分析稀疏值流图(SVFG)中指针引用的传递和值传递来判定内存泄漏.Melton[31]提出了一种过程间分析的算法,利用符号执行生成内存状态转换图(MSTG),在分析过程中监测内存泄漏.除了学术界,许多商业工具,如Klocwork[11],Coverity[12]和 Fortify[13]等在实际的软件开发中使用广泛.然而,静态分析工具依旧受限于规模,虽然已有大量针对全局、可扩展的内存分析的研究工作,如何将静态分析应用到大规模的软件上依然是一个受到广泛关注的问题.本文提出的方法在现有的静态分析方法的基础上对静态分析的警报进行验证和分类,以期提高静态分析工具的实用性和查准率.

内存泄漏的动态测试工具则是在程序执行的过程中,根据内存的实际变化来判断泄漏是否发生.动态测试能够准确地检测某一运行过程中是否发生了内存泄漏,同时能够准确地找出泄漏点,但是它也存在一些问题.例如,动态检测需要通过插装或者外部监测的方法对内存情况进行监测和控制,具有较大的开销,对设备硬件要求较高.除此之外,动态检测工具只能检测输入路径的内存情况,对输入的测试用例的质量和覆盖度要求很高,对于覆盖不到的路径很可能产生漏报,因此,动态测试技术依赖较高质量的测试用例.目前,学术界也有许多针对C/C++的内存泄漏动态检测工作.LeakPoint[1]基于污点传播的方法来监测内存,追踪内存最后使用的位置以及失去引用的位置.Purify[2]采用插装的方式来跟踪内存的使用情况,在目标代码中插入进行内存追踪的语句,是动态测试技术最早的思想基础.Omega[3]和 Maebe[6]主要采用指针计数的思想记录内存对象的引用计数.SafeMem[5]利用纠错内存(ECC memory)开发了内存使用情况分析技术,可以降低监测的开销.Valgrind[32]采用了动态插装技术,即在执行时进行插装.Sniper[33]主要利用处理器中的性能监测单元(performance monitoring unit,简称 PMU)取样,在运行时监测堆的状态,提出了基于 staleness 的内存泄漏检测技术.在此基础上,文献[34]对堆进行抽象建模,并使用机器学习来检测内存泄漏.另外,动态测试技术在 Java 等托管语言中的内存泄漏检测方面也得到了很多应用[35−46].

本文的工作主要有两个方面与上述相关工作不同:第一,本文工作的目标在于确认静态分析的警报,与任意静态分析技术互补;第二,本文进行警报确认的混合执行测试,利用了可达警报路径制导,减少了内存泄漏无关路径的测试执行,与其他动态测试技术相比,降低了开销,提高了效率.

文献[47]对内存泄漏进行了自动修复,通过对程序中指针、控制流、数据流等的分析,在检测内存泄漏的同时尝试安全地修复泄漏点.在这类工作中,能够成功修复的部分往往代表正确的检测.但是这类工作修复过程较为保守,能够修复的部分并不全面,可能造成漏报或无法修复的情况.其中,无法修复的情况无法确认其正确性.而我们的工作基于已有静态分析工具给出的警报,对所有情况进行分析,不仅针对正确的警报,同时也针对误报的警报以及 BLOAT 类型,给出准确的分类.

4.2 测试生成方法

测试生成在软件测试中是十分重要的组成部分,生成高质量的测试用例一直是学术界和业界的关注热点.

将代码抽象为模型,则运行程序的过程就是对模型的具体化的过程.

对程序的测试,可以使用不同的具体数值对模型进行实例化,以检验在任何输入情况下是否都可以得到符合预期的结果.这样的方法可称为具体执行(concrete execution).

区别于具体执行,符号执行(symbolic execution)是将程序执行过程中的变量抽象为符号表达式,经过符号化过程,每一条路径都有了相对应的符号表示的约束条件,可以利用约束求解器解出不同程序路径所对应的具体值范围[48],形成输入.

