计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(1):1−11 [doi: 10.15888/j.cnki.csa.008848]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

二进制代码安全分析综述①

周忠君[1], 董荣朝[1], 蒋金虎[2], 张为华[2]

[1](复旦大学 软件学院, 上海 200433)
[2](复旦大学 上海市数据科学重点实验室, 上海 200433)
通信作者: 张为华, E-mail: zhangweihua@fudan.edu.cn

摘　要: 近几十年来, 计算机硬件性能和软件规模技术已不同以往, 其承载了人类社会生活生产的方方面面. 计算机技术的飞速发展, 也带来了人们对程序安全问题的关注. 由于市面上存在着较多的遗留软件, 这些软件无人维护且缺乏源代码支持, 其安全性令人担忧, 而二进制分析技术被用来解决该类软件问题. 二进制分析技术根据其检测方式不同可分为: 基于静态的二进制代码分析技术、基于动态的二进制代码分析技术和动静态混合的二进制代码分析技术. 本文调研了近年来的二进制代码安全分析领域上相关研究, 分别详细阐述了这 3 类技术中的主要方法, 并对其关键技术进行详细介绍.

关键词: 程序安全; 二进制分析; 静态分析; 动态分析; 动静态混合分析

引用格式: 周忠君,董荣朝,蒋金虎,张为华.二进制代码安全分析综述.计算机系统应用,2023,32(1):1–11. http://www.c-s-a.org.cn/1003-3254/8848.html

Survey on Binary Code Security Techniques

ZHOU Zhong-Jun[1], DONG Rong-Chao[1], JIANG Jin-Hu[2], ZHANG Wei-Hua[2]

[1](Software School, Fudan University, Shanghai 200433, China)
[2](Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 200433, China)

Abstract: In recent decades, computer hardware performance and software scale technology have greatly changed, and they have been involved in all aspects of human social life and production. The rapid development of computer technology has also brought about the concern of program security issues. Since there is a large amount of legacy software on the market, which is unmaintained and lacks source code support, people are worried about its security. As a result, binary analysis techniques are used to address security issues of this kind of software. Furthermore, the techniques can be classified as follows according to their detection ways: static binary code analysis techniques, dynamic binary code analysis techniques, and dynamic and static binary code analysis techniques. This study reviews the recent research on binary code security analysis, describes the main approaches in the above three techniques, and introduces the key techniques in detail.

Key words: program security; binary analysis; static analysis; dynamic analysis; hybrid dynamic and static analysis

1　引言

随着信息和自动化时代的到来, 手机、电脑、平板等各类电子产品已经完全融入了人们现代社会的日常生活中. 软件作为这些电子产品的重要体现, 其安全问题也越来越受到广泛关注. 由于高效的生产工具以及从事软件开发的人员的增多, 现如今的软件规模和数量都不同以往. 据统计, 以 Google Play 上的应用为例, 2009 年 12 月其平台上只有 16 000 个应用, 而 2021 年 9 月其数量到达 270 万[1]. 由于软件设计不严谨、软件开发人员的水平受限以及编程语言的固有不安全特

① 收稿时间: 2022-03-30; 修改时间: 2022-04-22; 采用时间: 2022-05-16; csa 在线出版时间: 2022-10-28
CNKI 网络首发时间: 2022-11-15

性等因素都可能导致生产的软件含有漏洞 (例如缓冲区溢出、整型溢出、格式化字符串攻击等[2–5]). 这类漏洞严重影响系统安全, 因而往往会被不法分子利用而对用户进行勒索敲诈, 危害社会秩序稳定. 因此, 软件程序的安全分析也是研究人员一直以来的热门课题.

一般来说, 软件安全分析可以从源代码级别和二进制级别两个层面进行分析. 虽然源代码级别的分析由于拥有程序的整体信息而分析难度较低, 精度较高. 但在实际生产中, 大部分的商业软件都并未开源, 而呈现给用户的仅是可执行的二进制代码的形式, 而且该类软件通常仅由一种语言编写, 跨语言分析难度较高, 因此针对源代码级别的分析具有局限性. 此外, 由于现实世界中还存在大量遗留软件, 该类型软件只提供二进制程序文件, 源代码丢失且无人维护, 这类软件的安全性也格外受到关注. 因此基于二进制级别的分析更具有通用性和实用性, 这也是研究人员主要的研究对象.

二进制级别的代码分析上存在着一定的挑战, 首先软件程序需要从源代码经过编译、优化之后生成二进制可执行程序, 二进制代码可读性降低, 且在程序的编译过程中往往会丢失掉上层的结构和类型等信息 (例如将变量和数据结构存到通用寄存器或者内存中), 从而导致分析任务更严峻, 难度更大; 其次在对二进制程序进行分析时, 一种常见的方法是通过分析程序的执行流或者调用流来避免执行可能有漏洞的区域, 这就需要获取程序的执行流或者调用流, 对于二进制程序来说, 由于代码存在直接跳转和间接跳转两种情况, 这使得构建程序的执行流变得困难, 并且在不同的架构下, 不同架构遵循的调用约定也不同, 这对于恢复程序的调用流也是一大挑战; 此外一些开发商在发行他们的程序时, 通常会对程序进行一些安全处理, 例如, 使用内存地址随机化技术, 甚至直接对二进制代码进行加密混淆, 这些都会对二进制代码分析带来一定的影响. 所以在本文中主要对二进制级别的安全分析的相关技术的难点和基本原理进行介绍, 调研了近年来的最新研究和进展, 并总结和比较了不同技术的特点.

二进制代码的安全分析按照是否要运行可以分为静态分析、动态分析以及动静态混合的分析技术. 静态分析技术一般首先将二进制程序反汇编成中间语言表示, 随后提取其特征信息来构建控制流图或数据流图再加以分析, 为了提高路径覆盖率, 更有符号执行和 Fuzz 等方法被结合使用; 动态分析技术则是通过插桩技术来实现指令级别的分析, 其可以在 (如 Call/Ret 指令、系统调用、库函数等) 插桩从而获得细粒度的程序运行时信息; 而动静态混合技术则是结合了静态分析和动态分析技术的特点, 一般先用静态分析技术来获取二进制程序的关键结构信息, 再结合动态分析技术对敏感路径进行分析, 其能有效地减少动态分析的所带来的性能开销.

