Table6 Comparison of runtime protection approaches

表 6 运行时防护方法对比

分类	被动防护				主动防护			
代表	插入 canary 值	存储 RETADDR 值	指针前后加 guardzone	低脂指针	更换动态链接库	加密指针型数据	随机化内存地址	去堆栈布局可预测性
技术特点	通过扩展编译器的方法,在函数调用栈的 RETADDR 附近添加一个 canary 值,确保调用栈的 RETADDR 值不被修改	1. RETADDR 值被存储到一个全局的表中,跳转执行前,进行比较. 2. 函数指针进行解引用之前,都要判断该指针是否指向代码区[46]	在内存对象前后都插入 guardzone.当危险指针解引用对应的值等于 guardzone,提示访问错误	分配内存时,把缓冲区基地址信息、缓冲区大小信息等元信息映射到分配的指针上,高效地实现了对缓冲区的边界检测[48]	将含有类似 Strcpy 函数等容易发生缓冲区溢出的动态函数链接库替换成安全的动态函数链接库	对指针型数据进行加密后再将其存放在内存中,指针解引用前再在寄存器中解密	通过加 padding 的方式随机化堆、栈、数段等内存的位置	通过对逻辑栈帧先分割再置换的方式更为彻底地打乱栈帧内变量和变量的分配位置信息、栈帧和栈帧间的位置信息,达到不让逻辑上相邻的栈空间内存在物理上相邻的目的
主要优点	能够防御大多数需要修改函数调用栈的溢出攻击	1. 能够防御大多数需要修改 RETADDR 的溢出攻击. 2. 也能防御所有更改函数指针的攻击	1. 能够防御多数更改类函数指针这类攻击 2. 由于静态分析的辅助,运行时开销较低	1. 对堆栈内存空间的保护与其他技术相比较为完善 2. 对内存改变较小,内存开销较小	保障了调用已替换的 C 库函数时,不会发生溢出	即使有溢出发生,也不会跳转到攻击者期待的攻击代码位置	1. 提高攻击难度 2. 运行开销较小	对跨栈帧攻击和栈帧内攻击防御效果均较好
主要缺点	1. 没有防御破坏堆的攻击方式 2. 没有防御更改函数指针的攻击方式 3. 没有防御针对 setjmp/longjmp 的攻击方式 4. 运行开销较大 5. 没有从根本上解决问题	1. 没有防御破坏堆的攻击方式 2. 没有防御修改 EBP 的攻击 3. 没有防御针对 setjmp/longjmp 的攻击方式 4. 运行开销较大 5. 没有从根本上解决问题	1. 忽略了访问对象值与 Guardzone 值相等的情况 2. 没有从根本上解决问题	1. 元信息在传递可能会有差错,导致漏报 2. 没有对数据段上的内存进行保护 3. 没有从根本上解决问题	1. 除了调用已替换的库函数以外,仍有很多溢出手段可以改变 RETADDR 的值和函数指针的值 2. 运行开销较大 3. 没有从根本上解决问题	1. 运行开销较大 2. 没有从根本上解决问题	1. 没有防住所有的溢出攻击,例如对栈帧内溢出防护效果一般 2. 没有从根本上解决问题	1. 只对栈上的空间进行保护 2. 在全面防护下的时空开销较大 3. 没有从根本上解决问题

3 分析讨论

本文所介绍的缓冲区溢出分析技术分类概览如图 2 所示,以下分别对上述技术进行分析讨论.

缓冲区溢出的漏洞触发点比较明确,对应为发生越界解引用的语句.现有的静态检测技术可以将缓冲区溢出越界解引用的位置以及缓冲区定义的位置和变量传递的位置等信息提供给软件的开发者,有利于从根本消除缓冲区溢出漏洞.但主要问题有:

(1) 缓冲区溢出相对于其他类型的漏洞,分析的精确度在很大程度上取决于区间分析的准确度.由于静态检测技术的不可判定性,且无法获得程序运行时信息,因此,现有静态技术只能对访问缓冲区的索引区间进行上近似或下近似,所以难免会产生误报和漏报.