混合执行测试(concolic testing)[14,15]则是对具体执行和符号执行[48,49]两种方法的结合.混合执行测试在具体执行的过程中,同时进行符号执行和符号化路径约束条件的收集.通过具体的执行,程序可以得到本次执行的路径中所有的约束条件,比如分支约束条件.对其中某一个分支约束条件取反,就会得到同一分支的不同路径的输入值,即得到一条新的路径.通过这样的方法,不断地探索程序路径空间,以尽可能地覆盖更多的代码.混合执行测试的优势在于:在符号执行过程中,一旦我们想得到一条路径的输入,就需要先得到这条路径中的所有条件约束在进行约束求解,而混合执行测试是边执行边生成的,同时,具体的数值可以被用于简化符号执行的约束条件.然而,混合执行测试也面临着挑战,比如无法处理规模较大的程序以及建模准确度不够.因此,混合执行测试

工具通常采用一定时间或迭代次数作为测试生成过程的限制.但是这也会带来一些问题,比如路径空间不全、遍历受限不完整等,依旧有亟需优化的地方.混合执行测试的方法被应用于越来越多的领域[50,51],实现了各种混合执行测试工具,例如,Conpy[52]是混合执行测试在 Python 上的实现.

测试生成与其他技术相结合的例子也有很多,比如:协同算法[53]将模型检验(model-checking)与动态符号执行工作 DART[14]相结合,从而尝试遍历程序中的所有状态;文献[54]同样利用了混合执行测试生成测试用例.这些测试生成的目的都是得到更高的覆盖度,尽可能地遍历程序的每一个角落.而我们的工作并不是针对于此,而是希望通过测试生成得到我们需要的某一条或者某几条路径.

与我们的思路有一定相似之处,一些学者也将动态和静态的技术结合起来使用,例如,DyTa[55]工具结合了静态程序验证与动态测试生成工作,以期达到降低误报和提高效率的目的;DSD-Crasher[56]工具依次采用动态-静态-动态方法来发现程序错误;Zhang 和 Saff 等人[57]则针对单元测试提出了一种静动态相结合的方法,用于自动地产生有效和多样的方法调用序列;Babić 等人[58]利用静态分析来指导二进制程序的自动测试生成;Taneja 等人[59]利用基于路径的测试生成来进行更加高效的回归测试;文献[60]提出了一种规则制导(rule-directed)的符号执行技术来高效地检查系统规则;李游等人[61]利用一种 n-变长子路径程序频谱(length-n subpath program spectra)来系统地引导符号执行过程覆盖更“少”被覆盖到的程序路径空间.本文的混合执行测试方法是以崔展齐等人[62]提出的一种目标制导的混合执行测试方法为基础的、高效的测试生成和执行方法,能够使得本文的静态警报确认更加高效.

4.3 静态分析结果的验证和误报消除

静态分析工具在实际中的应用十分广泛,上文也进行了分析.静态分析工具极容易出现误报和漏报的情况,在这个方向也有一定的相关工作积累.

Ruthruff 等人[63]通过挖掘现有警报中潜在的信息以及相关联的代码,建立基于统计的回归模型来预测之后静态警报的类型.Heckman 等人[64]提出一种减少误报的技术,并用一个基准程序集 FAULTBENCH 来验证该技术.Kim 和 Ernst[65]通过分析软件的多个版本来对警报进行分类.FEEDBACK-RANK[66]利用已知的对静态分析警报的修正,开发出一种基于概率的排序方法来降低误报的数目.Z-Ranking 利用人工验证静态分析结果的统计数据和频次来对警报进行优先级排序[67].Boogerd 和 Moonen 通过采用似然分析来排序静态分析警报[68].Kim 和 Ernst[69]则通过软件变更历史,找到静态分析警报和实际错误修复之间的关系,进而提出一种基于历史的警报优先级(history-based warning prioritization,简称 HWP)处理算法,从而利用修复的历史来对警报进行排序.Dillig 等人提出一种基于诱发推测的算法,该算法被用于判定静态分析中丢失的信息,从而诊断静态分析的警报[70].Clarify[71]工具通过对软件执行情况的总结,利用机器学习的方法,以期改善静态分析的错误报告.ALETHEIA[72]工具同样使用了机器学习的方法,用户只需要人工确认一部分警报,其余的警报利用机器学习的方法进行分类.Chimdyalwar 等人[73]提出了循环抽象的有界模型检验的方法来检测静态分析中的误报,用已知的小边界的抽象循环替换程序中的循环.