静态分析的好处在于可以全盘对软件的二进制代码进行分析, 不会出现遗漏, 但缺点是缺少程序在执行时的一些信息, 例如对于指令间接跳转的地址无法确定、可能会分析到不会执行的代码等. 动态分析可以获得程序在执行过程中的上下文信息, 但是动态分析通常需要程序的输入, 需要尽可能多地考虑程序输入并确保程序输入的质量, 以便能够尽可能的覆盖所有的代码段, 其次, 对于 zero-day 漏洞或者恶意软件来说, 很难确定程序的输入, 这使得很难对二进制程序进行动态分析; 由于动态分析需要大量的输入, 动态分析也会带来的一定的开销; 此外, Polino 等人[6] 提出一种反插桩架构, 这使得攻击者可以通过检测二进制插桩工具的存在, 对动态分析程序进行削弱和屏蔽, 这使得使用动态分析变得更加困难. 动静结合技术首先使用静态分析缩小输入, 再执行二进制程序进行漏洞查找, 动静结合技术在可接受的开销里有着良好的漏洞查找性能.

近年来出现了一系列不同的方法[7–26] 解决二进制程序分析中所面临的问题, 其对现有的一些程序进行了详尽的分析, 包含对二进制程序进行漏洞检测、恶意行为识别与监控、污点分析、冗余代码消除等, 以此来增强二进制程序的安全性.

本文第 1 节介绍了二进制安全的相关背景介绍; 第 2 节对当下主要的二进制安全分析技术做了分类和总结, 其包括基于静态的二进制分析技术、基于动态的二进制分析技术以及动静态混合的二进制分析技术, 并对这 3 类技术做了评价比较; 第 3 节则阐述了对二进制安全分析技术的未来研究方向与挑战; 第 4 节则是本文的结束语.

2 二进制安全分析技术

二进制代码安全分析通常被用来检测程序的恶意行为, 其常见的应用场景包括漏洞检测、恶意软件分类、逆向工程等; 或用来生成路径覆盖全的测试用例, 如符号执行、Fuzz 等技术. 一般来说, 可以将二进制程

序的分析技术分为 3 类: 基于静态的二进制分析技术、基于动态的二进制分析技术和动静态混合的二进制分析技术. 本节针对以上 3 类技术, 分别调研了近年来的最新研究和进展, 并对其做了分类和总结.

2.1 整体概述

在二进制安全分析技术领域, 静态二进制安全分析技术操作方便并且快捷, 尤其是对于已知的程序漏洞, 能够迅速地在二进制代码中找出, 使用已打补丁的代码进行代码相似性分析大大降低了假阳性; 近来, 通过使用深度学习神经网络等算法更是增加了静态方法寻找程序漏洞的精确度, 使得使用静态分析方法进行二进制漏洞分析变得更加有吸引力.

对于动态二进制安全分析, 可以在执行时通过插桩分析关键数据流以及通过大量有限的输入分析程序的执行流, 能够更直接地找到程序漏洞, 动态安全分析可以对某个用户程序甚至是系统在运行时的指令、数据、控制流分析, 然而动态分析的缺点是插桩带来的性能影响, 由于插桩往往是在一条指令、一个关键数据之间发生, 因此维护整个程序尽可能小的受到影响是动态分析的关键.

动静结合的分析技术对于需要动态分析的程序来说是一个更好的选择, 在离线模式下静态分析控制流图以及关键信息流, 在动态模式下实时监测与跟踪离线分析下可能含有程序漏洞的部分, 以此来降低插桩范围, 降低性能开销. 在具有并行能力的动静结合安全分析技术下, 能够带来更准确的漏洞查找能力以及更低性能开销.

2.2 基于静态的二进制安全分析技术

静态的二进制安全分析指的是在目标程序运行之前, 利用现有技术对其二进制代码进行安全分析, 静态二进制安全分析主要被用来进行程序的漏洞检测分析. 程序漏洞检测, 一般可以分为已知漏洞的检测与未知漏洞的识别与检测, 即是否在目标程序中找出新的、尚未被发现的漏洞. 对于已知的漏洞, 静态二进制安全分析通常采用代码相似性分析技术来查找漏洞; 对于尚未被发现的漏洞, 通常使用一些数据流分析、符号验证等方法来寻找代码中可能出现的漏洞. 随着深度学习神经网络等算法的流行, 使用一些神经网络算法精确查找二进制代码的安全漏洞方法也开始使用.

代码相似性分析是将已知漏洞的二进制代码段与目标程序进行匹配, 如果目标程序中包含有与漏洞二进制代码段高度相似的部分, 则证明目标程序中包含有漏洞. 如图 1 所示, 使用已知漏洞的漏洞签名通过具体相似性分析算法与目标程序进行比较, 最后通过相应的决策算法判断目标程序中是否包含这一漏洞. Pewny 等人[7] 在 2015 年提出了使用代码相似性漏洞检测方法跨指令集架构检测不同指令集架构中的已知漏洞. 除了使用代码相似性分析方法, Feng 等人[8] 在 2014 年提出了使用二进制代码的语义相似性来检测已知的漏洞, 使用高级语言解释包含漏洞的二进制代码的语义, 并生成漏洞签名, 与目标程序匹配, 从而在程序上语义进行漏洞的相似性检测. 然而, 由于包含漏洞的代码往往只需要经过少量的修改就不再会被攻击, 例如提高权限、设置变量的边界条件等, 因此被打补丁的程序往往与包含漏洞的程序在代码相似性或者语义相似性上都很高, 这些代码仍然维持着他们原本的功能. 因此单纯使用漏洞程序进行代码相似性分析, 会出现一定的假阳性. Xu 等人[9]、Jang 等人[10] 分别提出了对已打补丁的程序与包含漏洞的程序分别生成相应的程序签名, 然后对目标程序进行相似性分析, 以此来降低假阳性, 提高漏洞检测的精确度.