(2) 在保证误报率和漏报率较低的同时,要高效地对缓冲区溢出漏洞进行检测,需要对缓冲区指针以及缓冲区的索引分别建立静态依赖图,追踪值的传递.然而在大型程序中,跨函数的值传递普遍存在,且依赖关系复杂,如何保证吞吐量足够大也是一大技术难点[55].

针对上述问题,本文有如下两个观点:(1) 因为造成严重后果的缓冲区溢出漏洞通常是攻击者通过外部输入来获得系统控制权限的,因此污染传播这种针对外部输入数据的传播路径进行跟踪的技术在静态检测缓冲区溢出漏洞这一领域扮演重要的角色,该技术与符号执行或抽象解释(比如区间分析)紧密结合或许可以更准确地检测缓冲区溢出漏洞;(2) 近年来,符号执行、抽象解释、污染传播等传统静态检测技术发展得较为成熟,或许科研人员可以转换思路,暂时不致力于研究提高传统静态技术本身,而是将新兴技术融入传统静态检测技术,比如将机器学习算法用于特征分类,辅助传统静态检测技术,可能会进一步提高缓冲区溢出漏洞静态检测的精度和效率.

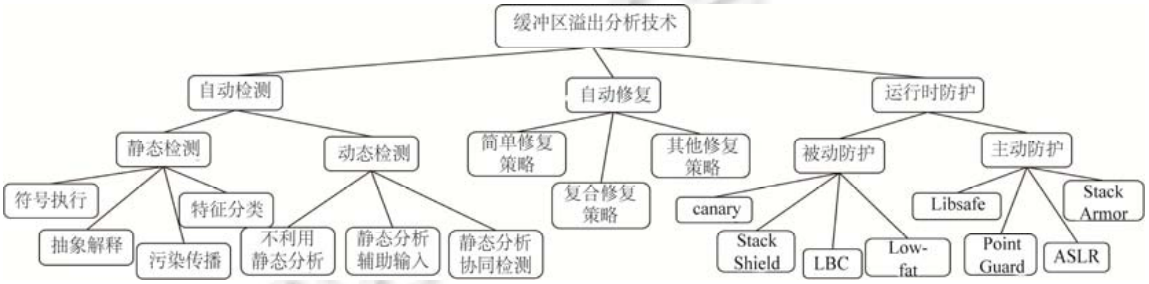


Fig.2 Overview of buffer overflow analysis technologies classification

图 2 缓冲区溢出分析技术分类概览

动态测试基于测试用例生成技术产生测试用例,通过运行程序发现缓冲区溢出漏洞.所发现的缓冲区溢出漏洞一定是真实的,不存在误报.但由于要产生触发缓冲区溢出的测试用例,需要保证访问缓冲区的索引大于缓冲区的大小.为了保证这种值的正确产生和条件的准确判断,动态测试方法在生成测试用例时通常采用符号执行或者遗传算法,因此有生成及运行测试用例时性能开销较大的问题.此外,由于无法做到让测试用例完全覆盖程序中所有可能的缓冲区溢出的程序点,有漏报率较高的缺点.我们认为,如果要更好地提高动态测试的检测能力,应更多地与静态分析结合,即采用静态分析辅助输入或者静态分析协同测试的方法.利用静态分析降低漏报,提高动态测试的覆盖率,利用动态测试去触发缓冲区漏洞,降低误报.例如,将 fuzzing 测试与静态分析结合:Haller 等人利用静态分析得到节点可达性的约束,用其来指导 fuzzing 生成测试用例[34];Mouzarani 等人在 Haller 等人的基础上,把溢出的触发条件与可达约束取交来指导 fuzzing 生成测试用例,比 Haller 等人的方法更 smart,效果更好[56].静态分析能够给出程序本身的部分约束条件,测试能够获得程序的运行信息,动态测试和静态分析恰好能够互补各自的不足.而缓冲区溢出漏洞检测对程序本身的约束信息和运行时信息都需要,动态测试若能恰到好处地借助静态分析,效果就会更理想,即 smarter 甚至优于 smartest.

