于启发式的检测方法还是存在被绕过的可能,例如,Isomeron,Size Does Matter 等研究中均给出了对此类启发式检测方法的bypass 实例[74,79].

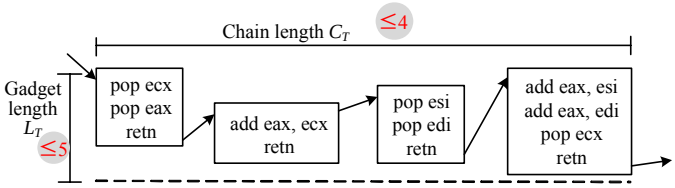


Fig.6 A heuristic detection strategy for code reuse attacks

图 6 基于启发式的代码重用攻击检测方法

3.2.2 栈异常

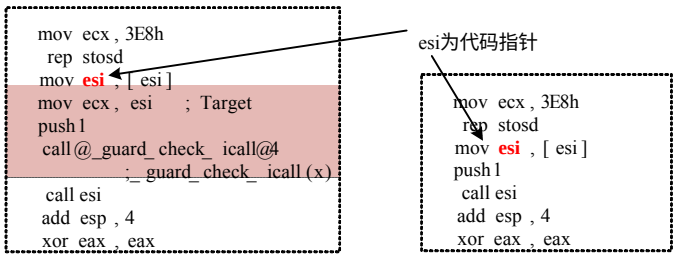
线程栈在程序函数调用过程中至关重要——保存了函数调用返回地址.利用线程栈的栈式结构,可以很容易地在栈上布局数据和程序跳转目标,实现 gadget 和函数的连续调用.为了方便布局攻击数据、尽可能地减小栈的破坏程度,攻击者通常会使用 stack pivot 指令将栈帧切换到一个位于堆上的伪造栈(forked stack)上.根据代码重用攻击过程中对栈或伪造栈的严重依赖性以及栈帧的异常切换,梁玉等人提出的 S-Tracker[80]和 Prakash[81]等人从栈完整性异常的角度对代码重用攻击进行检测.S-Tracker 对栈关键要素 esp,ebp 等进行了完整性定义,限制 esp 等指针的范围,在敏感行为发生时,触发对栈的检测;Prakash 等则是从代码重用中类似于 stack pivot 的关键 gadget 入手,使用 instrument 的方式对其修改,使得这类指令被调用时触发检测机制,通过判断栈指针完整性,实现对代码重用攻击的检测.

3.3 典型防护技术的实践与应用

对于防护方案,从理论、原型的提出到最终实际部署、应用是一个相对漫长的过程.近两年来,以 CFG 和 CET 为代表的基于控制流完整性的平台级防护方案在工业界得到应用和实践.

3.3.1 CFG

CFG(control flow guard)是微软对控制流完整性的一个平台级实现,通过代码编译和程序运行时(runtime)相结合的方式实现对 call 的间接跳转目标的限制,从而有效缓解包括代码重用在内的攻击方式.具体来说,微软在 Visual Studio 2015 中引入了 CFG,并通过代码生成选项"/guard:cf"开启对间接跳转指令的插桩.图 7 给出了插桩前后的对比.自 Windows 10 开始,微软正式引入了操作系统级的 CFG 支持.图 7(a)中额外调用的 _guard_check_ icall 在支持 CFG 的操作系统上,会通过调用 ntdll!LdrpVaildateUserCallTarget 对跳转目标进行验证;在不支持 CFG 的系统中是一个空调用,不引发额外的操作.操作系统通过位图 CFGBitmap 存储了当前调用的有效跳转目标集合,在 ntdll!LdrpValidateUserCallTarget 当中首先获取 CFGBitmap 对象,然后索引到对应的值,从而判断当前跳转是否有效:如果无效,则终止进程[82,83].



(a) 开启cfg (b) 未开启cfg

Fig.7 Comparison of code blocks generated by Visual Studio with CFG on and off

图 7 Visual Studio 开启 CFG 前后生成的代码对比

3.3.2 CET

2016 年 6 月,Intel 发布了称为 Control-flow Enforcement Technology(CET)的技术预览白皮书,正式公开了其准备推出的主要用于防护 ROP,COP/JOP(call/jump oriented programming)的芯片级解决方案.该方案的核心思想源自学术界早先提出的 shadow stack,即在系统中的特定内存区域构造一个专门用来存储间接跳转数据的栈[47,84].具体来说,当一个子函数被调用时,像传统方式一样,其返回地址将会被保存到线程栈上;同时也会被保存到 shadow stack 上;在程序后续执行中若遇到返回指令,则处理器需要保证线程栈上的返回地址与影子栈中的地址相匹配.如果二者不匹配,处理器则会抛出异常,使得操作系统能够捕获该异常并停止程序执行.