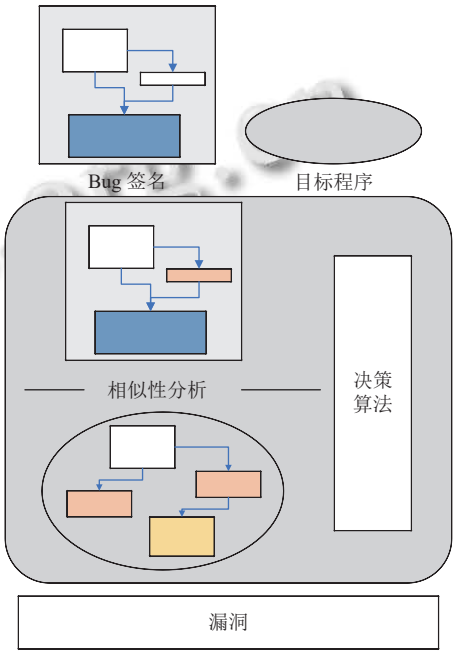

图 1 代码相似性分析

对于尚未被发现的漏洞, 静态二进制安全分析技术通常使用程序数据流或者控制流来分析和检查敏感信息可能的走向以及程序执行的走向、权限的更改等.

IntScope[11] 使用符号执行技术对程序中的数据流进行分析来检测存在整型溢出这一漏洞的代码段. IntScope 将可执行文件的二进制码转化为中间表达语言 (IR), 对包含敏感数据流的 IR 部分进行符号验证, 判断这些数据流是否存在整型溢出. PiOS[27] 对 iOS 系统的手机软件进行漏洞检测, 对可执行文件构建控制流图 (CFG), 标记代码中的敏感数据以及可以泄露数据的代码段, 例如输出函数等, 然后检查控制流图中用户敏感信息可出现的一些关键数据流, 对这些数据流进行可行性分析, 找出可能执行到泄露区域的数据流, 从而查找到了新的漏洞. 在嵌入式系统的固件中, 存在一类绕过权限验证的固件漏洞, 称为后门 (backdoor). Firmalice[28] 使用符号验证的方法, 通过对固件软件中可能的申请权限的敏感控制流进行分析, 得出能够到达获得高级权限或者绕过权限的控制流路径, 不同之处在于, firmalice 使用输入决策树分析得出这些路径在实际使用中是否能够被执行, 来判断这条控制流路径是不是一个真正的漏洞. 除了对用户程序以及嵌入式程序的漏洞检测外, 日常使用的 Linux 操作系统、Windows 操作系统以及 Apple kernel 也存在着一定的漏洞, DR CHCEKER[29]、digtool[30] 以及 iDEA[31] 分别使用一些静态二进制安全分析方法的组合完成了对上述操作系统的漏洞检测, 主要是针对权限泄露、信息泄露等漏洞.

近年来, 随着深度学习的兴起, 使用一些神经网络、机器学习算法来进行漏洞检测的方法也逐渐被提出. VESTIGE[32] 的关键思想还是在于使用代码相似性进行漏洞检测. VESTIGE 通过对二进制代码构建全新的带标签的函数调用图, 利用图神经网络训练生成更精确的嵌套 (embeddings), 将包含已知漏洞的嵌套与目标程序进行相似性比较, 从而能够得到更精确的代码相似性检测结果. Ouyang 等人[33] 使用自然语言处理 (NLP) 中的深度学习神经网络来进行漏洞检测, 其核心也是代码相似性分析, 使用 Word2Vec 工具获得已知的漏洞代码的特征向量, 使用长短时记忆网络 (LSTM) 对特征向量进行训练, 生成深度学习神经网络模型, 再对目标程序进行漏洞检测, 以获得更高的漏洞检测精确度.

2.3 基于动态的二进制安全分析技术

静态分析技术无法获得程序运行时的动态信息 (如指令间接跳转地址) 和分析细粒度的系统级别行为, 从而导致其分析灵活度和精确性较差.

动态分析技术通常利用二进制插桩技术来监控程序运行时行为以获得更准确的信息, 其主要是通过向二进制程序中插入指令级别的额外代码来检测或改变程序行为, 从而进行分析. 一般来说, 可以根据是否要对内核代码进行插桩把动态二进制插桩技术分为应用级的动态二进制插桩技术和系统级的动态二进制插桩技术. 应用级动态二进制插桩技术其只能对运行在用户空间的指令进行插桩分析, 而无法进入系统调用等处于系统态的代码空间. 经典的应用级动态二进制插桩工具有 PIN[34]、DynamoRIO[35]、Valgrind[36]、DynInst[37] 和 Libdetox[38] 等. 为了能对如系统调用等内核级别的代码进行检测, 近年来, 有许多基于系统级动态二进制插桩分析工具被提出来. 由于系统级动态二进制插桩工具一般首先要仿真整个虚拟机环境, 因此这些工具大多都基于完整的系统级动态二进制翻译器上进行实现. 最常见的是基于 QEMU[39] 上实现的系统级动态二进制插桩工具有很多包括 QTrace[40], Panda[41] 和 PEMU[13] 等.

基于上述二进制插桩技术, 许多动态二进制安全分析技术被提出. 本节接下来对各个动态二进制安全分析技术进行阐述.

动态污点分析 (dynamic taint analysis) 是一种在程序运行时标记感兴趣数据源及记录其传播路径的一种动态程序分析技术, 其被多用于在程序安全分析检测、信息流控制等. 动态污点分析技术其流程一般如图 2 所示, 其由 3 个部分组成: 污点源, 污点汇聚点, 无害处理. 污点源则为用户所感兴趣的敏感数据, 一般通过插桩技术来跟踪污点源的传播路径; 污点汇聚点则为产生安全或者泄露隐私的边界; 无害处理则为通过一定加密等手段使得数据变得安全的操作. 动态污点分析技术通过追踪运行时程序的敏感数据传播路径来确保程序的安全性.

