绕过栈保护(DEP)、地址随机化(ASLR)以及控制流完整性检测(CFI)[52−57]等.针对栈缓冲区溢出的控制流劫持ROP攻击可以绕过DEP和ASLR,同样处于卡内基·梅隆大学的Schwartz也开发了相应的自动利用系统Q[59],其收集目标程序中的Gadget,并通过面向Gadget的编程语言QooL自动构建ROP攻击[47−51].

新加坡国立大学的胡宏等人构造了一种数据流劫持攻击方法,其能够在不改变控制流的前提下,通过修改指针和数据指向,进而触发权限提升和实现内存数据窃取.他们还设计了一种数据流自动缝合技术FlowStitch[8],FlowStitch通过构造2D-DFG来寻找活跃的可用数据和指针,通过预先识别安全敏感数据来缩小缝合数据的选择范围.其能够识别的敏感数据包括系统调用参数、被配置文件污染的变量以及栈保护随机数(canary).FlowStitch将路径约束、影响约束和控制流完整性约束三者的合取式交给Z3求解器求解,进而获得触发数据流攻击的输入数据.胡宏等人在后续的研究中[58]证明了这种攻击的图灵完备性,也就是说,数据流攻击可以完成图灵机能完成的任何事情.

3 存在的问题和挑战

精准执行可达性分析应用大都采用双向符号分析和约束求解技术,这主要是因为其能够准确模拟一条路径的执行行为,进而保证了执行可达性分析结论的准确性.但是,该领域仍存在许多待解决的问题.

(1) 复杂约束的模型构建和关联分析缺乏有效的分析框架和理论指导.已有的自动漏洞利用方案大都针对栈缓冲区溢出,存在一定的局限性,且相应的内存建模技术还不成体系,更多的案例研究有待开展;

(2) 路径探测策略缺乏理论指导.对于定向符号执行中常用的最短路径优先等启发式策略,如何进行理论上的定量评估和分析以及是否存在更高效的探测策略尚未明确.对固定代码位置目标可达路径模式研究还较少,对此,模糊测试中关于路径迁移关系的研究思路[31,107]值得借鉴;

(3) 数据结构的识别和推理能力不足.路径敏感的指针别名分析问题、数据结构高级性质的逆向推理和归纳问题以及对全局变量操作的逆向推理问题等,都是亟待解决的;

(4) 动态控制流迁移逆向分析存在挑战.源代码中的动态控制流迁移表现为函数指针的动态赋值、类的动态实例化、反射以及同步操作等;在二进制分析中的动态控制流迁移则表现为间接跳转和中断.该困难集中显现在基于事件和消息的系统以及含有回调机制的系统中,包括操作系统和GUI程序;

(5) 路径爆炸问题仍然是多路径执行可达性搜索的最大障碍,现有的路径归纳和组合分析方法还不完善.现有的循环路径摘要算法局限在处理简单的周期性字符处理和整数运算上,无法处理指针数据结构以及带有函数调用的循环.对面向逆向分析的按需路径摘要技术以及综合高层模式和执行细节的分析技术还有待探索;

(6) 面向程序分析的SMT求解技术还存在提升空间.由于SMT公式的可满足问题是将执行可达性分析问题按照路径片段拆解得到的,而现有的技术所使用的求解器大都是相对通用的,因此存在进一步优化的可能.程序路径的经验知识的重要性已经在用机器学习改进求解技术的研究中有所显现[98];

(7) 人工API建模与验证的成本高于执行可达性分析带来的回报,这阻碍了对不同类型程序的扩展研究.目前,精准执行可达性分析应用研究的自选测试集中大都是一些交互简单的文件处理和网络服务程序,缺乏对操作系统内核、设备驱动、中间件、带有图形界面的程序以及Android等新平台上的精准执行可达性分析的研究;

(8) 并发程序和含有随机数程序的精准执行可达性分析仍存在调度策略组合爆炸和执行路径重现方面的挑战.