在 CET 技术中,Intel 从 CPU 级别对 shadow stack 提供了安全特性支持.Shadow stack 仅用于控制流转移操作,其中保存了函数返回地址等,与一般的数据栈相对独立.为了防止 shadow stack 被篡改,Intel 通过 CPU 的内存管理单元 MMU 在页表保护中提供了新的扩展属性,使得页内存除了具备读、写、执行保护外,还可被标记为 shadow stack 页.被标记为 shadow stack 的内存页,普通软件和指令无法直接对其进行写操作,只有通过特定的控制流转移指令和 Intel 提供的专门的 shadow stack 操作指令,才能实现对 shadow stack 的写操作,从而从硬件体系结构提供了 shadow stack 这一安全特性.

此外,Intel 还提供了一个新寄存器 SSP(shadow stack pointer).该寄存器始终指向 shadow stack 栈顶.当 CET 功能打开时,执行函数返回指令时,CPU 将自动地从 SSP 寄存器指向的 shadow stack 栈顶处取保存的返回地址,并与普通栈中的返回地址进行对比:二者如果不一致,CPU 则抛出异常,操作系统捕获该异常后进行后续处理.

和 CFG 一样,CET 技术也需要通过平台支撑才能完全实现防护效果.Intel 为 CET 技术提供了新指令 ENDBRANCH,该指令可用于标记合法的间接跳转目标.只有开发者使用支持 CET 技术的编译器来生成目标程序,才会使得目标程序中的返回地址等合法间接跳转目标被 ENDBRANCH 标记.在程序运行时,CPU 内部构建了一个状态机来跟踪 call/jmp 等间接跳转指令:当执行到 call/jmp 时,状态机由 IDLE 进入 WAIT_FOR_ENDBRANCH 状态.该状态下,其下一条指令必须是 ENDBRANCH:如果不是 ENDBRANCH,则说明跳转目标是非法的,触发 CET 保护,抛出异常;如果是 ENDBRANCH,则状态机再次进入 IDLE 状态.通过上述方式,实现对基于 COP/JOP 等代码重用方式的防护.

完善的防护方案需要构建从整个生态系统出发,尽可能地减少攻击面,提升安全性.CFG 和 CET 的设计和实施过程中,都是从底层硬件支撑、系统层程序执行环境保障、编译器层程序代码生成等环节进行了安全性和兼容性考虑,使其可快速地、最大化地被实际部署.但是,考虑到提供 CET 的 Intel 芯片目前尚未发布,且攻防对抗是一个持续、演进的过程,新的攻击方法存在绕过 CET,CFG 这类防护策略的可能性,所以二进制代码重用技术仍然是软件安全领域所面临的一个重要威胁.

4 总结与展望

代码重用作为一种程序执行方式,以其灵活的代码组织方式、功能逻辑动态生成的特点,在过去被广泛用于漏洞攻击当中.严峻的网络空间安全形势使得代码重用技术的研究价值凸显,学术界、工业界的研究人员在近 10 年中,从攻击和防护两个方面开展了深入的研究.本文前面部分重点对已有研究的分析和总结,一方面探讨了代码重用技术在攻击过程中的关键要素、逻辑组织方式及其攻防博弈下的演变历程;另一方面,从防护和缓解攻击的角度分析了已有防御技术的思路,如基于控制流完整性的防护思路和基于随机化的防护思路.此外,文中也对最新的来自工业界防护方法 CFG,CET 进行了分析,为研究实用性更强的防护技术提供了启发.

代码重用技术除了用于攻击之外,这种动态生成程序逻辑的程序执行方式亦可用于代码混淆,尤其是对关键功能、关键逻辑的代码混淆和隐藏[85,86].例如,在程序运行过程中,通过服务器推送的特定数据输入,动态生成所需要的功能逻辑,实现关键功能执行.这种方式的一个关键是在正常程序中构建一个风险可控的“漏洞”,使得程序控制流有机会被隐藏的功能逻辑使用.因此,研究如何利用基于二进制代码重用的程序执行方式进行代码逻辑混淆、代码隐写等应用,是一项可探索的工作.