在以上这些相关工作中,大多数都是通过静态分析的结果进行二次分析,并没有结合动态执行.文献[70]虽然类似于我们的方法,进行了动态测试,但其目的主要在于采集更多的信息,帮助开发人员理解和修复缺陷.因此,当前并没有与我们完全相同的工作.我们在前期工作[74]中提出了采用混合执行测试方法对静态内存泄漏警报进行分类的思想和方法,这是本文之前的工作,也是本文的基础.本文增加了与动态测试技术的对比实验,在该实验中,一方面对比了内存泄漏的动态测试工具,另一方面,利用高覆盖度的测试用例集合作为测试输入,替代混合执行产生的测试用例,执行插装后的程序对警报进行分类,进一步证明了本文思想和方法的有效性和效率.利用动静态结合的方法,能够对静态内存泄漏警报进行准确的分类,同时也大大降低了时空开销.

5 总结与展望

本文提出了一种基于混合执行测试的 C/C++程序静态内存泄漏警报自动确认方法,对静态分析工具报告的内存泄漏警报进行动态的自动化确认和分类.首先,在被测程序的控制流图上进行警报的可达性分析,计算程

序中分支语句到路径片段中节点的可达性,得到路径制导信息;其次,以控制流图上静态内存泄漏警报的可能路径为覆盖目标,基于混合执行测试方法产生测试用例并执行;最后,追踪和监控运行时内存对象的状态,在测试执行结束时判断内存泄漏是否存在,从而完成对静态内存泄漏警报的确认和分类.

在本文方法的实现过程和实验中,我们也遇到了一些问题和挑战,这也是我们未来的研究方向:(1) 本文对内存状态的判断是通过对其的追踪和更新完成的,可以在状态中加入更多内容以得到更多的信息,例如在MUST- LEAK 中,利用内存泄漏的大小对警报进行一个更细的排序;(2) 在内存追踪时加入内存对象的指针或者引用计数,从而更准确地验证警报;(3) 将当前的工作扩展到其他类型的内存缺陷中;(4) 研究更高效的测试策略,提升本文方法的规模化能力.

致谢 在此,向对本文工作给予建议的老师、参与本文实验的所有同学、给我们提出建议的评审专家表示感谢.

References:

[1] Clause J, Orso A. LEAKPOINT: Pinpointing the causes of memory leaks. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering. New York: ACM Press, 2010. 515−524. [doi: 10.1145/1806799.1806874]

[2] Hastings R, Joyce B. Purify: Fast detection of memory leaks and access errors. In: Proc. of the USENIX Conf. Berkeley: Usenix Association, 1992. 125−138.

[3] Omega: An instant leak detector tool for valgrind. 2016. http://www.brainmurders.eclipse.co.uk/omega.html

[4] Novark G, Berger ED, Zorn BG. Efficiently and precisely locating memory leaks and bloat. ACM SIGPLAN Notices, 2009,44(6): 397−407. [doi: 10.1145/1543135.1542521]

[5] Qin F, Lu S, Zhou Y. SafeMem: Exploiting ECC-memory for detecting memory leaks and memory corruption during production runs. In: Proc. of the 11th IEEE Int'l Symp. on High-Performance Computer Architecture. Piscataway: IEEE, 2005. 291−302. [doi: 10.1109/HPCA.2005.29]

[6] Cherem S, Princehouse L, Rugina R. Practical memory leak detection using guarded value-flow analysis. ACM SIGPLAN Notices, 2007,42(6):480−491. [doi: 10.1145/1273442.1250789]