4 总结与展望

精准执行可达性分析是软件测试、软件调试和漏洞挖掘实践中的核心技术.本文从理论和应用两方面对精准执行可达性分析相关研究进行了跟踪、归纳和总结,对精准执行可达性分析方法和应用进行了分析和分类.虽然精准执行可达性分析在中小应用程序的分析方面已经取得了一些成功,但是对到达目标状态的执行路径

的一般性分布特点还缺乏理论和经验研究的支持,引导不同程序执行到特定位置的方法体系还未形成,复杂场景约束的建模和关联求解也还未形成系统化的方法.除这些特有的问题外,精准执行可达性分析仍然面临诸如搜索空间爆炸、难解的 SMT 问题、接口建模和并发分析等程序分析的挑战.由于精准执行可达性分析以确定位置的目标状态为导向,灵活利用不同抽象层和逻辑切面约束信息来设计更加专注的试探策略,以及通过抽象目标状态特性,形成有针对性的聚焦和精准推理,仍然是未来研究的主题.

在复杂约束的关联分析方面,还缺乏对许多内存漏洞的利用自动构造问题的研究.诸如堆溢出、释放后重用、使用时未初始化以及与数据竞争和锁等并发控制相关的漏洞利用自动构造问题还有待尝试.除漏洞利用构造问题外,更多含有复杂约束的应用场景有待发现,可考虑在兼容、版本变更、缺陷自动修复等需要进行多因素考量的场景中挖掘类似的问题.另外,寻找触发同一缺陷的差异化执行路径[26]也是一个有价值的研究主题.在一些非合作场景中,为了降低试探失败的风险,对输入的成功率有了更高的要求,对非常态软件(代码混淆、加密)的精准执行可达性分析值得扩展.

在解决路径爆炸问题上,归纳和摘要对重复计算的预测和等价类划分是有效的.多元素数据结构常用操作的摘要和与他们相关的最弱前置条件快速推算方法值得研究.例如,线性数据结构的处理就有很多可归纳的操作,包含查找、排序、筛选、截取、拼接、复制、逆序等.探索高层逻辑和系统机制上以非公式形态出现的最弱前置条件表达[13],对执行可达性分析有重要的研究意义.除了从模型和机制上抽象和简化问题,程序分析技术也需要大数据和分布式计算来强化自身,毕竟单台计算机的处理能力是有限的.当然,分布式程序上的各类调试和测试问题也给精准执行可达性分析带来了新机遇.

对与外部软硬件交互复杂的程序,如操作系统内核、设备驱动、中间件、图形界面程序和分布式程序等的分析需要较强的领域知识作辅助以保障准确性.在对执行路径规律不了解的情况下,案例研究和经验统计研究是很好的突破口.此外,软件执行机制的高层抽象模型需要与执行路径细节分析有机地结合起来[13],通过分析 API 文档来抽取高层逻辑模型以及最弱前置条件相关信息[108]值得进一步探索.

对于运行时决定的间接控制流跳转问题,虽然在二进制程序中没有较好的追溯办法,但在源代码中,通过细致的分析还是可以找到若干可能的函数调用点的.由于目标状态单一且代码位置明确,指向分析不再是全局的而是按需进行的.以 C 语言为例,识别这些控制流除了要展开宏以及识别动态和静态加载的库函数,还需要对自动化编译脚本以及配置文件等进行分析.

最后,与其他学科和理论相结合可以开启更好的研究视角.例如,机器学习技术可能对基本数据结构和算法的识别以及自动程序摘要有帮助,特殊编程模式下的数据依赖定位和约束求解[98]中也可能蕴含着丰富的统计规律.使用了神经网络等“黑盒子”技术的软件系统也给执行可达性分析带来了新的挑战.

References:

[1] Rice HG. Classes of recursively enumerable sets and their decision problems. Trans. of the American Mathematical Society, 1953, 74(2):358−366. [doi: 10.2307/1990888]

[2] Shoshitaishvili Y, Wang R, Salls C, Stephens N, Polino M, Dutcher A, Grosen J, Feng S, Hauser C, Kruegel C, Vigna G. SOK: (state of) The art of war: Offensive techniques in binary analysis. In: Proc. of the 2016 IEEE Symp. on Security and Privacy (S&P). Piscataway: IEEE Press, 2016. 138−157. [doi: 10.1109/SP.2016.17]

[3] Christakis M, Müller P, Wüstholz V. Guiding dynamic symbolic execution toward unverified program executions. In: Proc. of the 38th Int'l Conf. on Software Engineering. New York: ACM Press, 2016. 144−155. [doi: 10.1109/SP.2012.31]

[4] Haller I, Slowinska A, Neugschwandtner M, Bos H. Dowsing for overflows: A guided fuzzer to find buffer boundary violations. In: King S, ed. Proc. of the 22nd USENIX Security Symp. Berkeley: USENIX Association, 2013. 49−64.

