智能模糊测试综述:问题探索和方法分类

王琴应[1)] 许嘉诚[2)] 李宇薇[3)] 潘祖烈[3)] 张玉清[4)] 张 超[5)] 纪守领[1)]

[1)](浙江大学计算机科学与技术学院 杭州 310007)

[2)](浙江大学控制科学与工程学院 杭州 310007)

[3)](国防科技大学电子对抗学院 合肥 230037)

[4)](中国科学院大学国家计算机网络入侵防范中心 北京 101408)

[5)](清华大学网络研究院 北京 100084)

摘 要 随着近年来软件系统规模以及复杂性的增加,安全漏洞数量持续增长、影响面逐步扩大,全球安全形势依然严峻.针对该问题,学术界和工业界致力于研究高效的漏洞挖掘技术,提前发现和修复潜在的漏洞.其中模糊测试作为先进的漏洞挖掘技术之一,吸引了学术界和工业界的广泛关注.为了进一步提高漏洞挖掘的能力,研究人员提出了智能模糊测试,即利用人工智能和程序分析等技术作为辅助,从而实现对复杂软件系统更高效的测试和分析并智能引导漏洞挖掘方向.本文回顾了近八年来智能模糊测试研究进展,提出了一个通用模糊测试流程模型和问题导向的智能模糊技术分类方法,从优化测试输入生成、提高测试效率以及增强测试预言机三个方面总结了当前智能模糊测试的优势和不足之处,最后对智能模糊测试面临的挑战和未来研究方向进行展望和总结.

关键词 模糊测试;软件与系统安全;漏洞挖掘;人工智能;程序分析

中图法分类号 TP319 **DOI号** 10.11897/SP.J.1016.2024.02059

A Review of Smart Fuzzing: Problem Exploration and Method Classification

WANG Qin-Ying[1)] XU Jia-Cheng[2)] LI Yu-Wei[3)] PAN Zu-Lie[3)]

ZHANG Yu-Qing[4)] ZHANG Chao[5)] JI Shou-Ling[1)]

[1)](*College of Computer Science and Technology, Zhejiang University, Hangzhou* 310007)

[2)](*College of Control Science and Engineering, Zhejiang University, Hangzhou* 310007)

[3)](*College of Electronic Engineering, National University of Defense Technology, Hefei* 230037)

[4)](*Department of National Computer Network Intrusion Protection Center, University of Chinese Academy of Sciences, Beijing* 101408)

[5)](*Institute for Network Science and Cyberspace, Tsinghua University, Beijing* 100084)

Abstract With the increasing scale and complexity of software systems in recent years, along with the continuous growth in the number of security vulnerabilities and their expanding impact, the global security situation remains challenging. In response to this issue, academia and industry have been devoted to researching efficient vulnerability discovery techniques to identify and address potential vulnerabilities in advance. Among these techniques, fuzzing has garnered significant attention from academia and industry as an advanced vulnerability detection approach. To further enhance the capability of vulnerability discovery, researchers introduced smart fuzzing, which leverages artificial intelligence and program analysis techniques to assist in more efficiently

收稿日期:2023-07-04;在线发布日期:2024-05-13.本课题得到国家自然科学基金联合重点项目(U1936215)、国家自然科学基金青年基金项目(62202484)、国家自然科学基金联合重点项目(U2336203)资助.**王琴应**,博士研究生,主要研究领域为物联网安全、软件与系统安全.E-mail:wangqinying@zju.edu.cn.**许嘉诚**,博士研究生,主要研究领域为模糊测试与操作系统安全.**李宇薇**,博士,副教授,主要研究领域为软件与系统安全、模糊测试、漏洞挖掘.**潘祖烈**,博士,教授,主要研究领域为网络安全与软件安全.**张玉清**,博士,教授,主要研究领域为网络与系统安全.**张 超**,博士,副教授,中国计算机学会(CCF)杰出会员,主要研究领域为软件与系统安全.**纪守领(通信作者)**,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为人工智能安全、数据驱动安全、软件与系统安全、大数据分析与多媒体理解.E-mail:sji@zju.edu.cn.

testing and analyzing complex software systems, intelligently guiding the direction of vulnerability discovery. This paper reviews the progress of smart fuzzing over the past eight years, proposes a general fuzzing procedure model and a problem-oriented classification method for smart fuzzing techniques, and summarizes the current advantages and shortcomings of smart fuzzing from three aspects: optimizing test input generation, improving test efficiency, and enhancing test oracles. Finally, this paper offers a prospective outlook and summary of the challenges and future research directions in the field of smart fuzzing.