随着防御技术的不断演进,代码重用攻击不局限于编译、发布的程序,可能更加倾向于动态生成代码的重

用.目前,针对 JIT 等方式动态生成的二进制代码的防护相对薄弱,这将成为攻击者的新的攻击面.因此,针对动态生成代码的攻击防护,也就是代码重用攻击对抗中的一个新的战场.

此外,从控制流完整性和随机化的角度研究代码重用攻击防御方法仍将继续.在控制流完整性防护方面,一方面是完善已有的防御方法,如 CET,CFG,使其更加完备;另一方面,继续探索如何高效地实现更细粒度的 CFI.对于随机化能力增强的研究可从 3 个方面展开:(1) 引入新的随机化机制,弥补现有随机化机制中的不足;(2) 对内存中不同的数据对象进行不同程度的随机化,除了使其具备防范代码重用攻击之外,还可提升其整体安全性;(3) 提高随机化的熵,增强随机化的效果.此外,邬江兴院士提出的拟态计算机理论则是从整个计算机体系结构出发,提高运行环境或执行机构的不确定性[87].对于该理论,从攻击者的角度来看,一类与运行环境、执行程序实体关系紧密的攻击方式将可能失效.因此,研究新的计算机体系、从根源上解决代码重用等众多内存漏洞攻击,也是可供探索的方向.

典型代码重用攻击需要配合内存信息泄漏,而这其中的信息泄露既包括程序漏洞引发的内存任意读缺陷,也包括由软件、硬件设计缺陷导致的侧信道信息泄露.因此,如何从硬件、系统、软件等层面有效防止程序运行信息泄漏、防止内存中敏感数据泄漏,是近期研究的一个方向.在数据中心、云环境中,针对硬件特性引发的侧信道信息泄露的研究也值得关注.

二进制代码重用在当前的计算机系统体系结构和软件生态环境下暂时无法消除,基于代码重用的攻防博弈仍将继续.因此,无论是为了在网络空间安全博弈中占据主动权,还是出于防护的目的,都有必要对二进制代码重用技术做进一步的研究.

References:

[1] Mei H, Wang QX, Zhang L, Wang J. Software analysis: A road map. Chinese Journal of Computers, 2009,32(9):1697−1710 (in Chinese with English abstract). http://cjc.ict.ac.cn/quanwenjiansuo/2009-9/mh.pdf [doi: 10.3724/SP.J.1016.2009.01697]

[2] Chen X, Gu Q, Liu SW, Liu SL, Ni C. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016, 27(1):1−25 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[3] Wang T. Research on binary-executable-oriented software vulnerability detection [Ph.D. Thesis]. Beijing: Peking University, 2011 (in Chinese with English abstract). http://d.wanfangdata.com.cn/Thesis/Y2024928

[4] Zhang HG, Han WB, Lai XJ, Lin DD, Ma JF, Li JH. Survey on cyberspace security. Scientica Sinica Informationis, 2016,46(2): 125−164 (in Chinese). [doi: 10.1360/N112015-00176]

[5] Zhang HG, Han WB, Lai XJ, Lin DD, Ma JF, Li JH. Survey on cyberspace security. Science China: Information Sciences, 2015, 58(11):1−43. [doi: 10.1007/s11432-015-5433-4]

[6] Liang Y. Research on key techniques of software binary code reuse [Ph.D. Thesis]. Wuhan: Wuhan University, 2016 (in Chinese with English abstract).

[7] Spafford EH. The Internet worm program: An analysis. ACM SIGCOMM Computer Communication Review, 1989,19(1):17−57. [doi: 10.1145/ 66093.66095]

[8] Wei Q, Wei T, Wang JJ. The evolution of exploitation and exploit mitigation. Journal of Tsinghua University (Science and Technology), 2011,51(10):1274−1280 (in Chinese with English abstract). [doi: 10.16511/j.cnki.qhdxxb.2011.10.015]

[9] APPLE. iOS security guide. 2015. https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[10] Miller C, Blazakis D, Daizovi D, Esser S, Lozz V, Weinmann RP. iOS Hacker's Handbook. Indianapolis: Wiley, 2012.