[5] Guo X, Wang P, Wang JY, Zhang HG. Program multiple execution paths verification based on *k* proximity weakest precondition. Chinese Journal of Computers, 2015,38(11):2203−2214 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2015.02203]

[6] Avgerinos T, Sang KC, Hao BLT, Brumley D. AEG: Automatic exploit generation. In: Proc. of the Network and Distributed System Security Symp. San Diego: Internet Society, 2011. [doi: 10.1145/2560217.2560219]

[7] Avgerinos T, Cha SK, Rebert A, Schwartz JE, Woo M, Brumley D. Automatic exploit generation. Communications of the ACM, 2014,57(2):74−84. [doi: 10.1145/2560217.2560219]

[8] Hu H, Chua ZL, Adrian S, Saxena P, Liang Z. Automatic generation of data-oriented exploits. In: Jung J, Holz T, eds. Proc. of the 24th USENIX Security Symp. Berkeley: USENIX Association, 2015. 177−192.

[9] Baluda M, Denaro G, Pezze M. Bidirectional symbolic analysis for effective branch testing. IEEE Trans. on Software Engineering, 2016,42(5):403−426. [doi: 10.1109/TSE.2015.2490067]

[10] Cadar C, Sen K. Symbolic execution for software testing: Three decades later. Communications of the ACM, 2013,56(2):82−90. [doi: 10.1145/2408776.2408795]

[11] 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). [doi: 10.3724/SP.J.1016.2009.01697]

[12] Nayrolles M, Hamoulhadj A, Tahar S, Larsson A. JCHARMING: A bug reproduction approach using crash traces and directed model checking. In: Proc. of the 22nd IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Piscataway: IEEE Press, 2015. 101−110. [doi: 10.1109/SANER.2015.7081820]

[13] Jensen CS, Prasad MR. Automated testing with targeted event sequence generation. In: Proc. of the 2013 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2013. 67−77. [doi: 10.1145/2483760.2483777]

[14] Johnson B, Song Y, Murphyhill E, Bowdidge R. Why don't software developers use static analysis tools to find bugs? In: Proc. of the 2013 Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2013. 672−681. [doi: 10.1109/ICSE.2013.6606613]

[15] Parnin C, Orso A. Are automated debugging techniques actually helping programmers? In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2011. 199−209. [doi: 10.1145/2001420.2001445]

[16] Fang M, Hafiz M. Discovering buffer overflow vulnerabilities in the wild: An empirical study. In: Proc. of the 8th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. New York: ACM Press, 2014. 23. [doi: 10.1145/2652524.2652533]

[17] King JC. Symbolic execution and program testing. Communications of the ACM, 1976,19(7):385−394. [doi: 10.1145/360248.360252]

[18] Dijkstra EW. A Discipline of Programming. New York: Prentice Hall, 1976.

[19] Jin JW, Ma FF, Zhang J. Brief introduction to SMT solving. Journal of Frontiers of Computer Science and Technology, 2015,9(7):769−780 (in Chinese with English abstract). [doi: 10.3778/j.issn.1673-9418.1405041]

[20] Li J, Liu WW. A survey on theoretical combination techniques of SMT solvers. Computer Engineering & Science, 2011,33(10):111−119 (in Chinese with English abstract).

[21] Beckman NE, Nori AV, Rajamani SK, Simmons RJ. Proofs from tests. In: Proc. of the 2008 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2008. 3−14. [doi: 10.1145/1390630.1390634]

[22] Tomb A. Program inconsistency detection: Universal reachability analysis and conditional slicing [Ph.D. Thesis]. Santa Cruz: University of California at Santa Cruz, 2011.

[23] Weiser M. Program slicing. In: Proc. of the 5th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 1981. 439−449. [doi: 10.1109/TSE.1984.5010248]

[24] Korel B, Laski J. Dynamic program slicing. Information Processing Letters, 1988,29(3):155−163. [doi: 10.1016/0020-0190(88)90054-3]

[25] Sinha N, Singhania N, Chandra S, Sridharan M. Alternate and learn: Finding witnesses without looking all over. In: Madhusudan P, Seshia SA, eds. Proc. of the 24th Int'l Conf. on Computer Aided Verification. Berlin, Heidelberg: Springer-Verlag, 2012. 599−615.