[7] Heine DL, Lam MS. A practical flow-sensitive and context-sensitive C and C++ memory leak detector. ACM SIGPLAN Notices, 2003,38(5):168−181. [doi: 10.1145/780822.781150]

[8] Heine DL, Lam MS. Static detection of leaks in polymorphic containers. In: Proc. of the 28th Int'l Conf. on Software Engineering. New York: ACM Press, 2006. 252−261. [doi: 10.1145/1134285.1134321]

[9] Orlovich M, Rugina R. Memory leak analysis by contradiction. In: Proc. of the Static Analysis. Berlin, Heidelberg: Springer-Verlag, 2006. 405−424. [doi: 10.1007/11823230_26]

[10] Xie Y, Aiken A. Context-And path-sensitive memory leak detection. ACM SIGSOFT Software Engineering Notes, 2005,30(5): 115−125. [doi: 10.1145/1095430.1081728]

[11] Klocwork. The Klocwork static analysis tool. 2001~2016. http://www.klocwork.com/

[12] Coverity. The coverity static analysis tools. 2016. http://www.coverity.com/

[13] HP Fortify. 2016. http://www8.hp.com/us/en/software-solutions/application-security/index.html

[14] Godefroid P, Klarlund N, Sen K. DART: Directed automated random testing. ACM SIGPLAN Notices, 2005,40(6):213−223. [doi: 10.1145/1064978.1065036]

[15] Sen K, Marinov D, Agha G. CUTE: A concolic unit testing engine for C. ACM SIGSOFT Software Engineering Notes, 2005,30(5): 263−272. [doi: 10.1145/1095430.1081750]

[16] Kosmatov N. All-Paths test generation for programs with internal aliases. In: Proc. of the 19th IEEE Int'l Symp. on Software Reliability Engineering. Piscataway: IEEE, 2008. 147−156. [doi: 10.1109/ISSRE.2008.25]

[17] Xu G, Mitchell N, Arnold M, Rountev A, Schonberg E, Sevitsky G. Finding low-utility data structures. ACM SIGPLAN Notices, 2010,45(6):174−186. [doi: 10.1145/1809028.1806617]

[18] Xu G, Arnold M, Mitchell N, Rountev A, Sevitsky G. Go with the flow: Profiling copies to find runtime bloat. ACM SIGPLAN Notices, 2009,44(6):419−430. [doi: 10.1145/1543135.1542523]

[19] Heckman S, Williams L. On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques. In: Proc. of the 2nd ACM-IEEE Int'l Symp. on Empirical Software Engineering and Measurement. New York: ACM Press, 2008. 41−50. [doi: 10.1145/1414004.1414013]

[20] Arnold M, Vechev M, Yahav E. QVM: An efficient runtime for detecting defects in deployed systems. ACM SIGPLAN Notices, 2008,43(10):143−162. [doi: 10.1145/1449955.1449776]

[21] CREST: An automatic test generation tool for C. 2016. https://github.com/jburnim/crest

[22] Yices: An SMT solver. 2016. http://yices.csl.sri.com/

[23] Siemens. 2016. http://sir.unl.edu/

[24] GNU core utilities. 2001~2016. http://www.gnu.org/software/coreutils/coreutils.html

[25] Valgrind. 2016. http://valgrind.org/

[26] Cadar C, Dunbar D, Engler DR. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. of the 8th USENIX Symp. on Operating Systems Design and Implementation. Berkeley: Usenix Association, 2008. 209−224.