对软件漏洞的自动修复是近年来新崛起的研究领域,该方法的优点是不但能够发现程序中的软件漏洞,还能自动地修复,使软件能够更安全、可靠地运行.目前,在缓冲区溢出方面的修复有较大问题,使用测试用例验证缓冲区溢出的修复结果.然而测试用例常常不完全,描述的很可能只是缓冲区索引越界的一个子集.因此,能够通过测试用例不代表修复了缓冲区溢出缺陷,无法保障修复的正确性,还可能引入新的缓冲区溢出或逻辑缺陷.软件发布后的自动修复有较大的运行开销,不适合在对实时性要求较高的环境中使用.我们认为,自动修复领域也许会由加入条件判断的单一修复策略方式,逐渐地向采用复合修复策略和采用其他修复策略转化.此外,生物

学中的基因移植[38]、遗传变异[42]等思想也将广泛应用于缓冲区溢出漏洞的修复.随着人工智能技术的崛起,采用其他修复策略的缓冲区溢出漏洞修复技术很有可能会有更好的修复效果.

运行时防护技术能够在一定程度上实时地保障软件不被溢出攻击,具有很高的应用价值.但其只能防御特定模式的溢出攻击,没有从根本上解决问题;另外,检测到溢出攻击时,部分技术采用的应对策略是停止程序运行,虽然可以防止缓冲区溢出的发生,但也较大地影响了软件的可用性;并且,运行时防护技术所增加的运行开销也往往较大.运行时防护技术需要更新颖的方法,尽可能少地修改程序执行环境,以减少正常程序操作期间的开销和不必要的副作用[57].被动防护通常是围绕 EBP、RETADDR、函数指针的完整性或者是缓冲区访问位置需在缓冲区范围内这一性质的完整性展开.目前最新的研究成果低脂指针[49]在对堆内存、栈内存的防护方面已经取得还不错的效果,值得进一步思考与完善的是,如何将该思想扩展到数据段上.主动防护通常是采取主动替换函数链接库、加密函数指针、随机化内存地址、去堆栈布局可预测性等方面,围绕内存的机密性和可用性展开.该技术目前较新的代表是 StackArmor[54],该方法对栈空间位置分配打乱得较为彻底,对栈缓冲区溢出的防护效果较为理想,值得进一步思考和完善的是如何扩充该思想,对堆上及数据段内存的分配规律也进行混淆.此外,被动防护技术与主动防护技术的结合也可能会起到较好的防护效果,但二者结合的内存开销该如何被降到合理范围内,也值得科研人员进一步探讨.

表 7 是上述 4 种分析方法的对比.

Table 7 Comparison of buffer overflow analysis approaches

表 7 缓冲区溢出分析方法对比

分析方法	漏洞自动检测——静态检测	漏洞自动检测——动态测试	漏洞自动修复	漏洞运行时防护
技术特点	在不运行软件的前提下检测程序中存在的缓冲区溢出漏洞[16]	从执行域中恰当地选取一组有限的测试用例来运行程序[30]	不仅能够发现程序中的软件漏洞,还能自动地修复	把软件放在一个安全的保护罩里去运行,实时地进行防护
技术难点	1. 如何保证误报率、漏报率均较低 2. 如何在保证误报率、漏报率均较低的同时,保证吞吐量足够大	1. 如何生成覆盖率高的测试用例 2. 如何生成能命中要害、触发缓冲区溢出漏洞发生的测试用例	1. 如何降低漏报率 2. 如何确保自动修复后,漏洞真正得以排除 3. 如何确保自动修复后,程序的原始语义不被修改	1. 如何降低运行开销 2. 如何防御尽可能多的攻击模式 3. 发现攻击后,如何处理以保障程序正常运行
主要优点	1. 在软件发布前发现漏洞 2. 可以将漏洞的位置信息汇报给软件的维护者 3. 有利于软件开发者从根本上消除缓冲区溢出漏洞 4. 代码执行路径覆盖率较高	1. 在软件发布前发现漏洞 2. 发现的疑似漏洞一定是缓冲区溢出漏洞,无误报	1. 发现的疑似漏洞一定是缓冲区溢出漏洞,无误报 2. 可以在软件发布前进行修复,降低缓冲区溢出漏洞的存在可能 3. 可以在软件发布后进行实时修复,降低运行过程中宕机及被攻击的风险	1. 可在软件发布后对其进行实时防护 2. 发现的疑似漏洞一定是缓冲区溢出漏洞,无误报
主要缺点	无法获得程序运行时信息,难免会产生误报和漏报	1. 生成及运行测试用例时性能开销较大 2. 无法做到完全覆盖程序中所有可执行路径,漏报率较高	1. 漏报率较高 2. 修复后无法保证漏洞真正得到排除 3. 无法保证修复后,程序的原始语义不被修改	1. 只能防御特定模式的溢出攻击,没有从根本上解决问题 2. 检测到溢出攻击时部分方法会停止程序运行,较大地影响了软件的可用性 3. 运行开销往往较大