[11] Schwartz EJ, Avgerinos T, Brumley D. *Q*: Exploit hardening made easy. In: Proc. of the 20th USENIX Conf. on Security. Berkeley: USENIX Association, 2011. 25−25. http://dl.acm.org/citation.cfm?id=2028067.2028092

[12] Wkipedia. Return-Oriented programming. 2015. https://en.wikipedia.org/w/index.php?title=Return-oriented_programming&oldid=669727753

[13] Shacham H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In: Proc. of the 14th ACM Conf. on Computer and Communications Security (CCS 2007). New York: ACM Press, 2007. 552−561. [doi: 10.1145/1315245.1315313]

[14] Bletsch T, Jiang X, Freeh VW, Liang ZK. Jump-Oriented programming: A new class of code-reuse attack. In: Proc. of the 6th ACM Symp. on Information, Computer and Communications Security (ASIACCS 2011). New York: ACM Press, 2011. 30–40. [doi: 10.1145/1966913.1966919]

[15] Checkoway S, Davi L, Dmitrienko A, Sadeghi AR, Shacham H, Winandy M. Return-Oriented programming without returns. In: Proc. of the 17th ACM Conf. on Computer and Communications Security (CCS 2010). New York: ACM Press, 2010. 559–572. [doi: 10.1145/1866307.1866370]

[16] Salwan J. ROPgadget. 2015. https://github.com/JonathanSalwan/ROPgadget

[17] Le L. Payload already inside: Data re-use for ROP exploits. In: Proc. of the BlackHat USA 2010. Las Vegas, 2010. https://media.blackhat.com/bh-us-10/whitepapers/Le/BlackHat-USA-2010-Le-Paper-Payload-already-inside-data-reuse-for-ROP-exploits-wp.pdf

[18] Ropshell. Free online ROP gadgets search. 2016. http://2www.ropshell.com/

[19] Chen P, Xiao X, Bing M, Li X, Shen X, Yin X. Automatic construction of jump-oriented programming shellcode (on the x86). In: Proc. of the 6th ACM Symp. on Information, Computer and Communications Security (ASIACCS 2011). New York: ACM Press, 2011. 20–29. [doi: 10.1145/1966913.1966918]

[20] Davi L, Alexandra D, Sadeghi AR, Winandy M. Return-Oriented programming without returns on ARM. Bochum: Ruhr University Bochum, 2010. http://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/ROP-without-Returns-on-ARM.pdf

[21] Kornau T. Return oriented programming for the ARM architecture [MS. Thesis]. Bochum: Ruhr-Universität Bochum, 2010. https://www.zynamics.com/downloads/kornau-tim--diplomarbeit--rop.pdf

[22] Buchanan E, Roemer R, Savage S, Shacham H. Return-Oriented programming: Exploits without code injection. In: Proc. of the BlackHat USA 2008. Las Vegas, 2008. https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf

[23] Roemer RG. Finding the bad in good code: Automated return-oriented programming exploit discovery [MS. Thesis]. San Diego: University of California, 2009. http://www.cs.ucsd.edu/~rroemer/doc/thesis.pdf

[24] Dullien T, Kornau T, Weinmann RP. A framework for automated architecture-independent gadget search. In: Proc. of the 4th USENIX Conf. on Offensive Technologies (WOOT 2010). Berkeley: USENIX Association, 2010. 1−10. http://www.cs.ucsd.edu/~rroemer/doc/thesis.pdf

[25] Flanagan C, Saxe JB. Avoiding exponential explosion: Generating compact verification conditions. In: Proc. of the 28th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. New York: ACM Press, 2001. 193–205. [doi: 10.1145/360204.360220]

[26] Solar D. Bugtraq: Getting around non-executable stack (and fix). 1997. http://seclists.org/bugtraq/1997/Aug/63

[27] Mcdonald J. Defeating solaris/SPARC non-executable stack protection. 1999. https://www.thc.org/root/docs/exploit_writing/sol-ne-stack.html

[28] Nergal. The advanced return-into-lib(c) exploits (PaX case study). Phrack Magazine, 2001,58(4). http://phrack.org/issues/58/4.html

[29] Krahmerk S. x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique. 2005. http://users.suse.com/~krahmer/no-nx.pdf

[30] Buchanan E, Roemer R, Shacham H, Savage S. When good instructions go bad: Generalizing return-oriented programming to RISC. In: Proc. of the 15th ACM Conf. on Computer and Communications Security (CCS 2008). New York: ACM Press, 2008. 27–38. [doi: 10.1145/1455770.1455776]

[31] Qian Y. ROP attack and defense technology based on ARM [MS. Thesis]. Shanghai: Shanghai Jiaotong University, 2012 (in Chinese with English abstract). http://cdmd.cnki.com.cn/Article/CDMD-10248-1013022062.htm

[32] Xing T, Chen P, Ding WB. BIOP: Automatic construction of enhanced ROP attack. Chinese Journal of Computers, 2014,37(5): 1111−1123 (in Chinese with English abstract). http://d.wanfangdata.com.cn/Periodical/jsjxb201405012 [doi: 10.3724/SP.J.1016.2014.01111]