针对动态污点分析相关技术, Dytan[14] 包含一个通用的高度灵活且可定制的污染分析框架, 其可以根据用户提供的配置文件根据污染源、传播策略和污染接收器指定要执行的动态污点分析的类型. 为了对二进制程序指令和数据流进行分析, 其利用 Pin 工具提供的丰富的 API 来进行插桩, 其可以同时对数据流和控制流所造成的污染进行跟踪, 提供了一个通用的分析模型. Minemu[15] 是一个基于 x86 架构污染分析跟踪技

术, 其针对动态污点分析性能较差问题, 设计了一种新的内存布局, 以减少在对内存操作进行污染传播分析时的开销; 污点分析的另一大开销是大量的寄存器分配操作, 为了缓解 x86 架构稀缺寄存器数量, 其使用 SSE 寄存器增加可使用寄存器范围, 缓解分配寄存器时可能导致的寄存器溢出的压力. TaintDroid[16] 是应用于智能手机上的一个能高效的、全系统实时追踪敏感信息泄露的技术, 其可以同时跟踪多种敏感污染源并集成了污染传播过程中的多种跟踪粒度来提高执行效率.

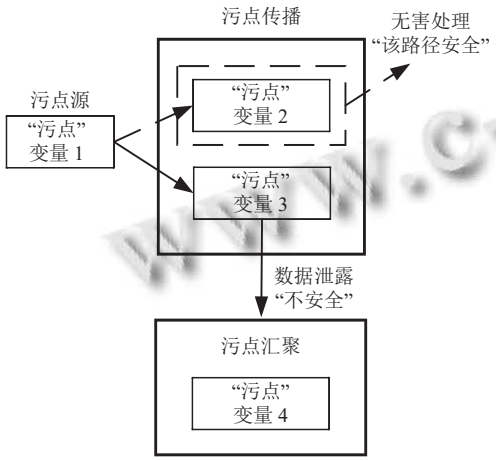

图 2 污点分析技术

在恶意软件检测上, X-Force[17] 是一种新型的二进制分析引擎, 其使用动态二进制插桩技术来探索二进制程序可走的所有路径, 也就是说不需要任何输入就能执行二进制程序. 在执行阶段, X-Force 还构建了一个无奔溃模型, 其可以通过检测/恢复异常, 设置已分配内存位置来避免无效内存访问引起的程序奔溃, 通过这种方式, X-Force 能有效挖掘程序中隐藏的恶意软件行为. Bernardi 等人[18] 提出了一种基于过程挖掘的动态恶意软件检测方法. 其主要思路为构建应用程序中的系统调用执行序列来作为软件动态行为特征指纹, 并通过训练分类器对恶意软件分类. Arora 等人[19] 提出了一种新的基于 Android 系统的智能手机的恶意软件检测方法, 其主要思路为检测应用程序中恶意活动的网络流量特征, 并训练基于规则的分类器来对该网络流量特征进行分类从识别恶意软件行为.

不透明谓词被广泛应用于软件保护和恶意软件中, 用于混淆程序控制流. LOOP[20] 是一种被用来有效检测和分析二进制程序中的不透明谓词的一种技术, 其首先借助于动态二进制插桩技术获取二进制程序运行时指令流信息, 随后分析其中的贡献谓词的指令信息并使用符号执行来构造一般的逻辑公式, 它表示不透明谓词的内在特征, 然后用约束求解器求解这些公式, 以此来识别不透明谓词信息. 在密码学安全领域中, K-Hunt[21] 是一种使用动态分析来识别二进制可执行文件中不安全密钥的技术. 其首先通过加密密钥操作的属性 (如大量使用算术指令, 产生具有高随机性的数据流, 以及执行长度与输入大小成比例) 来识别出加密密钥操作代码块, 随后根据加密操作代码块中会访问的数据及加密秘钥特征识别出加密密钥在内存中存储的位置, 最后通过跟踪该部分数据的来源及传播过程来评估其安全性.

在对内核代码及系统级行为的检测下, QVMII[42] 是一个强大的系统级动态二进制插桩工具, 其可以对系统调用操作、文件操作、进程操作等行为进行监测. 对于系统调用, 其将监测代码作为回调函数插入翻译后的代码中, 以便于在运行中碰到系统调用指令能通过该回调函数对系统调用进行统计分析, 而该回调函数还包括保存和恢复系统调用的参数和返回值功能. 对于文件操作, 其提供了独立于客户操作系统和硬件的文件系统监视插件, 该文件系统监视插件识别所有打开的文件的列表并监控和收集操作系统对文件的操作. 而对于进程操作, 则其提供当前执行的所有客户进程的列表, 且存储该进程包含的信息如当前执行上下文、父进程执行上下文、进程号、可执行映像的名称, 并将该类信息提供给用户使用. Qelt[43] 提供了跨指令集架构的且支持系统级动态二进制插桩的分析方法, 其不仅可以提供仿真虚拟化环境平台, 还可以对内核代码和库函数等系统级行为进行细粒度监控分析. 通常来说系统级动态二进制插桩工具都有着较差的效率, 因此为了加速系统级动态二进制插桩工具的性能, Qelt 还提出了一种利用宿主的浮点计算单元来优化跨架构的浮点模拟操作, 还优化了对间接跳转指令的处理和使用动态伸缩的软件 (translation look-aside buffers, TLB) 技术, 并降低了并行仿真中的翻译过程的开销.

2.4 动静态混合的二进制安全分析技术

静态分析技术无法获得程序运行时信息导致对间接跳转指令等行为分析精确度较差, 而动态分析技术则由于受到输入所影响无法覆盖全路径的代码且通常执行耗时且效率较差. 动静态混合的二进制安全分析

技术则结合了静态和动态分析技术这两者的优点, 规避了两者的不足之处被使用.