4 总结与展望

本文首先介绍了缓冲区溢出漏洞危害的严重性和广泛性;然后,从如何利用缓冲区溢出漏洞的角度,依次介

绍了缓冲区溢出漏洞的定义、系统内存组织方式和缓冲区溢出攻击方式;接下来,基于对缓冲区漏洞的理解,根据调研结果将缓冲区溢出分析技术分为 3 类:漏洞自动检测技术、漏洞自动修复技术、漏洞运行时防护技术,并对每一类技术进行了介绍;最后,本文对这 3 类技术进行了分析讨论.

根据调研结果,我们认为,缓冲区溢出分析领域未来有 3 个可能的研究方向.

4.1 对二进制代码进行分析

因为很多商业软件不会把源代码暴露给外界,所以对二进制代码的缓冲区溢出漏洞自动检测与修复,很有可能是下一步的研究热点.2016 年 8 月,美国举办的 CGC 挑战赛目标为二进制代码的自动发现、利用及修复,其中一大类漏洞类型为缓冲区溢出[58].自 2009 年以来,程序缺陷修复成为软件工程领域研究的一大热点,涌现出很多不同的自动缺陷修复技术.目前主流技术针对一般类型的缺陷,通过运行测试过滤掉错误的补丁[59,60].针对特定类型的缺陷也有一些研究,如内存泄漏[61]、数据竞争[62]等.但是主要针对源代码进行分析,而对二进制代码漏洞进行检测和修复的研究较少.在缓冲区漏洞这一重要的程序缺陷上,对二进制代码的分析将会更有价值.

对二进制代码的缓冲区溢出漏洞的检测与修复主要有两方面的挑战:第一是对二进制代码进行反汇编,由于很多二进制代码进行了加壳,解析过程有较大困难;第二是保证对缓冲区溢出漏洞修复的正确性,如果一种自动修复方法不能保证修复的正确性,开发与测试人员则不会使用该自动修复方法.同时,必须保证修复补丁不会影响程序的执行效率,因此要尽可能地只在必要的程序位置生成修复.

4.2 结合机器学习算法进行分析

随着深度学习算法和人工智能领域近年来的快速发展,科研人员将越来越关注如何结合机器学习和人工智能算法来实现更准确的缓冲区溢出检测.Mou 等人[63]使用深度学习的方法进行研究程序分类问题.Rahul 等人[64]将深度学习应用于程序缺陷修复.我们认为,对缓冲区溢出的检测也将可以使用机器学习技术.

缓冲区溢出有一定的模式,例如对数组、指针的声明、对数组下标的赋值与运算以及对缓冲区溢出位置的数据操作.对缓冲区溢出的检测可以归结为某段代码有无缓冲区溢出漏洞的有无问题,因此转化为机器学习中的二元分类问题.通过使用机器学习算法,可以学习在不同的上下文中出现缓冲区溢出漏洞的模式,进而提高精度.此外,机器学习算法可以用于检测结果的筛选,并和程序分析方法结合使用.例如,首先,通过程序分析方法提供一个可能的缓冲区溢出漏洞列表;再使用机器学习方法精化分析结果,过滤掉检测结果中的误报和漏报.