[33] Snow KZ, Monrose F, Davi L, Dmitrienko A, Liebchen C, Sadeghi A. Just-in-Time code reuse: On the effectiveness of fine-grained address space layout randomization. In: Proc. of the 2013 IEEE Symp. on Security and Privacy (SP). 2013. 574–588. [doi: 10.1109/SP.2013.45]

[34] Snow KZ, Davi L. Just in time code reuse. In: Proc. of the Blackhat USA 2013. Las Vegas, 2013. http://media.blackhat.com/us-13/US-13-Snow-Just-In-Time-Code-Reuse-Slides.pdf

[35] Bittau A, Belay A, Mashtizadeh A, Mazières D, Boneh D. Hacking blind. In: Proc. of the 2014 IEEE Symp. on Security and Privacy (SP). 2014. 227−42. [doi: 10.1109/SP.2014.22]

[36] Athanasakis M, Elias A, Michalis P, Georgios P, Sotiris I. The devil is in the constants: Bypassing defenses in browser JIT engines. In: Proc. of the 2015 Network and Distributed System Security Symp. (NDSS 2015). 2015. [doi: 10.14722/ndss.2015.23209]

[37] Qian Y, Wang YJ, Xue Z. ROP attack and defense technology based on ARM. Information Security and Communications Privacy, 2012,(10):75−77 (in Chinese with English abstract). [doi: 10.3969/j.issn.1009-8054.2012.10.036]

[38] Xia Y, Liu Y, Chen H, Zang B. CFIMon: Detecting violation of control flow integrity using performance counters. In: Proc. of the IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN 2012). Boston: IEEE, 2012. 1−12. [doi: 10.1109/DSN.2012.6263958]

[39] Carlini N, Wagner D. ROP is still dangerous: Breaking modern defenses. In: Proc. of the 23rd USENIX Security Symp. (USENIX Security 2014). San Diego: USENIX Association, 2014. 385−399. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/carlini

[40] Carlini N, Barresi A, Mayer M, Wagner D, Gross TR. Control-Flow bending: On the effectiveness of control-flow integrity. In: Proc. of the 24th USENIX Security Symp. (USENIX Security 2015). Washington: USENIX Association, 2015. 161−176. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/carlini

[41] Cheng YQ, Zhou ZW, Yu M, Ding XH, Deng RH. ROPecker: A generic and practical approach for defending against ROP attacks. In: Proc. of the 2014 Network and Distributed System Security Symp. (NDSS 2014). 2014. [doi: 10.14722/ndss.2014.23156]

[42] Qiao R, Zhang MW, Sekar R. A principled approach for ROP defense. In: Proc. of the 31st Annual Computer Security Applications Conf. (ACSAC 2015). New York: ACM Press, 2015. 101−110. [doi: 10.1145/2818000.2818021]

[43] Szekeres, L, Payer M, Wei T, Song D. SoK: Eternal war in memory. In: Proc. of the 2013 IEEE Symp. on Security and Privacy (SP). San Francisco: IEEE Computer Society, 2013. 48−62. [doi: 10.1109/SP.2013.13]

[44] Schuster F, Tendyck T, Liebchen C, Davi L, Sadeghi AR, Holz T. Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications. In: Proc. of the 2015 IEEE Symp. on Security and Privacy (SP). 2015. 745−62. [doi: 10.1109/SP.2015.51]

[45] Liang Y, Peng GJ, Luo Y, Zhang HG. Mitigating ROP attacks via ARM-specific in-place instruction randomization. China Communications, 2016,13(9):208−826. [doi: 10.1109/CC.2016.7582313]

[46] Abadi M, Budiu M, Erlingsson Ú, Ligatti J. Control-Flow integrity. In: Proc. of the 12th ACM Conf. on Computer and Communications Security (CCS 2005). New York: ACM Press, 2005. 340−353. [doi: 10.1145/1102120.1102165]

[47] Abadi M, Budiu M, Erlingsson Ú, Ligatti J. Control-Flow integrity principles, implementations, and applications. ACM Trans. on Information and System Security, 2009,13(1):4:1−4:40. [doi: 10.1145/1609956.1609960]

[48] Wu CG, Li JJ. The evolution of control flow integrity. 2016 (in Chinese). http://www.inforsec.org/wp/?p=495