动态数据流跟踪技术由于其需要进行细粒度的指令基本行为分析, 其会导致原有程序性能会降低一个数量级. 而针对此问题, Jee 等人[44] 在 2012年提出了一种结合静态和动态分析的方式来提高动态数据流跟踪性能的一种技术, 其架构如图 3 所示, 其首先使用一个静态分析器和动态分析器来对目标二进制程序进行分析以提取出基本块和控制流信息, 基本块则为包含一系列指令序列且该指令序列仅有一个入口点, 而控制流信息则是由基本块之间的跳转关系所连接的信息; 随后将获得的基本块和控制流信息传输到分析器中, 分析器从中获取数据依赖关系, 并使用一系列特定的优化策略如死代码消除和复制传播来减少冗余跟踪操作; 最后分析器再将优化后的分析代码通过特定的动态二进制插桩软件如 PIN 将其插入到原有程序中, 以进行数据跟踪操作. 而在实际运行过程中, 由于静态分析器和动态分析器的不完整性, 有可能部分的基本块无法被分析器所优化, 该部分基本块则用原有的未优化的跟踪代码进行检查, 而同时分析器在运行时也会时刻收集这部分未优化的基本块将其添加到分析列表中, 从而生成一个正向反馈循环过程.

ShadowReplica[45] 是一种高效的具有并行能力的动静态混合的二进制安全分析技术, 其通过将分析过程和执行过程解耦, 和使用空闲的 CPU 核并行运行被检测的应用程序和监测工具代码来加速性能. 在分析过程其主要使用一种离线的方式收集二进制程序信息, 其通过一个静态分析器和动态分析器来获得如内存信息、控制流信息以及操作系统事件等, 并使用优化策略[44] 来生成将插入应用程序的优化代码. 而在执行阶段, ShadowReplica 会为每个应用线程生成一个影子线程, 影子线程运行与该应用线程相关的分析代码并且运行在一个空闲的 CPU 核上. ShadowReplica 只需要在应用线程指定需要监测的位置插入一个存根, 通过该存根与影子线程进行通信来统计分析, 其通过这种并行的方式来加速动态运行过程中的分析过程. 为了最小化的减少应用线程和影子线程之间的通信开销, ShadowReplica 还设计并优化了一个共享环缓冲数据结构, 以便在多核 CPU 上的线程之间有效地共享数据, 提高执行效率.

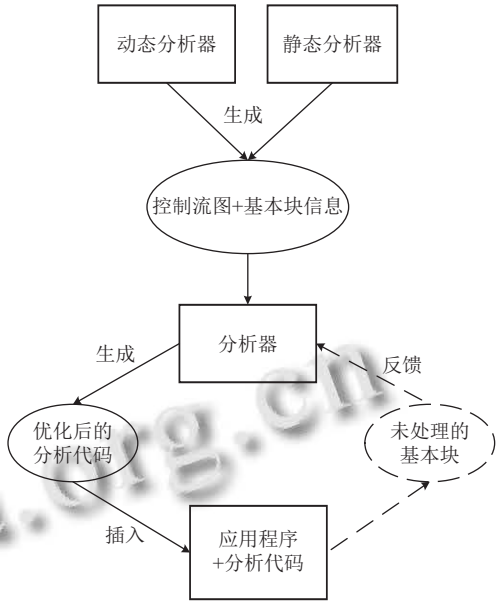

图 3 动静态混合分析方法

StraightTaint[46] 是一种在 x86 架构上的基于轻量级日志记录的动静态混合的二进制程序污点分析技术. 其流程主要包含两个阶段: 在线日记记录阶段和离线分析阶段. 在线日记记录阶段是非常轻量级的, 其在动态二进制插桩工具上实现一个日志记录工具, 主要记录二进制程序运行时的基本块信息和控制流信息. 而在线日记记录阶段收集到的日志数据将被传到离线分析工具中使用, 离线分析工具将该日志数据重建为直接代码轨迹, 并将原有代码中复杂的 x86 架构指令信息给转换为较为简单的中间代码表示 BIL[47], 再通过符号执行技术来对污染数据源进行因果数据流分析. StraightTaint 另一特性可以进行条件污点分析, 具体来说就是当碰到跳转指令时, 其会收集 x86 架构上的条件码信息 (存储在 EFLAGS 寄存器中) 来组成污染条件约束, 以便获得在指定条件约束集下的污染跟踪信息. StraightTaint 这种通过解耦程序执行和污染分析过程, 其使得在不与应用程序频繁进行数据通信的情况下, 实现离线符号污染分析.

与上面 3 种技术不同, BitBlaze[12] 则是一种支持全系统且覆盖路径全面的动静态混合的二进制安全分析工具. 其由 3 个组件组成: TEMU, VINE 和 Rudde. TEMU 是基于 QEMU 实现的一个系统级动态二进制插桩工具, 其可以记录包含内核代码在内的所有程序的指令、内存等信息及可以进行动态污点分析, 而 TEMU 作为 BitBlaze 的动态分析模块进行对二进制程

序检测分析. VINE 则是 BitBlaze 的静态分析模块, 其包含一个前端和后端模块, 前端模块将复杂的 x86 指令反汇编 VINE-IR (中间代码表示) 以简化指令集, 而后端模块则通过对生成 VINE-IR 进行分析, 以生成控制流图、数据流图、符号执行、路径约束等信息. Rudder 则将由 TEMU 和 VINE 所记录的路径约束信息进行取反来不断生成新的输入样例, 以覆盖更全面的路径来测试程序的安全性. 因此 BitBlaze 则通过这 3 个组件的合作的方式, 其结合了动静态的优点, 提供了一个覆盖路径全面的支持全系统的二进制安全分析工具.

2.5 评价

基于静态的二进制代码安全分析、基于动态的二进制代码安全分析以及动静态混合的二进制代码安全分析 3 类技术都各有优缺点. 对于二进制代码安全分析算法中, 比较注重的评价指标包括分析速度、检测结果的准确度以及自动化程度. 因此本小节首先对这 3 类方法从性能、准确性以及自动化程度进行一个整体评价.

从性能角度来说, 基于静态分析的技术直接对二进制代码全局信息进行分析, 其本质上是对二进制代码文本进行算法分析和处理, 其分析速度与分析算法的设计和二进制代码大小与复杂程度直接相关, 而不被该程序实际会运行的时间所影响, 因此其通常都有着较快的分析速度; 而基于动态分析的技术一般需要利用动态二进制插桩技术获取程序运行时的动态信息, 其需要执行二进制可执行程序并以指令级别或基本块级别的粒度进行分析, 由于要收集程序运行时信息, 其通常要向原有程序中插入插桩指令, 这种方式膨胀了原有程序, 导致其性能较差, 通常会降低数倍到数十倍不止; 动静态混合的分析方法则通过静态分析来进行预处理以缩小动态分析时所需要的关键路径或者通过动态分析来获得基本块、控制流等信息后再通过离线的静态分析来加快分析速度并提高准确率.