4.3 综合利用多种技术进行分析

如何综合使用漏洞自动检测、漏洞自动修复、漏洞运行时防护这 3 类技术,以获得更准确的检测结果和更低的性能开销,也值得科研人员、工程技术人员进行深入的研究.目前的研究分别在静态检测、动态测试、漏洞自动修复、漏洞运行时防护等方面对缓冲区溢出漏洞进行研究.这些方法各有优劣,可以结合使用,使对缓冲区溢出漏洞检测的精度更高.但是在综合使用的过程中,还可能有各种问题需要解决,例如,如何有效地筛选动态测试中使用的测试用例,使自动修复获得更高的速率;如何使用静态分析的结果帮助运行时防护更有针对性地进行计算,从而节省运行时的防护开销.此外,3 类检测技术若结合使用势必带来更多潜在的效率问题,如何在各个技术的使用阶段整合资源,从而达到最优的并行效果,也是值得研究的问题.

References:

[1] The OWASP top ten 2004. https://www.owasp.org/index.php/Top_10_2004

[2] CWE. 2011. http://cwe.mitre.org/

[3] National Computer Network Intrusion Protection Center. Analysis of ten important security breaches in November 2014, 2014 (in Chinese).

[4] National Information Security Vulnerability Library (in Chinese). 2016. http://www.cnnvd.org.cn/

[5] Ye T, Zhang LM, Wang LZ, Li XD. An empirical study on detecting and fixing buffer overflow bugs. In: Proc. of the ICST 2016. 2016. 91−101. [doi: 10.1109/ICST.2016.21]

[6] CVE. 2016. http://cve.mitre.org/

[7] Piromsopa K, Enbody RJ. Buffer-Overflow protection: The theory. In: Proc. of the 2006 Int'l Conf. on Electro/Information Technology. 2006. 454−458. [doi: 10.1109/EIT.2006.252128]

[8] Chen K, Wagner D. Large-Scale analysis of format string vulnerabilities in debian linux. In: Proc. of the Workshop on Programming Languages and Analysis for Security (PLAS 2007). San Diego, 2007. 75−84. [doi: 10.1145/1255329.1255344]

[9] Ahmad K. Classification and prevention techniques of buffer overflow attacks. In: Proc. of the 5th National Conf.; Indiacom-2011 Computing for Nation Development. 2011.

[10] Bishop M, Engle S, Howard D, Whalen S. A taxonomy of buffer overflow characteristics. IEEE Trans. on Dependable and Secure Computing, 2012,9(3):305−317. [doi: 10.1109/TDSC.2012.10]

[11] Wilander J, Kamkar M. A comparison of publicly available tools for dynamic buffer overflow prevention. In: Proc. of the NDSS 2003. 2003. 149−162.

[12] Novark G, Berger ED. DieHarder: Securing the heap. In: Proc. of the 17th ACM Conf. on Computer and Communications Security. ACM Press, 2010. 573−584. [doi: 10.1145/1866307.1866371]

[13] Sotirov A. Heap Feng Shui in Javascript. In: Black Hat Europe 2007. 2007.

[14] Padmanabhuni B, Tan HBK. Defending against buffer-overflow vulnerabilities. Computer, 2011,44(11):53−60. [doi: 10.1109/MC.2011.229]

[15] Grieco G, Mounier L, Potet ML, Rawat S. A stack model for symbolic buffer overflow exploitability analysis. In: Proc. of the ICST Workshops 2013. 2013. 216−217. [doi: 10.1109/ICSTW.2013.33]

[16] Mei H, Wang QX, Zhang L, Wang J. Software analysis: A road map. Chinese Journal of Computers, 2009,32(9):1697−1708 (in Chinese with English abstract).

[17] Wagner D, Foster J, Brewer E, Aiken A. A first step towards automated detection of buffer overrun vulnerabilities. In: Proc. of the Network and Distributed System Security Symp. San Diego, 2000. 3−17.