Keywords fuzzing; software and system security; vulnerability discovery; artificial intelligence; program analysis

1 引 言

随着社会信息化的迅速发展,软件系统规模不断增大,同时也涌现出了新应用场景下的新型软件系统,如物联网设备、大型操作系统、数据库、云系统、人工智能系统等.软件系统漏洞已成为信息基础设施面临的首要威胁,大量政企网络被攻击篡改,网站平台大规模数据泄露事件频发,生产业务系统安全隐患突出.因此,漏洞挖掘的重要性和必要性日益凸显.在众多的漏洞挖掘方法中,模糊测试已成为当前最流行的方法之一.模糊测试是一种通过输入恶意或异常数据来挖掘软件系统漏洞的技术.传统的模糊测试方法通常依赖于随机或半随机的数据生成技术来对目标软件进行测试,包括随机模糊测试和代码覆盖率引导的模糊测试.前者实现简单,但测试盲目,生成的多数数据可能与实际输入格式不匹配,导致无法深入软件系统的内部逻辑,测试覆盖率低.后者相比随机模糊测试效率更高,利用代码覆盖率作为反馈,可以有针对性地生成能触及未探索代码区域的测试样例,从而提高测试覆盖范围.然而随着软件复杂度的增加和安全机制的加强,仅依赖代码覆盖率引导的传统模糊测试在测试效率和漏洞检测上仍存在较大局限性,如测试大型软件系统时存在的种子爆炸、测试成本急剧上升的问题,以及在测试复杂逻辑时由于难以触发特殊条件或模拟特定状态而导致漏洞漏报的问题.

近年来,一大批来自学术界和工业界的学者探索了模糊测试存在的不足和软件系统测试的挑战,前瞻性地提出了一系列算法和技术帮助提高模糊测试的效率和漏洞挖掘能力,即智能模糊测试.智能模糊测试概念自 2007 年以来被研究人员多次用于描述结合静态分析、符号执行、人工智能等技术辅助的模糊测试框架[1-3].相比于随机模糊测试和代码覆盖率引导的模糊测试,智能模糊测试通过引入动静态程序分析技术以及人工智能方法,帮助模糊测试工具获得更多关于目标软件系统的知识,智能引导漏洞挖掘方向,能够更好地应对层出不穷的新型漏洞以及大型复杂软件系统的应用场景.此外,智能模糊测试可以减少人力的投入以及由于人工主观性、分析不完备带来的误报和漏报.然而,由于解决问题的角度不同和测试对象的差异,所提出的智能模糊测试技术也各有侧重.目前关于智能模糊测试的技术缺乏系统的整理、归纳、总结、分析.

虽然有学者在 2017～2022 年对模糊测试研究工作[4-11]进行了系统的整理,但是他们或关注某项特定的智能技术,如基于机器学习的模糊测试技术[7],或关注某一特定领域系统的模糊测试,如嵌入式系统的模糊测试研究综述[10].与本文工作最接近的是 Zhu 等人[9]、Manes 等人[6]和 Li 等人[11]对模糊测试的总结.Manes 等人和 Li 等人主要停留在对模糊测试基本流程、测试对象以及可应用的技术的讨论,缺少对已有工作整体化和系统性视角的梳理.Zhu 等人从模糊测试理论、输入搜索空间和自动化测试三个角度梳理已有的模糊测试研究工作,并列举了模糊测试流程中技术点的优化方法.但该分类方法主要适用于基于 AFL 的模糊测试工作,且对三个角度下相关技术的梳理没有形成一个整体性视角,使得读者较难理解模糊测试技术之间的联系,也缺乏对测试预言的详细梳理与分析.另外,近年来研究人员和从业者都投入了大量且多样化的努力来改进模糊测试,这种工作量的激增也使得很难单单从模糊测试流程优化的角度进行分类并获得全面、连贯的模糊测试观点.