从准确度来说, 基于静态分析的技术虽然能利用二进制代码的全局信息, 但其不能获取运行时动态信息 (如间接跳转指令目标地址), 无法获得完整的程序控制流信息, 分析的准确性较低, 且对于不规范的二进制代码其可能分析到不被执行的代码区域, 导致其分析结果存在一定的假阳性; 基于动态分析的技术, 其可以获取程序运行时完整信息, 甚至可以修改二进制程序行为, 其通常有着较高的分析准确性, 但基于动态分析的技术其分析结果与程序输入直接相关, 无法做到全路径测试, 而覆盖更全面的路径则会增加测试次数, 从而加大分析时间; 动静态混合的分析方法则结合动静态共同分析来提升准确度, 但一般其程序工作量较大且目前该方面相关研究方法较少.

从自动化程度来说, 基于静态分析的方法对于同一指令集架构的二进制程序一般可以直接使用, 仅需要对一些基本参赛进行设置和调整, 自动化程度较高, 用户所需的干预较少, 但对于不同指令集架构的二进制程序则需要重写静态分析算法; 由于基于动态分析的方法使用到动态二进制插桩技术, 而插桩代码通常需要用户根据需求来编写, 因此基于动态分析的方法则需要较多的用户干预. 在对于跨指令集架构的二进制程序分析上, 如 Qelt[43] 其利用动态二进制翻译技术来有效分析跨指令集架构的二进制程序; 动静态混合的方法需要同时对静态分析算法和动态分析算法进行设置, 其还需要静态分析和动态分析信息进行交互, 具体来说根据不同的算法有着不同的用户干预程度.

为了能更进一步比较不同技术之间的特点, 本文选取了第 2.2–2.4 节中介绍过的 3 类技术的几篇关键论文, 对其进行评价比对. 具体如表 1 所示, 该表展示在分析粒度、性能、准确性以及应用领域这几个评价指标下的比对结果.

从这些比较可以看出, 基于静态、动态以及动静混合的技术各有优劣, 可以根据具体场景以及对性能及准确性和用户干预程度的需求来选择不同的方法进行检测分析.

3 未来研究方向与挑战

二进制代码安全分析技术的发展还有着较大的空间, 本节展望了二进制代码分析技术在未来所可能注重的研究方向, 包括分析算法的高效性; 基于人工智能算法的智能化分析方法; 以及如何有效从二进制程序中精确类型恢复和内核代码的安全分析技术这 4 个方面去阐述未来研究方向与挑战.

1) 高效性

二进制代码安全分析的算法需要同时满足性能及精度的支持, 和较全面的代码覆盖率. 从算法层面上来说, 一个有效的研究方向为利用二进制代码底层的抽

象信息, 或早先工作的先验知识, 以及一些启发式空间搜查算法来提高分析速度; 对于输入的二进制代码也可使用一些代码去膨胀技术[48,49] 来减少无效分析及增强二进制代码安全性. 从硬件及程序编写层面上来说, 设计并行化的分析程序并利用硬件特性来实现高规模、高并发的分析算法.

表 1 不同方法评价

方法类型	方法	分析粒度	性能	准确性	应用领域
静态方法	Pewny等人[7]	基本块粒度	高	存在假阳性	跨架构漏洞检测
	Apposcopy[8]	基本块粒度	高	存在假阳性	安卓软件漏洞检测
	IntScope[11]	执行轨迹	中	准确	x86架构漏洞检测
	PiOS[27]	执行轨迹	中	准确	iOS软件漏洞检测
	VESTIGE[32]	指令级粒度	慢	存在假阳性	x86漏洞检测
动态方法	Dytan[14]	指令级粒度	慢	较准确	污点分析
	Bernardi等人[18]	函数粒度	较慢	存在假阳性	恶意行为监测
	LOOP[20]	指令级粒度	中	存在假阳性	不透明谓词分析
	K-Hunt[21]	指令级粒度	中	较准确	不安全密钥识别
	QVMII[42]	基本块粒度	慢	准确	全系统插桩检测
动静混合方法	Jee等人[44]	基本块粒度	中	较准确	污点分析
	ShadowReplica[45]	基本块粒度	较高	较准确	污点分析
	StraightTaint[46]	基本块粒度	较高	较准确	污点分析
	BitBlaze[12]	基本块粒度	慢	准确	全系统插桩检测

2) 智能化

近年来, 人工智能算法在各个领域上被发挥出重要作用. 在二进制安全分析领域, 基于机器学习或深度学习的算法对于如高结构化输入生成、代码相似性检测、恶意软件分类、脆弱路径筛选等都有着重要贡献. 获取较大数据集需要很高的人力成本, 利用半监督或者无监督的训练方法可以较好解决该问题[23]. 但目前来说, 基于人工智能算法的二进制安全分析的相关研究还较少, 更多自动化且适应于不同应用场景的人工智能算法是一个值得研究的方向.

3) 类型恢复

二进制代码的类型恢复问题是二进制代码安全分析领域如漏洞检测[50,51]、二进制重写[52]、逆向工程[53]等中重要的一环. 由于源代码在经过编译后会丢失大量与类型相关的信息, 导致二进制代码的数据结构的类型恢复是个严峻的挑战. 二进制代码类型恢复算法通常难以恢复各种类型的数据结构, 且分析结果过于保守, 耗时较长, 因此其对二进制代码分析的帮助有限. 研究提高二进制代码类型恢复的精确度及兼顾性能也是一个值得关注的问题.