[49] Abadi M, Budiu M, Erlingsson Ú, Ligatti J. A theory of secure control flow. In: Lau KK, Banach R, eds. Proc. of 7th Int'l Conf. on Formal Engineering Methods (ICFEM 2005). Berlin, Heidelberg: Springer-Verlag, 2005. 11−24. [doi: 10.1007/11576280_9]

[50] Arden O, George MD, Liu J, Vikram K, Askarov A, Myers AC. Sharing mobile code securely with information flow control. In: Proc. of the 2012 IEEE Symp. on Security and Privacy (SP). 2012. 191−205. [doi: 10.1109/SP.2012.22]

[51] Zhang C, Wei T, Chen ZF, Duan L, Szekeres L, McCamant S, Song D, Zou W. Practical control flow integrity and randomization for binary executables. In: Proc. of the 2013 IEEE Symp. on Security and Privacy (SP). 2013. 559−73. [doi: 10.1109/SP.2013.44]

[52] Zhang M, Sekar R. Control flow integrity for COTS binaries. In: Proc. of the 22nd USENIX Security Symp. (USENIX Security 2013). Washington: USENIX Association, 2013. 337−352. [doi: 10.1145/2818000.2818016]

[53] Goktas E, Athanasopoulos E, Bos H, Portokalidis G. Out of control: Overcoming control-flow integrity. In: Proc. of the 2014 IEEE Symp. on Security and Privacy (SP). 2014. 575−89. [doi: 10.1109/SP.2014.43]

[54] Niu B, Tan G. Modular control-flow integrity. In: Proc. of the 35th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2014). New York: ACM Press, 2014. 577−587. [doi: 10.1145/2594291.2594295]

[55] Niu B, Tan G. RockJIT: Securing just-in-time compilation using modular control-flow integrity. In: Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2014). New York: ACM Press, 2014. 1317−28. [doi: 10.1145/2660267.2660281]

[56] Mohan V, Larsen P, Brunthaler S, Hamlen K, Franz M. Opaque control-flow integrity. In: Proc. of the 2015 Network and Distributed System Security Symp. (NDSS 2015). 2015. [doi: 10.14722/ndss.2015.23271]

[57] Davi L, Koeberl P, Sadeghi AR. Hardware-Assisted fine-grained control-flow integrity: Towards efficient protection of embedded systems against software exploitation. In: Proc. of the 51st Annual Design Automation Conf. (DAC 2014). New York: ACM Press, 2014. 133:1−133:6. [doi: 10.1145/2593069.2596656]

[58] Pewny J, Holz T. Control-Flow restrictor: Compiler-based CFI for iOS. In: Proc. of the 29th Annual Computer Security Applications Conf. (ACSAC 2013). New York: ACM Press, 2013. 309−318. [doi: 10.1145/2523649.2523674]

[59] Chiueh TC, Hsu FH. RAD: A compile-time solution to buffer overflow attacks. In: Proc. of the 21st Int'l Conf. on Distributed Computing Systems. Mesa: IEEE, 2001. 409−417. [doi: 10.1109/ICDSC.2001.918971]

[60] Davi L, Sadeghi AR, Winandy M. ROPdefender: A detection tool to defend against return-oriented programming attacks. In: Proc. of the 6th ACM Symp. on Information, Computer and Communications Security (ASIACCS 2011). New York: ACM Press, 2011. 40−51. [doi: 10.1145/1966913.1966920]

[61] Frantzen M, Shuey M. StackGhost: Hardware facilitated stack protection—Vol.10. In: Proc. of the 10th Conf. on USENIX Security Symp. (SSYM 2001). Berkeley: USENIX Association, 2001. 5−5. http://dl.acm.org/citation.cfm?id=1267612.1267617

[62] Sinnadurai S, Zhao Q, Wong W. Transparent runtime shadow stack: Protection against malicious return address modifications. 2014. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.120.5702

[63] Iintel. Pin—A dynamic binary instrumentation tool. https://software.intel.com/en-us/articles/pintool

[64] PAX team. Address space layout raondomization (ASLR). 2001. https://pax.grsecurity.net/docs/aslr.txt

[65] Ubuntu. Ubuntu security features—Address space layout randomisation (ASLR). Ubuntu Wiki: 2016, https://wiki.ubuntu.com/Security/Features#Address_Space_Layout_Randomisation_.28ASLR.29

[66] Schulz P. Android security analysis challenge: Tampering dalvik bytecode during runtime. Bluebox Security, 2013, https://bluebox.com/android-security-analysis-challenge-tampering-dalvik-bytecode-during-runtime/