为了整理和理解智能模糊测试文献并使其保持连贯性,本文提出了一个通用的模糊测试流程模型

以及新的模糊测试文献的分类方法,主要对 2016—2023 年的研究进展进行深入系统的整理和科学归纳、总结、分析,以便为后续学者了解或研究智能模糊测试提供指导.具体而言,本文提出的模糊测试流程模型包含测试样例生成、测试样例执行和评估以及漏洞测试预言三个关键流程.此外,本文通过广泛调研相关文献,对这些关键环节下的子流程及其工作模式进行了细致的分类和拆解.该流程模型帮助促进了对不同智能技术在模糊测试过程中应用与相互关系的统一理解.基于该流程模型,本文进一步建立了一个模糊测试漏洞挖掘能力模型,明确了影响模糊测试漏洞挖掘效能的三大核心优化问题:复杂测试样例的生成、模糊测试循环的效率提升以及测试预言机预测的准确性.同时,本文研究发现现有工作在不同的模糊测试子流程中创新性地引入了多种不同的智能技术,主要旨在解决这三项核心问题.因此,从问题导向的角度出发,本文对现有的智能模糊测试技术进行系统性分类,提供了一个问题导向的新视角.

本文的主要贡献包括:

(1)提出了一个通用模糊测试流程模型以及问题导向的智能模糊测试文献的分类方法,帮助研究人员对智能模糊测试技术进行整体把握并理解其关联和影响;

(2)对近八年的模糊测试研究进展进行了全面的梳理和总结,包括智能技术在模糊测试中的应用、针对特定场景的模糊测试方法以及对不同类型程序系统的模糊测试研究;

(3)分析最新模糊测试领域的挑战和问题,并提出了未来研究的方向和潜在的解决方法.

本文第 1 节为引言;第 2 节介绍了模糊测试的背景和意义;第 3 节阐述了综述方法;第 4～6 节分别围绕复杂测试输入生成、模糊测试循环的效率优化和测试预言系统地总结和分析了近八年智能模糊测试的研究进展;第 7 节讨论了挑战和未来研究的方向;第 8 节为结束语.

2　模糊测试的背景和意义

模糊测试是一种自动化、动态化的软件测试技术,其核心思想在于向目标系统注入自动或半自动化生成的非预期输入,同时监控目标系统的执行状况,以期触发系统的异常行为从而发现目标系统的缺陷和漏洞.相比于人工分析和静态分析,模糊测试的自动化程度更高、误报率低,已经成为最流行、最有效的漏洞挖掘技术之一,被广泛用于各个领域的测试.截至 2023 年,Google 的开源软件模糊测试平台(OSS-Fuzz)已在 850 个开源软件中识别了超过 8900 个安全漏洞和 28000 个程序缺陷[12].

模糊测试的发展历程如图 1 所示.威斯康星大学的 Barton Miller 教授于 1988 年首先提出了模糊测试的概念,并开发了一个命令行模糊测试工具旨在测试 Unix 程序的健壮性[13].2001 年和 2002 年,芬兰奥卢大学和 Aitel 分别提出了 PROTOS 测试套件[14]和开源模糊测试框架 SPIKE[15],用于网络协议和网络应用程序的测试.此后,针对不同目标应用程序的模糊测试工具不断涌现,如 2004 年针对文件的模糊测试工具 FileFuzz[16]、经典的协议模糊测试框架 Peach[17]等.2008 年,Mozilla 安全团队发布了 Jsfunfuzz[18],提出了基于语法的模板文件来构造复杂测试输入.该阶段模糊测试以黑盒测试为主,通过发送随机畸形测试例,模拟攻击者构造输入等手段触发漏洞,存在缺少反馈,漏报概率高且覆盖率低的问题.