4) 内核代码安全分析

内核代码中的软件漏洞会导致严重的安全问题. 而随着虚拟化技术、容器技术及多核的兴起, 内核代码的安全问题更是值得关注. 内核代码复杂而庞大, 其中驱动程序代码是主要产生漏洞的代码部分, 而驱动程序中的代码包含较重的指针调用及与硬件直接相关, 如何有效分析该内核驱动模块代码也是一个挑战. 在内核代码安全分析方法中, 使用动态二进制翻译技术来仿真相同指令架构或跨指令集架构的整个虚拟机并监控包含用户态及系统态的所有指令及数据是一个有效手段[43], 其能对包括内核在内的整个虚拟机代码进行安全分析, 但这种方式通常有着较差的性能且难以覆盖内核代码全路径, 如何对其性能优化及提高代码测试路径也是值得研究的方向.

其余的如漏洞自动补丁技术和二进制重写技术等方向也都是值得探究的研究方向.

4 结束语

近年来, 计算机程序安全问题已越来越受到研究学者的广泛关注. 而由于缺乏源代码的支持, 二进制代码的安全性问题更是当今计算机安全领域中最关键的研究热点之一. 本文调研了近十多年来的二进制代码安全分析领域相关研究, 并根据其检测方式对其分类: 基于静态的二进制代码安全分析技术, 基于动态的二进制代码安全分析技术和动静态混合的二进制代码安全分析技术. 基于静态的二进制代码分析技术不需要

执行二进制程序, 而直接去挖掘二进制代码中的信息, 从中发现程序漏洞. 基于动态的二进制代码安全分析技术则是边执行二进制程序边分析结果, 其可以获得静态分析所不能获得的程序运行时信息 (如间接跳转指令目标等). 基于动态的二进制分析方法一般使用动态二进制插桩技术来实现该目标, 许多安全分析技术利用插桩技术来实现对二进制程序进行分析. 基于动静态混合的二进制安全分析技术, 其结合了动静态分析技术两者的优势, 同时满足高效和准确性的特点. 在此基础上本文还探究了二进制安全分析技术的未来研究方向与挑战, 本文给出了二进制代码安全分析技术的发展趋势将重点关注其高效性、智能化、类型恢复问题及内核代码安全分析等研究方向.

参考文献

1 Mobile Internet & Apps. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. [2022-03-21].

2 Kirsch J, Zhechev Z, Bierbaumer B, *et al*. PwIN—Pwning Intel piN: Why DBI is unsuitable for security applications. Proceedings of the 23rd European Symposium on Research in Computer Security. Barcelona: Springer, 2018. 363–382.

3 Perkins JH, Kim S, Larsen S, *et al*. Automatically patching errors in deployed software. Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. Big Sky: ACM, 2009. 87–102.

4 Xie JY, Fu X, Du XJ, *et al*. Autopatchdroid: A framework for patching inter-APP vulnerabilities in Android application. Proceedings of 2017 IEEE International Conference on Communications (ICC). Paris: IEEE, 2017. 1–6.

5 Dolan-Gavitt B, Hulin P, Kirda E, *et al*. Lava: Large-scale automated vulnerability addition. Proceedings of 2016 IEEE Symposium on Security and Privacy (SP). San Jose: IEEE, 2016. 110–121.

6 Polino M, Continella A, Mariani S, *et al*. Measuring and defeating anti-instrumentation-equipped malware. Proceedings of the 14th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Bonn: Springer, 2017. 73–96.

7 Pewny J, Garmany B, Gawlik R, *et al*. Cross-architecture bug search in binary executables. Proceedings of 2015 IEEE Symposium on Security and Privacy. San Jose: IEEE, 2015. 709–724.

8 Feng Y, Anand S, Dillig I, *et al*. Apposcopy: Semantics-based detection of Android malware through static analysis. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong: ACM, 2014. 576–587.

9 Xu YF, Xu ZZ, Chen BH, *et al*. Patch based vulnerability matching for binary programs. Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. Online: ACM, 2020. 376–387.

10 Jang HJ, Yang K, Lee G, *et al*. QuickBCC: Quick and scalable binary vulnerable code clone detection. Proceedings of the 36th IFIP International Conference on ICT Systems Security and Privacy Protection. Oslo: Springer, 2021. 66–82.

11 Wang TL, Wei T, Lin ZQ, *et al*. IntScope: Automatically detecting integer overflow vulnerability in x86 binary using symbolic execution. Proceedings of the Network and Distributed System Security Symposium. San Diego: NDSS, 2009. 1–14.

12 Song D, Brumley D, Yin H, *et al*. BitBlaze: A new approach to computer security via binary analysis. Proceedings of the 4th International Conference on Information Systems Security. Hyderabad: Springer, 2008. 1–25.

13 Zeng JY, Fu YC, Lin ZQ. PEMU: A pin highly compatible out-of-VM dynamic binary instrumentation framework. Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Istanbul: ACM, 2015. 147–160.

14 Clause J, Li WC, Orso A. Dytan: A generic dynamic taint analysis framework. Proceedings of the 2007 International Symposium on Software Testing and Analysis. London: ACM, 2007. 196–206.

15 Bosman E, Slowinska A, Bos H. Minemu: The world ’s fastest taint tracker. Proceedings of the 14th International Workshop on Recent Advances in Intrusion Detection. Menlo Park: Springer, 2011. 1–20.

16 Enck W, Gilbert P, Han S, *et al*. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems, 2014, 32(2): 5.

17 Peng F, Deng Z, Zhang XY, *et al*. X-force: Force-executing binary programs for security applications. Proceedings of the 23rd USENIX Security Symposium. San Diego: USENIX, 2014. 829–844.

18 Bernardi ML, Cimitile M, Distante D, *et al*. Dynamic malware detection and phylogeny analysis using process mining. International Journal of Information Security, 2019,

18(3): 257–284. [doi: 10.1007/s10207-018-0415-3]

19 Arora A, Garg S, Peddoju SK. Malware detection using network traffic analysis in Android based mobile devices. Proceedings of the 2014 8th International Conference on Next Generation Mobile Apps, Services and Technologies. Oxford: IEEE, 2014. 66–71.

20 Ming J, Xu DP, Wang L, *et al*. LOOP: Logic-oriented opaque predicate detection in obfuscated binary code. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. Denver: ACM, 2015. 757–768.