[26] Alipour MA, Groce A, Gopinath R, Christi A. Generating focused random tests using directed swarm testing. In: Proc. of the 25th Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2016. 70−81. [doi: 10.1145/2931037.2931056]

[27] Chen TY, Kuo FC, Merkel RG, Tse TH. Adaptive random testing: The ART of test case diversity. Journal of Systems & Software, 2010,83(1):60−66. [doi: 10.1016/j.jss.2009.02.022]

[28] Harman M, Mansouri SA, Zhang Y. Search-Based software engineering: Trends, techniques and applications. ACM Computing Surveys, 2012,45(1):1−61. [doi: 10.1145/2379776.2379787]

[29] Xuan J, Xie X, Monperrus M. Crash reproduction via test case mutation: Let existing test cases help. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. New York: ACM Press, 2015. 910−913. [doi: 10.1145/2786805.2803206]

[30] Soltani M, Panichella A, Deursen AV. A guided genetic algorithm for automated crash reproduction. In: Proc. of the Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2017. 209−220. [doi: 10.1109/ICSE.2017.27]

[31] Böhme M, Pham VT, Roychoudhury A. Coverage-Based greybox fuzzing as Markov chain. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. New York: ACM Press, 2016. 1032−1043. [doi: 10.1145/2976749.2978428]

[32] Zamfir C, Candea G. Execution synthesis: A technique for automated software debugging. In: Proc. of the 5th European Conf. on Computer Systems. New York: ACM Press, 2010. 321−334. [doi: 10.1145/1755913.1755946]

[33] Ma KK, Phang KY, Foster JS, Hicks M. Directed symbolic execution. In: Yahav E, ed. Proc. of the 18th Int'l Conf. on Static Analysis. Berlin, Heidelberg: Springer-Verlag, 2011. 95−111. [doi: 10.1007/978-3-642-23702-7_11]

[34] Marinescu PD, Cadar C. KATCH: High-Coverage testing of software patches. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. New York: ACM Press, 2013. 235−245. [doi: 10.1145/2491411.2491438]

[35] Isaev IK, Sidorov DV. The use of dynamic analysis for generation of input data that demonstrates critical bugs and vulnerabilities in programs. Programming & Computer Software, 2010,36(4):225−236. [doi: 10.1134/S0361768810040055]

[36] Sidiroglou-Douskos S, Lahtinen E, Rittenhouse N, Piselli P, Long F, Kim D, Rinard M. Targeted automatic integer overflow discovery using goal-directed conditional branch enforcement. In: Proc. of the 20th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2015. 473−486. [doi: 10.1145/2775054.2694389]

[37] Pham VT, Ng WB, Rubinov K, Roychoudhury A. Hercules: Reproducing crashes in real-world application binaries. In: Proc. of the 37th Int'l Conf. on Software Engineering—Vol.1. Piscataway: IEEE Press, 2015. 891−901. [doi: 10.1109/ICSE.2015.99]

[38] Chen N, Kim S. STAR: Stack trace based automatic crash reproduction via symbolic execution. IEEE Trans. on Software Engineering, 2015,41(2):198−220. [doi: 10.1109/TSE.2014.2363469]

[39] Holte RC, Felner A, Sharon G, Sturtevant NR. Bidirectional search that is guaranteed to meet in the middle. In: Schuurmans D, Wellman M P, eds. Proc. of the 30th AAAI Conf. on Artificial Intelligence. Palo Alto: AAAI Press, 2016. 3411−3417.

[40] Xie X, Chen B, Liu Y, Le W, Li X. Proteus: Computing disjunctive loop summary via path dependency analysis. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2016. 61−72. [doi: 10.1145/2950290.2950340]

[41] Xie X, Liu Y, Le W, Li XH, Chen HX. S-Looper: Automatic summarization for multipath string loops. In: Proc. of the 2015 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2015. 188−198. [doi: 10.1145/2771783.2771815]

[42] Godefroid P, Luchaup D. Automatic partial loop summarization in dynamic test generation. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2011. 23−33. [doi: 10.1145/2001420.2001424]

[43] Nie CJ, Liu HF, Su PR, Feng DG. A loop summarization method for dynamic binary program analysis. Chinese Journal of Electronics, 2014,42(6):1110−1117 (in Chinese with English abstract). [doi: 10.3969/j.issn.0372-2112.2014.06.012]