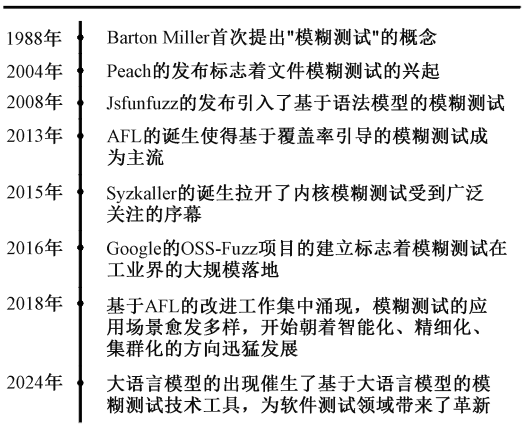

图 1　模糊测试的发展历程

2013 年底,American Fuzzy Lop(AFL)[19]诞生.其独创的源码插桩编译、覆盖率引导的思想标志着灰盒模糊测试时代的到来,也成为模糊测试技术发展历程中最重要的一次里程碑.AFL 采用遗传算法以及精心设计的变异策略,显著提升了测试效率.随后,模糊测试技术进入高速发展阶段.2015 年,Google 开源了 Syzkaller[20].该工具基于预定义的系统调用模板实现,支持测试 Linux 等操作系统内核.2016 年起,Google 建立了 OSS-Fuzz[12]和 ClusterFuzz[21]平台,模糊测试技术在工业界大规模落地.从 2017 年开始,一大批结合粒子群优化、符号执行、注意力模型等智能技术辅助的模糊测试工作大

量涌现，如 CollAFL[22]、MOpt[23]等，从种子选择、能量分配、变异调度等角度优化经典测试框架. 同时，研究人员致力于拓宽模糊测试技术的应用场景，以应对更加复杂的目标系统. 此外，2024 年见证了多个基于大语言模型的模糊测试技术工具的出现. 这些工具基于优化提示生成来优化模糊测试的测试样例生成，在代码覆盖率和自动化水平上体现了显著的优势[24-28].

3 综述方法

3.1 文献收集与分析

为了对智能模糊测试的研究框架建立整体性视角. 本文遵循系统文献调研的流程[29]收集和分析相关文献. 系统文献调研的目标是找到尽可能多和尽可能全面的相关文献. 本文采用了最常见的文献搜索策略即数据库检索和“滚雪球”，选取的文献遵循了以下标准：(1)该文献针对模糊测试的复杂测试输入生成、模糊测试循环的效率优化和测试预言三个优化问题提出了新的技术，且取得了一定效果；(2)该文献对现有的智能化模糊测试进行改进和评估；(3)该文献已经公开发表在国内外的专著、会议、期刊中，并具有一定的影响力.

根据三个标准，本文通过三个步骤筛选文献：(1)根据研究方向确定检索的关键词为 fuzzing、fuzzer、sanitizer、bug oracle 等，确定的中文关键词包括模糊测试、测试预言、漏洞检测等. 检索的数据源包括 IEEE Xplore Digital、Springer Link Online Library、ACM Digital Library 等. 检索时间区间定义在 2016 年至 2023 年；(2)按照标题、关键词、摘要、结论和来源的优先顺序对上述文献进行筛选，其标准为：①根据《中国计算机协会推荐国际会议和期刊目录》，选择 A 类以及 A 类以上的权威网络空间安全和软件工程英文会议及期刊. ②与智能模糊测试技术有关；(3)由第一作者和第二作者对上述文件进行全文排查，并对这些文献通过滚雪球的方式再进行补充和筛选.

通过上述步骤对文献进行筛选后，共得到 132 篇相关文献，这些文献都是与智能模糊测试技术直接相关，对模糊测试三个优化方面的关键技术提供了新思路. 本文对这些收集到的论文进行阅读，建立了一个考察智能模糊测试的整体性视角，分析它们拟解决的问题和优化的模糊测试环节. 图 2 和图 3 分别展示了本文所总结文献发表年份和拟解决关键问题的分布情况. 其中图 2 和图 3 只收录了 2024 年 1 月至 3 月的论文情况. 可见，近五年来，智能模糊测试论文数量增长迅速，大多数论文致力于解决复杂测试输入和模糊测试循环优化问题，而测试预言的研究仍处于初步阶段.