21 Li JR, Lin ZQ, Caballero J, *et al*. K-Hunt: Pinpointing insecure cryptographic keys from execution traces. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto: ACM, 2018. 412–425.

22 Lin Y, Gao DB. When function signature recovery meets compiler optimization. Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2021. 36–52.

23 Downing E, Mirsky Y, Park K, *et al*. DeepReflect: Discovering malicious functionality through binary reconstruction. Proceedings of the 30th USENIX Security Symposium. Online: USENIX, 2021. 3469–3486.

24 Jang D. Badaslr: Exceptional cases of ASLR aiding exploitation. Computers & Security, 2022, 112: 102510.

25 Dinesh S, Burow N, Xu DY, *et al*. RetroWrite: Statically instrumenting COTS binaries for fuzzing and sanitization. Proceedings of 2020 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2020. 1497–1511.

26 Wartell R, Mohan V, Hamlen KW, *et al*. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. Proceedings of the 2012 ACM Conference on Computer and Communications Security. Raleigh: ACM, 2012. 157–168.

27 Egele M, Kruegel C, Kirda E, *et al*. PiOS: Detecting privacy leaks in iOS applications. Proceedings of the Network and Distributed System Security Symposium. San Diego: NDSS, 2011. 177–183.

28 Shoshitaishvili Y, Wang RY, Hauser C, *et al*. Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware. Proceedings of the 22nd Annual Network and Distributed System Security Symposium. San Diego: NDSS, 2015.

29 Machiry A, Spensky C, Corina J, *et al*. DR.Checker: A soundy analysis for Linux kernel drivers. Proceedings of the 26th USENIX Security Symposium. Vancouver: USENIX, 2017. 1007–1024.

30 Pan JF, Yan GL, Fan XC. Digtool: A virtualization-based framework for detecting kernel vulnerabilities. Proceedings of the 26th USENIX Security Symposium. Vancouver: USENIX, 2017. 149–165.

31 Bai XL, Xing LY, Zheng M, *et al*. iDEA: Static analysis on the security of Apple kernel drivers. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. Online: ACM, 2020. 1185–1202.

32 Ji YD, Cui L, Huang HH. VESTIGE: Identifying binary code provenance for vulnerability detection. Proceedings of the 19th International Conference on Applied Cryptography and Network Security. Kamakura: Springer, 2021. 287–310.

33 Ouyang WL, Li M, Liu QQ, *et al*. Binary vulnerability mining based on long short-term memory network. Proceedings of 2021 World Automation Congress (WAC). Taipei: IEEE, 2021. 71–76.

34 Luk CK, Cohn R, Muth R, *et al*. Pin: Building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005, 40(6): 190–200. [doi: 10.1145/1064978.1065034]

35 Bruening D, Zhao Q, Amarasinghe S. Transparent dynamic instrumentation. Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments. London: ACM, 2012. 133–144.

36 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]

37 Buck B, Hollingsworth JK. An API for runtime code patching. The International Journal of High Performance Computing Applications, 2000, 14(4): 317–329. [doi: 10.1177/109434200001400404]

38 Payer M, Gross TR. Fine-grained user-space security through virtualization. ACM SIGPLAN Notices, 2011, 46(7): 157–168. [doi: 10.1145/2007477.1952703]

39 Bellard F. QEMU, a fast and portable dynamic translator. Proceedings of the Annual Conference on USENIX Annual Technical Conference. Anaheim: USENIX, 2005. 41–46.

40 Tong X, Moshovos A. QTrace: A framework for customizable full system instrumentation. Proceedings of 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Philadelphia: IEEE, 2015. 245–255.

41 Dolan-Gavitt B, Hodosh J, Hulin P, *et al*. Repeatable reverse

engineering with PANDA. Proceedings of the 5th Program Protection and Reverse Engineering Workshop. Los Angeles: ACM, 2015. 4.

42 Dovgalyuk P, Fursova N, Vasiliev I, *et al*. QEMU-based framework for non-intrusive virtual machine instrumentation and introspection. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 944–948.

43 Cota EG, Carloni LP. Cross-ISA machine instrumentation using fast and scalable dynamic binary translation. Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. Providence: ACM, 2019. 74–87.

44 Jee K, Portokalidis G, Kemerlis VP, *et al*. A general approach for efficiently accelerating software-based dynamic data flow tracking on commodity hardware. Proceedings of the 19th Annual Network and Distributed System Security Symposium. San Diego: NDSS, 2012.

45 Jee K, Kemerlis VP, Keromytis AD, *et al*. ShadowReplica: Efficient parallelization of dynamic data flow tracking. Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. Berlin: ACM, 2013. 235–246.

46 Ming J, Wu DH, Wang J, *et al*. StraightTaint: Decoupled offline symbolic taint analysis. Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. Singapore: IEEE, 2016. 308–319.

47 Brumley D, Jager I, Avgerinos T, *et al*. BAP: A binary analysis platform. Proceedings of the 23rd International Conference on Computer Aided Verification. Snowbird: Springer, 2011. 463–469.

48 Ghaffarinia M, Hamlen KW. Binary control-flow trimming. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. London: ACM, 2019. 1009–1022.

49 Quach A, Prakash A, Yan L. Debloating software through piece-wise compilation and loading. Proceedings of the 27th USENIX Security Symposium. Baltimore: USENIX, 2018. 869–886.

50 Lin ZQ, Zhang XY, Xu DY. Automatic reverse engineering of data structures from binary execution. Proceedings of the 11th Annual Information Security Symposium. West Lafayette: ACM, 2010. 5.

51 Wang S, Liu TY, Tan L. Automatically learning semantic features for defect prediction. Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). Austin: IEEE, 2016. 297–308.

52 Fu YC, Lin ZQ. Bridging the semantic gap in virtual machine introspection via online kernel data redirection. ACM Transactions on Information and System Security, 2013, 16(2): 7.

53 Jacobson ER, Rosenblum N, Miller BP. Labeling library functions in stripped binaries. Proceedings of the 10th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools. Szeged: ACM, 2011. 1–8.

(校对责编: 孙君艳)