[44] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Driessche GVD, Schrittwieser J, Antonoglou L, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever L, Lillcrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. Mastering the game of Go with deep neural networks and tree search. Nature, 2016,529(7587):484−489. [doi: 10.1038/nature16961]

[45] Hwang GH, Tai KC, Huang TL. Reachability testing: An approach to testing concurrent software. In: Proc. of the 1st Asia-Pacific Software Engineering Conf. IEEE, 1994. 246−255. [doi: 10.1109/APSEC.1994.465255]

[46] Lu S, Park S, Zhou Y. Finding atomicity-violation bugs through unserializable interleaving testing. IEEE Trans. on Software Engineering, 2012,38(4):844−860. [doi: 10.1109/TSE.2011.35]

[47] 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. New York: ACM Press, 2007. 552−561. [doi: 10.1145/1315245.1315313]

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

[49] Checkoway S, Davi L, Dmitrienko A, Dmitrienko A, Sadegh AR. Return-Oriented programming without returns. In: Proc. of the 17th ACM Conf. on Computer and Communications Security. New York: ACM Press, 2010. 559−572.

[50] Bosman E, Bos H. Framing signals—A return to portable shellcode. In: Proc. of the 2014 IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society, 2014. 243−258. [doi: 10.1109/SP.2014.23]

[51] Bittau A, Belay A, Mashtizadeh A, Mazieres D, Boneh D. Hacking blind. In: Proc. of the 2014 IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society, 2014. 227−242. [doi: 10.1109/SP.2014.22]

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

[53] Tice C, Roeder T, Collingbourne P, Checkoway S, Erlingsson Ú, Lozano L, Pike G. Enforcing forward-edge control-flow integrity in GCC & LLVM. In: Fu K, ed. Proc. of the 23rd USENIX Conf. on Security Symp. Berkeley: USENIX Association, 2014. 941−955. https://www.usenix.org/node/184460

[54] Zhang M, Sekar R. Control flow integrity for COTS binaries. In: King S T, ed. Proc. of the 22nd USENIX Conf. on Security. Berkeley: USENIX Association, 2013. 337−352. https://www.usenix.org/node/174767

[55] Zhang C, Wei T, Chen Z, Duan L, Szekeres L, McCamant S, Song D, Zou W. Practical control flow integrity and randomization for binary executables. In: Proc. of the IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society, 2013. 559−573. [doi: 10.1109/SP.2013.44]

[56] Niu B, Tan G. Per-Input control-flow integrity. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security. New York: ACM Press, 2015. 914−926. [doi: 10.1145/2810103.2813644]

[57] Veen VVD, Andriesse D, Göktaş E, Gras B, Sambuc L, Slowinska A, Bos H, Giuffrida C. Practical context-sensitive CFI. In: Proc. of the 22nd ACM SIGSAC Conf. on Computer and Communications Security. New York: ACM Press, 2015. 927−940. [doi: 10.1145/2810103.2813673]

[58] Hong H, Shweta S, Sendroiu A, Zhang LC, Saxena P, Liang Z. Data-Oriented programming: On the expressiveness of non-control data attacks. In: Proc. of the IEEE Symp. on Security and Privacy (Oakland 2016). Piscataway: IEEE Press, 2016. [doi: 10.1109/SP.2016.62]

[59] Schwartz EJ, Avgerinos T, Brumley D. Q: Exploit hardening made easy. In: Wagner D, ed. Proc. of the 20th USENIX Conf. on Security Symp. Berkeley: USENIX Association, 2011. 25. http://www.usenix.org/events/sec11/tech/full_papers/Schwartz.pdf

[60] Reps T. Undecidability of context-sensitive data-dependence analysis. ACM Trans. on Programming Languages and Systems (TOPLAS), 2000,22(1):162−186. [doi: 10.1145/345099.345137]

[61] Lhoták O, Hendren L. Context-Sensitive points-to analysis: Is it worth it? In: Proc. of the 15th Int'l Conf. on Compiler Construction. Berlin, Heidelberg: Springer-Verlag, 2006. 47−64. [doi: 10.1007/11688839_5]

[62] Andersen LO. Program analysis and specialization for the C programming language [Ph.D. Thesis]. Copenhagen: University of Copenhagen, 1994.