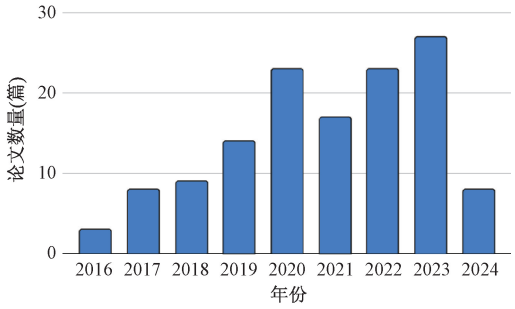

图 2 文献发表时间的分布情况

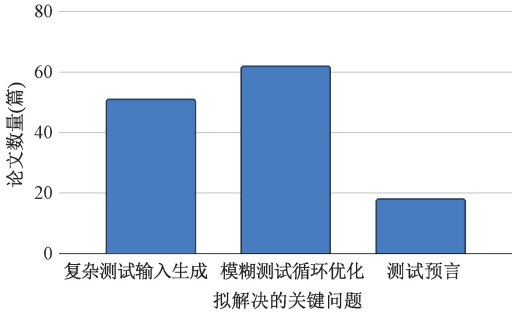

图 3 文献拟解决关键问题的分布情况

3.2 研究思路和框架

根据现有文献的分析，近年来涌现了大量结合静态分析、符号执行、机器学习等智能技术的模糊测试研究. 尽管如此，目前在学术界和工业界对于这些众多模糊测试工作的系统性和连贯性理解仍然存在明显不足. 为此，本文提出了一种通用的模糊测试流程模型，概括了模糊测试技术的能力模型，并识别出三个核心优化问题. 针对这些核心问题，本研究提出了一个问题导向的分类方法，以系统化地评估和指导智能模糊测试技术的发展. 这一方法不仅有助于理解每种技术如何具体贡献于模糊测试的不同阶段，也便于发现当前流程中可能的改进机会.

与现有的模型相比，本文提出的模糊测试流程模型更具通用性和简洁性，易于理解和应用. 当前大多数模糊测试模型通常专注于特定的应用领域或测试环节，缺乏广泛适用性. 如 Zhu 等人[9]提出的模型主要适用于 AFL，包括种子调度、字节调度、变异调度、测试执行、适应度函数等，不适用于对 Syzkaller 等模糊测试工作的分析，且过程相对复杂. 如

图 4 所示,本文构建的模型紧密遵循模糊测试的定义,以测试样例作为核心,包括测试样例生成、测试样例执行和评估以及测试预言三个关键环节.在此模型中,生成的测试样例在目标系统上执行并接受评估,其结果反馈至测试样例生成环节,以指导进一步的测试样例生成,从而形成一个闭环的测试循环.同时,测试过程产生的信息还会被送至漏洞预测环节以判断潜在漏洞的存在.(1)测试样例的生成.该环节的方法主要分为基于生成和基于变异两类.基于生成的策略通常采用预设的测试样例模板,并在测试的初始阶段或过程中根据需求更新这些模板,以明确输入数据的特定特征和结构并生成结构性更强的输入数据.而基于变异的模式,通过对现有有效输入的随机变异生成新的测试样例,涉及种子调度和种子变异两个子环节.其中种子的调度包括种子的选择和能量分配.(2)测试样例执行和评估.包括测试样例的执行和测试数据的收集两个子环节.执行测试样例时,会同步收集覆盖率等反馈信息,这些信息将被用来指导测试样例生成过程中的种子调度、变异、模板更新及测试样例生成.此外,测试过程中控制流和数据流的行为将作为漏洞测试预言环节的输入.(3)测试预言.依据是否存在漏洞的范式,可分为特定错误检测和差分测试两大类.特定错误检测通过使用预定义的错误模式,在目标系统执行过程中对照这些模式实时监测软件系统行为,以识别潜在的错误.另一方面,差分测试采用将同一输入同时送入多个功能等效但实现或版本有异的系统中,通过分析它们输出间的差异来发掘可能的缺陷.该通用模糊测试模型通过将模糊测试流程细分为关键环节,提供了一个清晰的框架,用以识别和定位智能技术在模糊测试中的具体应用点.该结构化方法不仅有助于在后续文献分析过程中理解每种技术如何具体贡献于模糊测试的不同流程,也便于发现当前流程中潜在改进机会.