[18] Larochelle D, Evans D. Statically detecting likely buffer overflow vulnerabilities. In: Proc. of the 10th Usenix Security Symp. Usenix, 2001.

[19] Yang ZH, Gong YZ, Xiao Q, Wang YW. The application of interval computation in software testing based on defect pattern. Journal of Computer-aided Design & Computer Graphic, 2008,20(12):1630−1635 (in Chinese with English abstract).

[20] Xiao Q, Gong YZ, Yang ZH, Jin DH, Wang YW. Path sensitive static defect detecting method. Ruan Jian Xue Bao/Journal of Software, 2010,21(2):209−217 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3782.htm [doi: 10.3724/SP.J.1001.2010.03782]

[21] Wang YW, Yao XH, Gong YZ, Yang ZH. A method of buffer overflow detection based on static code analysis. Journal of Computer Research and Development, 2012,49(4):839−845 (in Chinese with English abstract).

[22] Brat G, Navas JA, Shi N, Venet A. IKOS: A framework for static analysis based on abstract interpretation. In: Proc. of the SEFM 2014. 2014. 271−277. [doi: 10.1007/978-3-319-10431-7_20]

[23] Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. In: Proc. of the CGO 2004. 2004. [doi: 10.1109/CGO.2004.1281665]

[24] Le W, Soffa ML. Marple: A demand-driven pathsensitive buffer overflow detector. In: Proc. of the 16th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Atlanta, 2008. 272−282. [doi: 10.1145/1453101.1453137]

[25] Le W, Soffa ML. Marple: Detecting faults in path segments using automatically generated analyses. ACM Trans. on Software Engineering and Methodology, 2013,22(3):18:1−18:38. [doi: 10.1145/2491509.2491512]

[26] Ding S, Tan HBK, Liu KP, Chandramohan M, Zhang HY. Detection of buffer overflow vulnerabilities in C/C++ with pattern based limited symbolic evaluation. In: Proc. of the COMPSAC Workshops 2012. 2012. 559−564. [doi: 10.1109/COMPSACW.2012.103]

[27] Gao F, Chen T, Wang Y, Situ L, Wang L, Li X. Carraybound: Static array bounds checking in C programs based on taint analysis. In: Proc. of the 8th Asia-Pacific Symp. on Internetware. ACM Press, 2016. 81−90. [doi: 10.1145/2993717.2993724]

[28] Padmanabhuni BM, Tan HBK. Predicting buffer overflow vulnerabilities through mining light-weight static code attributes. In: Proc. of the ISSRE Workshops 2014. 2014. 317−322. [doi: 10.1109/ISSREW.2014.26]

[29] B Padmanabhuni BM, Tan HBK. Buffer overflow vulnerability prediction from x86 executables using static analysis and machine learning. In: Proc. of the COMPSAC 2015. 2015. 450−459. [doi: 10.1109/COMPSAC.2015.78]

[30] Patton P. Software Testing. 2nd ed., SAMS, 2005.

[31] Rawat S, Mounier L. Offset-Aware mutation based fuzzing for buffer overflow vulnerabilities: Few preliminary results. In: Proc. of the ICST Workshops 2011. 2011. 531−533. [doi: 10.1109/ICSTW.2011.9]

[32] Wang WH, Lei Y, Liu DG, Kung DC, Csallner C, Zhang DZ, Kacker R, Kuhn R. A combinatorial approach to detecting buffer overflow vulnerabilities. In: Proc. of the DSN 2011. 2011. 269−278. [doi: 10.1109/DSN.2011.5958225]

[33] Lei Y, Kacker R, Kuhn DR, Okun V, Lawrence J. IPOG/IPOG-D: Efficient test generation for multi-way combinatorial testing. Software Testing, Verification and Reliability, 2008,18(3):125−148. [doi: 10.1002/stvr.381]

[34] Haller I, Slowinska A, Neugschwandtner M, Bos H. Dowsing for overflows: A guided fuzzer to find buffer boundary violations. In: Proc. of the USENIX Security 2013. 2013. 49−64.