[67] Liang Y, Ma X, Wu D, Tang X, Gao D, Peng G, Jia C, Zhang H. Stack layout randomization with minimal rewriting of Android binaries. In: Kwon S, Yun A, eds. Proc. of the Information Security and Cryptology (ICISC 2015). LNCS 9558, Cham: Springer Int'l Publishing, 2016. 229−45. [doi: 10.1007/978-3-319-30840-1_15]

[68] Lee B, Lu L, Wang T, Kim T, Lee W. From zygote to morula: Fortifying weakened ASLR on Android. In: Proc. of the 2014 IEEE Symp. on Security and Privacy (SP). 2014. 424−439. [doi: 10.1109/SP.2014.34]

[69] Hiser J, Nguyen-Tuong A, Co M, Hall M, Davidson JW. ILR: Where'd my gadgets go? In: Proc. of the 2012 IEEE Symp. on Security and Privacy (SP). 2012. 571−85. [doi: 10.1109/SP.2012.39]

[70] Wartell R, Mohan V, Hamlen KW, Lin Z. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. In: Proc. of the 2012 ACM Conf. on Computer and Communications Security (CCS 2012). New York: ACM Press, 2012. 157−168. [doi: 10.1145/2382196.2382216]

[71] Pappas V, Polychronakis M, Keromytis AD. Smashing the gadgets: Hindering return-oriented programming using in-place code randomization. In: Proc. of the 2012 IEEE Symp. on Security and Privacy (SP). 2012. 601−615. [doi: 10.1109/SP.2012.41]

[72] Chen Y. A survey of address space layout randomization (ASLR) enforcement. 2016 (in Chinese). http://www.inforsec.org/wp/?p=1009

[73] Backes M, Nürnberger S. Oxymoron: Making fine-grained memory randomization practical by allowing code sharing. In: Proc. of the 23rd USENIX Security Symp. (USENIX Security 14). San Diego: USENIX Association, 2014. 433−447. https://www.usenix.org/node/184466

[74] Davi L, Christopher L, Sadeghi AR, Snow KZ, Monrose F. Isomeron: Code randomization resilient to (just-in-time) return-oriented programming. In: Proc. of the 2015 Network and Distributed System Security Symp. (NDSS 2015). 2015. [doi: 10.14722/ndss.2015.23262]

[75] Lu K, Nurnberger S, Backes M, Lee W. How to make ASLR win the clone wars: Runtime re-randomization. In: Proc. of the 2016 Network and Distributed System Security Symp. (NDSS 2016). 2016. http://www.cc.gatech.edu/~klu38/publications/runtimeaslr-ndss16.pdf

[76] Backes M, Holz T, Kollenda B, Koppe P, Nürnberger S, Pewny J. You can run but you can't read: Preventing disclosure exploits in executable code. In: Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security (CCS 2014). New York: ACM Press, 2014. 1342−1353. [doi: 10.1145/2660267.2660378]

[77] Crane S, Liebchen C, Homescu A, Davi L, Larsen P, Sadeghi AR, Brunthaler S, Franz M. Readactor: Practical code randomization resilient to memory disclosure. In: Proc. of the 2015 IEEE Symp. on Security and Privacy (SP). 2015. 763−80. [doi: 10.1109/SP.2015.52]

[78] Pappas V, Polychronakis M, Keromytis AD. Transparent ROP exploit mitigation using indirect branch tracing. In: Proc. of the 22nd USENIX Security Symp. (USENIX Security 2013). Berkeley: USENIX Association, 2013. 447−462. http://dl.acm.org/citation.cfm?id=2534766.2534805

[79] Göktaş E, Athanasopoulos E, Polychronakis M, Bos H, Portokalidis G. Size does matter: Why using gadget-chain length to prevent code-reuse attacks is hard. In: Proc. of the 23rd USENIX Security Symp. (USENIX Security 2014). San Diego: USENIX Association, 2014. 417−432. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/goktas

[80] Liang Y, Fu J, Peng G, Peng B. S-Tracker: Attribution of shellcode exploiting stack. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2014,42(11):39−46 (in Chinese with English abstract). [doi: 10.13245/j.hust.141108]

[81] Prakash A, Yin H. Defeating ROP through denial of stack pivot. In: Proc. of the 31st Annual Computer Security Applications Conf. (ACSAC 2015). New York: ACM Press, 2015. 111−120. [doi: 10.1145/2818000.2818023]

[82] Tang J. Exploring control flow guard in Windows 10. 2015. http://documents.trendmicro.com/assets/wp/exploring-control-flow-guard-in-windows10.pdf