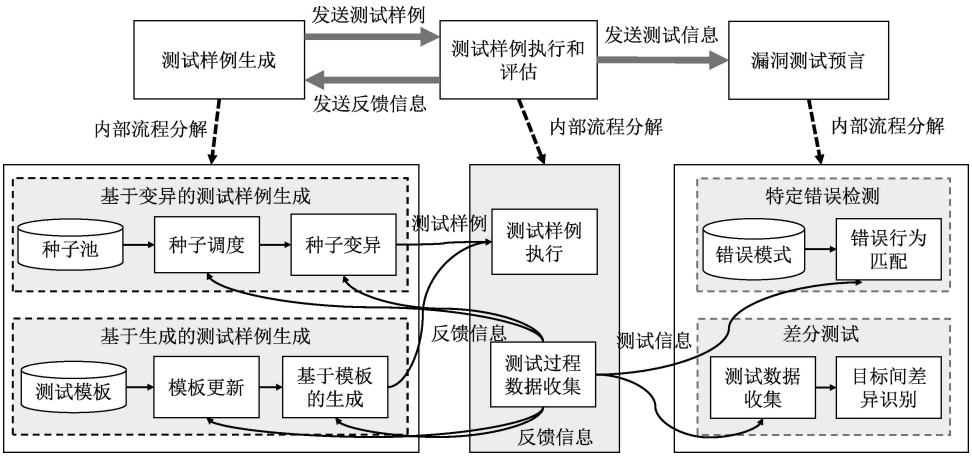

图 4　通用模糊测试流程模型

经过对相关文献的综合分析和对智能模糊测试通用流程模型的深入研究,本文总结了智能模糊测试的漏洞检测能力模型.如图 5 所示,该模型分析了模糊测试优化的三个核心问题与模糊测试技术漏洞挖掘能力的关系.具体而言,测试预言的能力界定了模糊测试工具能够识别的漏洞类型,而复杂测试输入的生成能力则决定了模糊测试可以探索的有效输入空间,这直接影响模糊测试能够发现的潜在漏洞的范围.此外,测试循环效率的优化直接关系到达到潜在漏洞发现上限的速度,从而决定了测试框架的整体效能.

在分类方法方面,以往的模糊测试综述往往按照测试的不同阶段进行划分,例如种子选择、能量分配等.这种方法的限制在于,同一测试阶段内的不同

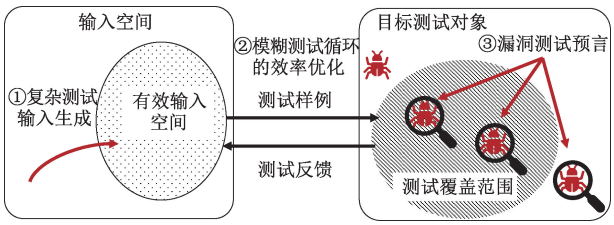

图 5　智能模糊测试的漏洞检测能力模型

智能技术可能旨在解决各异的问题,这使得基于测试阶段进行分类的研究综述难以有效地对比不同技术的优劣.相比之下,本文采用的分类框架以问题为出发点,使得对于解决相似问题的模糊测试技术的分析和比较变得更为直接和有效,同时也便于更全面地概括现有模糊测试工作的能力和特点.如图 6 所示,本文根据智能技术在模糊测试中的应用目