[35] Padmanabhuni BM, Tan HBK. Light-Weight rule-based test case generation for detecting buffer overflow vulnerabilities. In: Proc. of the AST@ICSE 2015. 2015. 48−52. [doi: 10.1109/AST.2015.17]

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

[37] Padmanabhuni BM, Tan HBK. Auditing buffer overflow vulnerabilities using hybrid static-dynamic analysis. In: Proc. of the COMPSAC 2014. 2014. 394−399. [doi: 10.1109/COMPSAC.2014.62]

[38] Sidiroglou-Douskos S, Lahtinen E, Long F, Rinard, M. Automatic error elimination by horizontal code transfer across multiple applications. ACM SIGPLAN Notices, 2015,50(6):43−54. [doi: 10.1145/2813885.2737988]

[39] Sidiroglou-Douskos S, Lahtinen E, Rinard M. Automatic discovery and patching of buffer and integer overflow errors. Technical Report, MIT-CSAIL-TR-2015-018, Boston: Massachusetts Institute of Technology, 2015.

[40] Shaw A, Doggett D, Hafiz M. Automatically fixing C buffer overflows using program transformations. In: Proc. of the DSN 2014. 2014. 124−135. [doi: 10.1109/DSN.2014.25]

[41] Gao F, Wang L, Li X. BovInspector: Automatic inspection and repair of buffer overflow vulnerabilities. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM Press, 2016. 786−791. [doi: 10.1145/2970276.2970282]

[42] Arcuri A. On the automation of fixing software bugs. In: Proc. of the CSE Companion 2008, Vol.I. 2008. 1003−1006. [doi: 10.1145/1370175.1370223]

[43] Chen G, Jin H, Zou DQ, Zhou BB, Liang ZK, Zheng WD, Shi XH. SafeStack: Automatically patching stack-based buffer overflow vulnerabilities. IEEE Trans. on Dependable and Secure Computing, 2013,10(6):368−379. [doi: 10.1109/TDSC.2013.25]

[44] Cowan C. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In: Proc. of the USENIX Security 1998. 1998. 63−78.

[45] Etoh H. GCC extension for protecting applications from stack-smashing attacks. 2000. http://www.trl.ibm.com/projects/security/ssp/

[46] Vendicator. Stack shield technical info file v0.7. 2001. http://www.angelfire.com/sk/stackshield/

[47] Hasabnis N, Misra A, Sekar R. Light-Weight bounds checking. In: Proc. of the CGO 2012. 2012. 135−144. [doi: 10.1145/2259016.2259034]

[48] Duck GJ, Yap RHC. Heap bounds protection with low fat pointers. In: Proc. of the 25th Int'l Conf. on Compiler Construction (CC 2016). ACM Press, 2016. 132−142. [doi: 10.1145/2892208.2892212]

[49] Duck GJ, Yap RHC, Cavallaro L. Stack bounds protection with low fat pointers. In: Proc. of the Symp. on Network and Distributed System Security. 2017.

[50] Baratloo A, Singh N, Tsai T. Transparent run-time defense against stack smashing attacks. In: Proc. of the 2000 USENIX Technical Conf. San Diego, 2000. 251−262.

[51] Cowan C, Beattie S, Johansen J, Wagle P. PointGuard™: Protecting pointers from buffer overflow vulnerabilities. In: Proc. of the USENIX Security 2003. 2003. 91−104.

[52] Bhatkar S, DuVarney DC, Sekar R. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In: Proc. of the USENIX Security 2003. 2003.