[83] Tencent PC Manager. Security features of Windows 10: The enforcement of exection flow. 2015 (in Chinese). http://www.freebuf.com/articles/security-management/58373.html

[84] Dang T, Maniatis P, Wagner D. The performance cost of shadow stacks and stack canaries. In: Proc. of the 10th ACM Symp. on Information, Computer and Communications Security (ASIA CCS 2015). New York: ACM Press, 2015. 555−566. [doi: 10.1145/2714576.2714635]

[85] Lu K, Xiong S, Gao D. RopSteg: Program steganography with return oriented programming. In: Proc. of the 4th ACM Conf. on Data and Application Security and Privacy (CODASPY 2014). New York: ACM Press, 2014. 265−272. [doi: 10.1145/2557547.2557572]

[86] Tang X, Liang Y, Ma X, Lin Y, Gao D. On the effectiveness of code-reuse based Android application obfuscation. In: Hong S, Park J, eds. Proc. of the Information Security and Cryptology (ICISC 2016). LNCS 10157, Cham: Springer-Verlag, 2017. 333−349. [doi: 10.1007/978-3-319-53177-9_18]

[87] Wu J. Mimic defense in cyberspace. Secrecy Science and Technology, 2014,(10):4−9 (in Chinese with English abstract). http://www.cnki.com.cn/Article/CJFDTotal-BMKJ201410001.htm

附中文参考文献:

[1] 梅宏,王千祥,张路,王戟.软件分析技术进展.计算机学报,2009,32(9):1697−1710. http://cjc.ict.ac.cn/quanwenjiansuo/2009-9/mh.pdf [doi: 10.3724/SP.J.1016.2009.01697]

[2] 陈翔,顾庆,刘望舒,刘树龙,倪超.静态软件缺陷预测方法研究.软件学报,2016,27(1):1−25. http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[3] 王铁磊.面向二进制程序的漏洞挖掘关键技术研究[博士学位论文].北京:北京大学,2011. http://d.wanfangdata.com.cn/Thesis/Y2024928

[4] 张焕国,韩文报,来学嘉,林东岱,马建峰,李建华.网络空间安全综述.中国科学:信息科学,2016,46(2):125−164. [doi: 10.1360/N112015-00176]

[6] 梁玉.软件二进制代码重用关键技术研究[博士学位论文].武汉:武汉大学,2016.

[8] 魏强,韦韬,王嘉捷.软件漏洞利用缓解及其对抗技术演化.清华大学学报:自然科学版,2011,51(10):1274−1280. [doi: 10.16511/j.cnki.qhdxxb.2011.10.015]

[31] 钱逸.基于 ARM 架构的 ROP 攻击与防御技术研究[硕士学位论文].上海:上海交通大学,2012. http://cdmd.cnki.com.cn/Article/CDMD-10248-1013022062.htm

[32] 邢骁,陈平,丁文彪,茅兵,谢立.BIOP:自动构造增强型 ROP 攻击.计算机学报,2014,37(5):1111−1123. http://d.wanfangdata.com.cn/Periodical/jsjxb201405012 [doi: 10.3724/SP.J.1016.2014.01111]

[37] 钱逸,王轶骏,薛质.基于 ARM 平台的 ROP 攻击及防御技术.信息安全与通信保密,2012,(10):75−77. [doi: 10.3969/j.issn.1009-8054.2012.10.036]

[48] 武成岗,李建军.控制流完整性的发展历程.2016. http://www.inforsec.org/wp/?p=495

[72] Chen Y.地址空间布局随机化(ASLR)增强研究综述.2016. http://www.inforsec.org/wp/?p=1009

[80] 梁玉,傅建明,彭国军,等.S-Tracker:基于栈异常的 shellcode 检测方法.华中科技大学学报(自然科学版),2014,42(11):39−46. [doi: 10.13245/j.hust.141108]

[83] 腾讯电脑管家.Win10 安全特性之执行流保护.2015. http://www.freebuf.com/articles/security-management/58373.html

[87] 邬江兴.网络空间拟态安全防御.保密科学技术,2014,(10):4−9. http://www.cnki.com.cn/Article/CJFDTotal-BMKJ201410001.htm

彭国军(1979－),男,湖北荆州人,博士,教授,CCF 专业会员,主要研究领域为恶意代码检测,可信软件.

张焕国(1945－),男,教授,博士生导师,CCF 高级会员,主要研究领域为信息安全,可信计算,密码学.

梁玉(1988－),男,博士,主要研究领域为系统安全,网络安全.

傅建明(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为系统安全,网络安全.