的——即解决模糊测试的三大核心问题及其衍生子问题进行了系统分类.通过对众多文献的综合分析,本研究清晰界定了各种现有智能技术在解决特定模糊测试问题时实施的优化策略及其对测试流程的具体改进.本文关于三大核心问题的挑战及其衍生子问题总结如下:(1)复杂测试输入生成优化旨在提高测试样例的有效性以扩大代码覆盖范围.在这一过程中,研究者面临的挑战包括严格的输入空间限制和有限的约束求解能力.具体而言,软件系统的输入往往具有复杂的结构性和依赖性,例如协议接口和内核系统调用不仅要求特定的参数和参数类型,而且还需要一系列特定的输入顺序才能触发深层次的代码逻辑.本文概括了针对这一问题的子问题,包括语法语义建模、约束关系求解和依赖关系推测.这些子问题的解决策略凸显了对结构化输入的深刻理解,并提出了相应的智能化方法来构建和更新测试模板,以生产满足复杂软件要求的精确测试样例.针对该问题,相关智能技术的应用主要集中在测试模板的构建、模板更新、基于模板的测试样例生成、种子变异的子流程中.(2)模糊测试循环的效率优化.旨在提升测试的速度和覆盖率.这一过程中,研究者面临的挑战包括测试样例变异盲目和程序空间庞大复杂.具体而言,在处理多达数万行代码的软件系统时,输入的复杂性及程序内嵌套的条件语句等约束关系,经常导致现有模糊测试工具无法高效地探索所有可能执行的路径.这些工具在尝试达到满足复杂约束条件的输入时,往往浪费大量资源.本文概括了针对这一问题的子问题,包括反馈机制优化、高效路径探索、路径约束探索、自适应优化策略和复杂程序空间探索.针对该问题,相关智能技术的应用主要集中在测试过程数据收集、种子调度和变异等子流程中,用于加强测试过程数据的收集、优化种子数据的调度以及提升变异策略的针对性.(3)测试预言优化旨在提升漏洞检测的检测率和准确性.当前模糊测试的漏洞检测能力受限,往往需要定义漏洞特征及触发模式设计特定的漏洞检测器.然而针对云服务程序和物联网设备以及内核等复杂程序,传统的测试预言难以覆盖新型的漏洞,如越权访问等.此外,测试过程需要控制和收集程序运行时信息,往往会引入较大开销,且无法发现未被定义的漏洞.本文概括了测试预言的子问题包括错误检测器和差分测试.针对该问题,相关智能技术的应用主要集中在如何扩展错误检测器的错误模式以及增强差分测试的设计.

在随后的章节中,本文将沿用本节所提出的分类方法,深入探讨智能技术在解决前述核心问题及其衍生子问题中的应用成效和对应的优化流程.通过对最新研究进展的总结和分析,旨在为后续研究提供指导,并推动智能模糊测试的发展.