[53] Use of ASLR, NX. 2008. http://blogs.msdn.com/david_leblanc/archive/2008/03/14/use-of-aslr-nx-etc.aspx

[54] Chen X, Slowinska A, Andriesse D, Bos H, Giuffrida C. StackArmor: Comprehensive protection from stack-based memory error vulnerabilities for binaries. In: Proc. of the NDSS. 2015. [doi: 10.14722/ndss.2015.23248]

[55] Shahriar H, Zulkernine M. Classification of static analysis-based buffer overflow detectors. In: Proc. of the SSIRI (Companion) 2010. 2010. 94−101. [doi: 10.1109/SSIRI-C.2010.28]

[56] Mouzarani M, Sadeghiyan B, Zolfaghari M. Smart fuzzing method for detecting stack-based buffer overflow in binary codes. IET Software, 2016,10(4):96−107. [doi: 10.1049/iet-sen.2015.0039]

[57] Shahriar H, Zulkernine M. Classification of buffer overflow vulnerability monitors. In: Proc. of the ARES 2010. 2010. 519−524. [doi: 10.1109/ARES.2010.15]

[58] CGC. 2016. https://www.darpa.mil/program/cyber-grand-challenge

[59] Le Goues C, Nguyen TV, Forrest S, Weimer W. Genprog: A generic method for automatic software repair. IEEE Trans. on Software Engineering, 2012,38(1):54−72. [doi: 10.1109/TSE.2011.104]

[60] Long F, Rinard M. Staged program repair with condition synthesis. In: Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM Press, 2015. 166−178. [doi: 10.1145/2786805.2786811]

[61] Li X, Zhou Y, Li MC, Chen YJ, Wang LZ, Li XD. Automatically validating static memory leak warnings for C/C++ programs. Ruan Jian Xue Bao/Journal of Software, 2017,28(4):827−844 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5189.htm [doi: 10.13328/j.cnki.jos.005189]

[62] Lu S, Tucek J, Qin F, Zhou YY. AVIO: Detecting atomicity violations via access interleaving invariants. ACM SIGARCH Computer Architecture News, 2006,34(5):37−48. [doi: 10.1145/1168919.1168864]

[63] Mou LL, Li G, Zhang L, Wang T, Jin Z. Convolutional neural networks over tree structures for programming language processing. In: Proc. of the 30th AAAI Conf. on Artificial Intelligence (AAAI). 2016. 1287−1293.

[64] Gupta R, Pal S, Kanade A, Shevade S. DeepFix: Fixing common C language errors by deep learning. In: Proc. of the AAAI 2017. 2017. 1345−1351.

附中文参考文献:

[3] 中国科学院大学国家计算机网络入侵防范中心.2014 年 11 月十大重要安全漏洞分析.2014.

[4] 国家信息安全漏洞库. http://www.cnnvd.org.cn/

[16] 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):1697−1708.

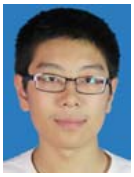
[19] 杨朝红,宫云战,肖庆,王雅文.基于缺陷模式的软件测试中的区间运算应用.计算机辅助设计与图形学学报,2008,20(12):1630−1635.

[20] 肖庆,宫云战,杨朝红,金大海,王雅文.一种路径敏感的静态缺陷检测方法.软件学报,2010,21(2):209−217. http://www.jos.org.cn/1000-9825/3782.htm [doi: 10.3724/SP.J.1001.2010.03782]

[21] 王雅文,姚欣洪,宫云战,杨朝红.一种基于代码静态分析的缓冲区溢出检测算法.计算机研究与发展,2012,49(4):839−845.

[61] 李筱,周严,李孟宸,陈园军,王林章,李宣东.C/C++程序静态内存泄漏警报自动确认方法.软件学报,2017,28(4):827−844. http://www.jos.org.cn/1000-9825/5189.htm [doi: 10.13328/j.cnki.jos.005189]

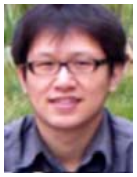
邵思豪(1993－),男,辽宁沈阳人,博士生,主要研究领域为信息安全,软件保障.



马骁(1994－),男,博士生,主要研究领域为程序分析.

高庆(1989－),男,博士,助理研究员,主要研究领域为软件分析,信息安全.

张世琨(1969－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件工程,信息安全.



马森(1989－),男,博士,助理研究员,主要研究领域为软件分析,信息安全.

胡津华(1961－),女,高级工程师,主要研究领域为信息安全.

段富尧(1995－),男,博士生,主要研究领域为程序分析.