[63] Steensgaard B. Points-To analysis in almost linear time. In: Proc. of the 23rd ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. New York: ACM Press, 1996. 32−41. [doi: 10.1145/237721.237727]

[64] Heintze N, Tardieu O. Demand-Driven pointer analysis. ACM SIGPLAN Notices, 2001,36(5):24−34. [doi: 10.1145/381694.378802]

[65] Zheng X, Rugina R. Demand-Driven alias analysis for C. ACM SIGPLAN Notices, 2008,43(1):197−208. [doi: 10.1145/1328897.1328464]

[66] Yan D, Xu G, Rountev A. Demand-Driven context-sensitive alias analysis for Java. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2011. 155−165. [doi: 10.1145/2001420.2001440]

[67] Shang L, Lu Y, Xue J. Fast and precise points-to analysis with incremental CFL-reachability summarisation: Preliminary experience. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2012. 270−273. [doi: 10.1145/2351676.2351720]

[68] Shang L, Xie X, Xue J. On-Demand dynamic summary-based points-to analysis. In: Proc. of the 10th Int'l Symp. on Code Generation and Optimization. New York: ACM Press, 2012. 264−274. [doi: 10.1145/2259016.2259050]

[69] Feng Y, Wang X, Dillig I, Dillig T. Bottom-Up context-sensitive pointer analysis for Java. In: Proc. of the Asian Symp. on Programming Languages and Systems. Cham: Springer Int'l Publishing, 2015. 465−484. [doi: 10.1007/978-3-319-26529-2_25]

[70] Wilhelm R, Sagiv M, Reps T. Shape analysis. In: Proc. of the Int'l Conf. on Compiler Construction. Berlin, Heidelberg: Springer-Verlag, 2000. 1−17. [doi: 10.1007/3-540-46423-9_1]

[71] Sagiv M, Reps T, Wilhelm R. Solving shape-analysis problems in languages with destructive updating. ACM Trans. on Programming Languages and Systems (TOPLAS), 1998,20(1):1−50. [doi: 10.1145/271510.271517]

[72] Sagiv M, Reps T, Wilhelm R. Parametric shape analysis via 3-valued logic. ACM Trans. on Programming Languages and Systems (TOPLAS), 2002,24(3):217−298. [doi: 10.1145/514188.514190]

[73] Itzhaky S, Bjørner N, Reps T, Sagiv M, Thakur A. Property-Directed shape analysis. In: Armin B, Roderick B, eds. Proc. of the Int'l Conf. on Computer Aided Verification. Cham: Springer Int'l Publishing, 2014. 35−51. [doi: 10.1007/978-3-319-08867-9_3]

[74] Godefroid P. Compositional dynamic test generation. ACM Sigplan Notices, 2007,42(1):47−54. [doi: 10.1145/1190215.1190226]

[75] Anand S, Godefroid P, Tillmann N. Demand-Driven compositional symbolic execution. In: Ramakrishnan CR, Jakob R, eds. Proc. of the Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Heidelberg: Springer-Verlag, 2008. 367−381. [doi: 10.1007/978-3-540-78800-3_28]

[76] Avgerinos T, Rebert A, Cha SK, Brumley D. Enhancing symbolic execution with veritesting. In: Proc. of the 36th Int'l Conf. on Software Engineering. New York: ACM Press, 2014. 1083−1094. [doi: 10.1145/2568225.2568293]

[77] Yi Q, Yang Z, Guo S, Wang C, Liu J, Zhao C. Postconditioned symbolic execution. In: Proc. of the IEEE Int'l Conf. on Software Testing, Verification and Validation. IEEE, 2015. 1−10. [doi: 10.1109/ICST.2015.7102601]

[78] Cadar C, Dunbar D, Engler D. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Draves R, Renesse R V, eds. Proc. of the USENIX Conf. on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2008. 209−224. https://dl.acm.org/citation.cfm?id=1855756

[79] He YX, Wu W, Chen Y, Xu C. Path sensitive program verification based on SMT solvers. Ruan Jian Xue Bao/Journal of Software, 2012,23(10):2655−2664 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4196.htm [doi: 10.3724/SP.J.1001.2012.04196]

[80] Qin S, He G, Luo C, Chin WN, Chen X. Loop invariant synthesis in a combined abstract domain. Journal of Symbolic Computation, 2013,50(3):386−408. [doi: 10.1016/j.jsc.2012.08.007]