[27] Dor N, Rodeh M, Sagiv M. Checking cleanness in linked lists. In: Proc. of the Static Analysis. Berlin, Heidelberg: Springer-Verlag, 2000. 115−134. [doi: 10.1007/978-3-540-45099-3_7]

[28] Hackett B, Rugina R. Region-Based shape analysis with tracked locations. ACM SIGPLAN Notices, 2005,40(1):310−323. [doi: 10.1145/1047659.1040331]

[29] Sui Y, Ye D, Xue J. Static memory leak detection using full-sparse value-flow analysis. In: Proc. of the 2012 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2012. 254−264. [doi: 10.1145/2338965.2336784]

[30] Sui Y, Ye D, Xue J. Detecting memory leaks statically with full-sparse value-flow analysis. IEEE Trans. on Software Engineering, 2014,40(2):107−122. [doi: 10.1109/TSE.2014.2302311]

[31] Xu Z, Zhang J, Xu Z. Melton: A practical and precise memory leak detection tool for C programs. Frontiers of Computer Science, 2015,9(1):34−54. [doi: 10.1007/s11704-014-3460-8]

[32] Nethercote N, Seward J. Valgrind: A framework for heavyweight dynamic binary instrumentation. ACM SIGPLAN Notices, 2007, 42(6):89−100. [doi: 10.1145/1273442.1250746]

[33] Jung C, Lee S, Raman E, Pande S. Automated memory leak detection for production use. In: Proc. of the 36th Int'l Conf. on Software Engineering. New York: ACM Press, 2014. 825−836. [doi: 10.1145/2568225.2568311]

[34] Lee S, Jung C, Pande S. Detecting memory leaks through introspective dynamic behavior modelling using machine learning. In: Proc. of the 36th Int'l Conf. on Software Engineering. New York: ACM Press, 2014. 814−824. [doi: 10.1145/2568225.2568307]

[35] Bond MD, McKinley KS. Bell: Bit-Encoding online memory leak detection. ACM SIGPLAN Notices, 2006,41(11):61−72. [doi: 10.1145/1168918.1168866]

[36] Bond MD, McKinley KS. Tolerating memory leaks. ACM SIGPLAN Notices, 2008,43(10):109−126. [doi: 10.1145/1449955.1449774]

[37] Yan D, Xu G, Yang S, Rountev A. Leakchecker: Practical static memory leak detection for managed languages. In: Proc. of the Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization. New York: ACM Press, 2014. 87. [doi: 10.1145/2544137.2544151]

[38] Bond MD, McKinley KS. Leak pruning. ACM SIGARCH Computer Architecture News, 2009,37(1):277−288. [doi: 10.1145/1508284.1508277]

[39] Pauw WD, Sevitsky G. Visualizing reference patterns for solving memory leaks in Java. In: Proc. of the European Conf. on Object-Oriented Programming. Berlin, Heidelberg: Springer-Verlag, 1999. 116−134. [doi: 10.1007/3-540-48743-3_6]

[40] Hauswirth M, Chilimbi TM. Low-Overhead memory leak detection using adaptive statistical profiling. ACM SIGPLAN Notices, 2004,39(11):156−164. [doi: 10.1145/1037187.1024412]

[41] Jump M, McKinley KS. Cork: Dynamic memory leak detection for garbage-collected languages. ACM SIGPLAN Notices, 2007, 42(1):31−38. [doi: 10.1145/1190215.1190224]

[42] Mitchell N, Sevitsky G. LeakBot: An automated and lightweight tool for diagnosing memory leaks in large Java applications. In: Proc. of the Object-Oriented Programming (ECOOP 2003). Berlin, Heidelberg: Springer-Verlag, 2003. 351−377. [doi: 10.1007/978-3-540-45070-2_16]

[43] Rayside D, Mendel L. Object ownership profiling: A technique for finding and fixing memory leaks. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: ACM Press, 2007. 194−203. [doi: 10.1145/1321631.1321661]

[44] Tang Y, Gao Q, Qin F. LeakSurvivor: Towards safely tolerating memory leaks for garbage-collected languages. In: Proc. of the USENIX 2008 Annual Technical Conf. on Annual Technical Conf. Berkeley: USENIX Association, 2008. 307−320.