[81] Li ZP, Zhang Y, Chen YY. A shape graph logic and a shape system. Journal of Computer Science and Technology, 2013,28(6):1063−1084. [doi: 10.1007/s11390-013-1398-1]

[82] Zhang Y, Chen YY, Li ZP. Theorem proving for a theory of shape graphs. Chinese Journal of Computers, 2016,39(12):2460−2480 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2016.02460]

[83] Chipounov V, Georgescu V, Zamfir C, Candea G. Selective symbolic execution. In: Fetzer C, Rodrigues R, eds. Proc. of the Workshop on Hot Topics in System Dependability. 2009. 1286−1299. https://infoscience.epfl.ch/record/139393/files/selsymbex.pdf

[84] Chipounov V, Kuznetsov V, Candea G. S2E: A platform for in-vivo multi-path analysis of software systems. ACM SIGPLAN Notices, 2011,46(3):265−278. [doi: 10.1145/1961296.1950396]

[85] Henderson A, Prakash A, Yan LK, Hu X, Wang X, Zhou R, Yin H. Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform. In: Proc. of the 2014 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2014. 248−258. [doi: 10.1145/2610384.2610407]

[86] Henderson A, Yan LK, Hu X, Prakash A, Yin H, McCamant S. DECAF: A platform-neutral whole-system dynamic binary analysis platform. IEEE Trans. on Software Engineering, 2017,43(2):164−184. [doi: 10.1109/TSE.2016.2589242]

[87] Lei Y, Carver RH. Reachability testing of concurrent programs. IEEE Trans. on Software Engineering, 2006,32(6):382−403. [doi: 10.1109/TSE.2006.56]

[88] Sen K, Agha G. Automated systematic testing of open distributed programs. In: Luciano B, Reiko H, eds. Proc. of the 9th Int'l Conf. on Fundamental Approaches to Software Engineering. Berlin, Heidelberg: Springer-Verlag, 2006. 339−356. [doi: 10.1007/11693017_25]

[89] Taylor RN, Levine DL, Kelly CD. Structural testing of concurrent programs. IEEE Trans. on Software Engineering, 1992,18(3):206−215. [doi: 10.1109/32.126769]

[90] Koppol PV, Tai KC. An incremental approach to structural testing of concurrent software. ACM SIGSOFT Software Engineering Notes, 1996,21(3):14−23. [doi: 10.1145/226295.226298]

[91] Long ZY. Automatic analysis and verification of concurrent programs [Ph.D. Thesis]. Beijing: University of Chinese Academy of Sciences, 2013 (in Chinese with English abstract).

[92] Zeng YQ. Research on context-bounded reachability of concurrent programs [MS. Thesis]. Guilin: Guilin University of Electronic Technology, 2013 (in Chinese with English abstract).

[93] Musuvathi M, Qadeer S, Ball T, Basler G, Nainar PA, Neamtiu I. Finding and reproducing Heisenbugs in concurrent programs. In: Draves R, Renesse RV, eds. Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2008. 267−280. https://dl.acm.org/citation.cfm?id=1855760

[94] Park S, Lu S, Zhou Y. CTrigger: Exposing atomicity violation bugs from their hiding places. In: Proc. of the Architectural Support for Programming Languages and Operating Systems. 2009. 25−36. [doi: 10.1145/1508284.1508249]

[95] Sen K. Race directed random testing of concurrent programs. ACM SIGPLAN Notices, 2008,43(6):11−21. [doi: 10.1145/1379022.1375584]

[96] Weeratunge D, Zhang X, Jagannathan S. Analyzing multicore dumps to facilitate concurrency bug reproduction. In: Proc. of the Int'l Conf. on Architectural Support for Programming Languages & Operating Systems. New York: ACM Press, 2010. 155−166. [doi: 10.1145/1736020.1736039]

[97] Huang J, Zhang C. LEAN: Simplifying concurrency bug reproduction via replay-supported execution reduction. In: Proc. of the ACM Int'l Conf. on Object Oriented Programming Systems Languages and Applications. New York: ACM Press, 2012. 451−466. [doi: 10.1145/2398857.2384649]

[98] Li X, Liang Y, Qian H, Hu YQ, Bu L, Yu Y, Chen X, Li X. Symbolic execution of complex program driven by machine learning based constraint solving. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2016. 554−559. [doi: 10.1145/2970276.2970364]

[99] Dinges P, Agha G. Targeted test input generation using symbolic-concrete backward execution. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2014. 31–36. [doi: 10.1145/2642937.2642951]

[100] Person S, Yang G, Rungta N, Khurshid S. Directed incremental symbolic execution. ACM SIGPLAN Notices, 2011,46(6):504–515.

[101] Babić D, Martignoni L, McCamant S, Song D. Statically-Directed dynamic automated test generation. In: Dwyer M, ed. Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2011. 12–22. [doi: 10.1145/2001420.2001423]

[102] Parvez MR. Combining static analysis and targeted symbolic execution for scalable bug-finding in application binaries [MS. Thesis]. Waterloo: University of Waterloo, 2016.

[103] Gao F, Wang L, Li X. BovInspector: Automatic inspection and repair of buffer overflow vulnerabilities. In: Lo D, Apel S, Khurshid S, eds. Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2016. 786–791.

[104] Jin W, Orso A. BugRedux: Reproducing field failures for in-house debugging. In: Proc. of the 34th Int'l Conf. on Software Engineering. Piscataway: IEEE Press, 2012. 474–484. [doi: 10.1109/ICSE.2012.6227168]

[105] Sang KC, Avgerinos T, Rebert A, Brumley D. Unleashing mayhem on binary code. In: Proc. of the IEEE Symp. on Security and Privacy. Washington: IEEE Computer Society, 2012. 380–394. [doi: 10.1109/SP.2012.31]

[106] Ramos DA, Engler D. Under-Constrained symbolic execution: Correctness checking for real code. In: Jung J, ed. Proc. of the 24th USENIX Security Symp. Berkeley: USENIX Association, 2015. 49–64. https://www.usenix.org/node/196226

[107] Rawat S, Jain V, Kumar A, Cojocar L, Giuffrida C, Bos H. VUzzer: Application-Aware evolutionary fuzzing. In: Proc. of the Network and Distributed System Security Symp. San Diego: Internet Society, 2017. [doi: 10.14722/ndss.2017.23404]

[108] Jana S, Kang YJ, Roth S, Ray B. Automatically detecting error handling bugs using error specifications. In: THolz T, Savage S, eds. Proc. of the USENIX Security Symp. Berkeley: USENIX Association, 2016. 345–362.

附中文参考文献:

[5] 郭曦,王盼,王建勇,张焕国.基于 k 近邻最弱前置条件的程序多路径验证方法.计算机学报,2015,38(11):2203–2214. [doi: 10.11897/SP.J.1016.2015.02203]

[11] 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. [doi: 10.3724/SP.J.1016.2009.01697]

[19] 金继伟,马菲菲,张健.SMT 求解技术简述.计算机科学与探索,2015,9(7):769–780. [doi: 10.3778/j.issn.1673-9418.1405041]

[20] 李婧,刘万伟.SMT 求解器理论组合技术研究.计算机工程与科学,2011,33(10):111–119.

[43] 聂楚江,刘海峰,苏璞睿,冯登国.一种面向程序动态分析的循环摘要生成方法.电子学报,2014,42(6):1110–1117. [doi: 10.3969/j.issn.0372-2112.2014.06.012]

[79] 何炎祥,吴伟,陈勇,徐超.基于 SMT 求解器的路径敏感程序验证.软件学报,2012,23(10):2655–2664. http://www.jos.org.cn/1000-9825/4196.htm [doi: 10.3724/SP.J.1001.2012.04196]

[82] 张昱,陈意云,李兆鹏.形状图理论的定理证明.计算机学报,2016,39(12):2460–2480. [doi: 10.11897/SP.J.1016.2016.02460]

[91] 龙震岳.并发程序的自动分析与验证[博士学位论文].北京:中国科学院大学,2013.

[92] 曾宇清.基于上下文限界的并发程序可达性研究[硕士学位论文].桂林:桂林电子科技大学,2013.

杨克(1989－),男,河北辛集人,硕士,CCF 学生会员,主要研究领域为软件安全分析,操作系统安全.

马恒太(1970－),男,博士,副研究员,主要研究领域为软件安全分析,操作系统安全.

贺也平(1962－),男,博士,研究员,博士生导师,主要研究领域为系统安全,隐私保护.

王雪飞(1989－),女,助理工程师,主要研究领域为软件安全分析,操作系统安全.