[45] Xu G, Bond MD, Qin F, Rountev A. LeakChaser: Helping programmers narrow down causes of memory leaks. ACM SIGPLAN Notices, 2011,46(6):270−282. [doi: 10.1145/1993316.1993530]

[46] Xu G, Rountev A. Precise memory leak detection for Java software using container profiling. In: Proc. of the 30th Int'l Conf. on Software Engineering. Piscataway: IEEE, 2008. 151−160. [doi: 10.1145/1368088.1368110]

[47] Gao Q, Xiong Y, Mi Y, Zhang L, Yang WK, Zhou ZP, Xie B, Mei H. Safe memory-leak fixing for C programs. In: Proc. of the 37th Int'l Conf. on Software Engineering, Vol.1. Piscataway: IEEE Press, 2015. 459−470. [doi: 10.1109/ICSE.2015.64]

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

[49] Clarke L. A system to generate test data and symbolically execute programs. IEEE Trans. on Software Engineering, 1976,SE-2(3):215−222. [doi: 10.1109/TSE.1976.233817]

[50] Giantsios A, Papaspyrou N, Sagonas K. Concolic testing for functional languages. In: Proc. of the 17th Int'l Symp. on Principles and Practice of Declarative Programming. New York: ACM Press, 2015. 137−148. [doi: 10.1145/2790449.2790519]

[51] Mesnard F, Payet É, Vidal G. Concolic testing in logic programming. Theory and Practice of Logic Programming, 2015,15(4-5):711−725. [doi: 10.1017/S1471068415000332]

[52] Chen T, Zhang X, Chen R, Yang B, Bai Y. Conpy: Concolic execution engine for python applications. In: Proc. of the Algorithms and Architectures for Parallel Processing. Springer Int'l Publishing, 2014. 150−163. [doi: 10.1007/978-3-319-11194-0_12]

[53] Gulavani BS, Henzinger TA, Kannan Y, Nori AV, Rajamani SK. SYNERGY: A new algorithm for property checking. In: Proc. of the 14th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. New York: ACM Press, 2006. 117−127. [doi: 10.1145/1181775.1181790]

[54] Vidal G. Concolic execution and test case generation in prolog. In: Proc. of the Logic-Based Program Synthesis and Transformation. Springer Int'l Publishing, 2014. 167−181. [doi: 10.1007/978-3-319-17822-6_10]

[55] Ge X, Taneja K, Xie T, Tillmann N. DyTa: Dynamic symbolic execution guided with static verification results. In: Proc. of the 33rd Int'l Conf. on Software Engineering. New York: ACM Press, 2011. 992−994. [doi: 10.1145/1985793.1985971]

[56] Csallner C, Smaragdakis Y, Xie T. DSD-Crasher: A hybrid analysis tool for bug finding. ACM Trans. on Software Engineering and Methodology, 2008,17(2):8. [doi: 10.1145/1348250.1348254]

[57] Zhang S, Saff D, Bu Y, Ernst MD. Combined static and dynamic automated test generation. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2011. 353−363. [doi: 10.1145/2001420.2001463]

[58] Babić D, Martignoni L, McCamant S, *et al*. Statically-Directed dynamic automated test generation. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2011. 12−22. [doi: 10.1145/2001420.2001423]

[59] Taneja K, Xie T, Tillmann N, Halleux JD. eXpress: Guided path exploration for efficient regression test generation. In: Proc. of the 2011 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2011. 1−11. [doi: 10.1145/2001420.2001422]

[60] Cui H, Hu G, Wu J, Yang J. Verifying systems rules using rule-directed symbolic execution. ACM SIGPLAN Notices, 2013,48(4):329−342. [doi: 10.1145/2499368.2451152]

[61] Li Y, Su Z, Wang L, Li X. Steering symbolic execution to less traveled paths. ACM SIGPLAN Notices, 2013,48(10):19−32. [doi: 10.1145/2544173.2509553]

[62] Cui Z, Wang L, Li X. Target-Directed concolic testing. Jisuanji Xuebao/Chinese Journal of Computers, 2011,34(6):953−964 (in Chinese with English abstract).

[63] Ruthruff JR, Penix J, Morgenthaler JD, Elbaum S, Rothermel G. Predicting accurate and actionable static analysis warnings: An experimental approach. In: Proc. of the 30th Int'l Conf. on Software Engineering. New York: ACM Press, 2008. 341−350. [doi: 10.1145/1368088.1368135]

[64] Heckman S, Williams L. On establishing a benchmark for evaluating static analysis alert prioritization and classification techniques. In: Proc. of the 2nd ACM-IEEE Int'l Symp. on Empirical Software Engineering and Measurement. New York: ACM Press, 2008. 41−50. [doi: 10.1145/1414004.1414013]

[65] Kim S, Ernst MD. Which warnings should I fix first. In: Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. New York: ACM Press, 2007. 45−54. [doi: 10.1145/1287624.1287633]

[66] Kremenek T, Ashcraft K, Yang J, Engler D. Correlation exploitation in error ranking. ACM SIGSOFT Software Engineering Notes, 2004,29(6):83−93. [doi: 10.1145/1041685.1029909]

[67] Kremenek T, Engler D. Z-Ranking: Using statistical analysis to counter the impact of static analysis approximations. In: Proc. of the Static Analysis. Berlin, Heidelberg: Springer-Verlag, 2003. 295−315. [doi: 10.1007/3-540-44898-5_16]

[68] Boogerd C, Moonen L. Prioritizing software inspection results using static profiling. In: Proc. of the 6th IEEE Int'l Workshop on Source Code Analysis and Manipulation. Piscataway: IEEE, 2006. 149−160. [doi: 10.1109/SCAM.2006.22]

[69] Kim S, Ernst MD. Prioritizing warning categories by analyzing software history. In: Proc. of the 4th Int'l Workshop on Mining Software Repositories. IEEE Computer Society, 2007. 27. [doi: 10.1109/MSR.2007.26]

[70] Dillig I, Dillig T, Aiken A. Automated error diagnosis using abductive inference. ACM SIGPLAN Notices, 2012,47(6):181−192. [doi: 10.1145/2345156.2254087]

[71] Ha J, Rossbach CJ, Davis JV, Roy I, Ramadan HE, Porter DE, Chen DL, Witchel E. Improved error reporting for software that uses black-box components. ACM SIGPLAN Notices, 2007,42(6):101−111. [doi: 10.1145/1273442.1250747]

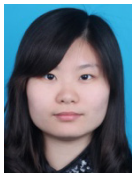
[72] Tripp O, Guarnieri S, Pistoia M, Aravkin A. ALETHEIA: Improving the usability of static security analysis. In: Proc. of the 2014 ACM SIGSAC Conf. on Computer and Communications Security. New York: ACM Press, 2014. 762−774. [doi: 10.1145/2660267.2660339]

[73] Chimdyalwar B, Darke P, Chavda A, Vaghani S, Chauhan A. Eliminating static analysis false positives using loop abstraction and bounded model checking. In: Proc. of the Formal Methods (FM 2015). Springer Int'l Publishing, 2015. 573−576. [doi: 10.1007/978-3-319-19249- 9_35]

[74] Li M, Chen Y, Wang L, Xu G. Dynamically validating static memory leak warnings. In: Proc. of the 2013 Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2013. 112−122. [doi: 10.1145/2483760.2483778]

附中文参考文献:

[62] 崔展齐,王林章,李宣东.一种目标制导的混合执行测试方法.计算机学报,2011,34(6):953−964.



李筱(1992－),女,山东烟台人,硕士生,主要研究领域为软件测试,测试自动化,移动应用测试.

周严(1991－),男,硕士生,主要研究领域为软件自动化测试,代码静态分析.

李孟宸(1988－),男,硕士,主要研究领域为软件自动化测试,程序分析.

陈园军(1988－),男,硕士,主要研究领域为软件测试,嵌入式系统,操作系统内存模型.

XU Guo-Qing (1981－),男,博士,助理教授,主要研究领域为程序设计语言,编译器优化,程序分析,系统软件,分布式系统,大数据分析.

王林章(1973－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为模型驱动的软件测试与验证,安全测试,软件测试自动化.

李宣东(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为软件建模与分析,软件测试与验证.