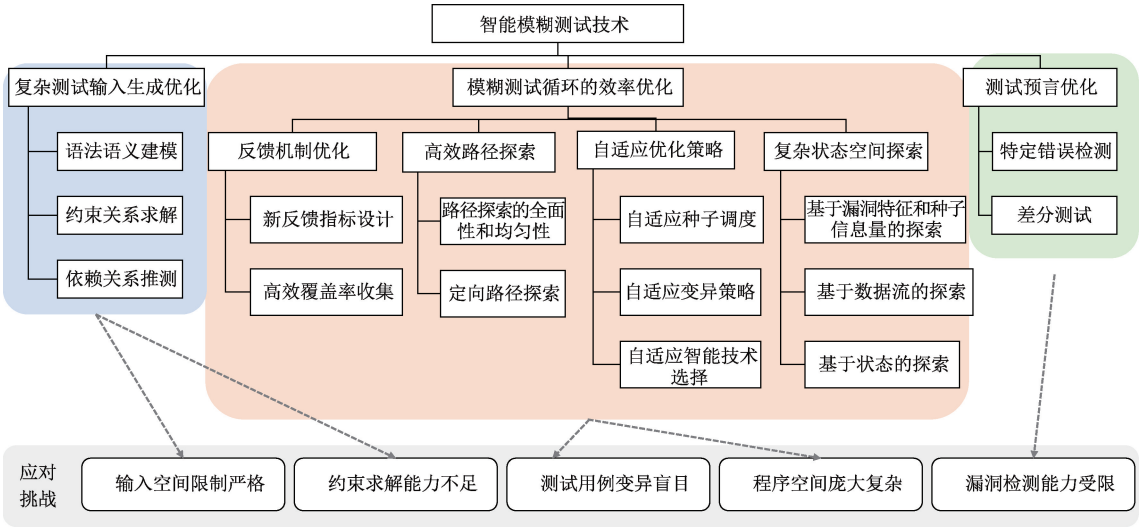

图 6 智能模糊测试分类方法

4 复杂测试输入生成

4.1 语法语义建模

面对模糊测试在处理结构化输入的挑战,尤其是生成满足语法和语义要求的测试用例,复杂测试输入生成的首要问题便是如何进行有效的测试样例语法语义建模.为了提取语法和语义知识,当前智能模糊测试技术根据知识源的差异,主要分为程序分析、文档解析、样本学习,以及大语言模型的应用.此

外,智能模糊测试方法还关注知识表示方法的选择、中间表示形式的优化,以及开发能够精准理解语法和语义要求的测试输入生成算法.这些策略致力于在不破坏输入结构的前提下,增强测试样例探测深度和效率.

(1)基于程序分析的语法语义模型提取.基于程序分析的语法语义模型提取方法在测试具有特定语法语义要求的输入上,如操作系统内核和物联网固件的测试输入,发挥着关键作用.研究人员利用静态和动态分析技术对被测对象的接口和相关程序进行自动化分析,推断所需输入格式,从而构建可靠的输入模型[20,30-36].例如,DIFUZE[31]通过静态分析技术尝试从接口类型列表中实现操作系统的系统调用处理程序的结构推断,并配合程序设备文件名配对等智能化技术来帮助生成系统调用的模板.然而,由于静态分析仅能覆盖显式的语法语义规则,DIFUZE对系统调用处理程序结构的识别存在漏报和漏报问题.动态分析方法,如 KSG[32]和 CoLaFUZE[37],通过扫描设备文件并在运行时识别系统调用处理函数,结合符号执行来识别命令值和参数类型,旨在克服静态分析的局限.尽管这些动态分析技术在某种程度上解决了误报问题,但仍存在一定的漏报问题.这是因为即使动态分析能够捕获运行时数据,它也需要对编程约定有足够的建模支持,否则仍难以完全恢复所有系统调用处理函数的细节.针对闭源的物联网固件,IoTFUZZER[38]和 DIANE[39]设计了智能化算法来提取相应的输入模型.IoTFUZZER从物联网设备的配套应用出发,绕过直接的语法语义推断,有效地生成满足程序要求的复杂输入.DIANE[39]则在 IoTFUZZER 的基础上进一步优化了算法,精确定位配套应用程序的关键代码位置,为IoT 设备生成高效测试输入.

(2)基于样本学习的语法语义模型提取.基于样本学习的语法语义模型提取是一种直接从测试样例的样本中学习输入格式的方法[40-44].Wang 等人[40]提出的 Skyfire 方法,利用数据驱动的技术从样本集合和语法中自动学习带概率的上下文相关文法,进而生成具有高覆盖率的种子.随着深度学习在图像和自然语言处理领域的突破,研究人员开始探索使用深度学习技术来推进此类任务,如 Learn&fuzz[43]、GANFuzz[45]、SmartSeed[42]等.特别是,Learn&fuzz[43]从大量 PDF 文件中抽取 PDF 对象作为样本,使用深度循环神经网络从样本中学习 PDF 文件规范,继而生成合法的 PDF 文件用于测试.此外,针对工业控制协议格式,GANFuzz[45]通过报文聚类、生成对抗神经网络建模、模型采样三个步骤,生成更多样化且格式良好的测试样例.这类基于样本学习的智能模糊测试技术显著依赖于可用样本的数量和质量.样本的多样性和代表性直接影响到学习得到的模型的泛化能力和测试的有效性.因此,获取广泛、多样化的样本集合是实现有效模糊测试的关键.此外,样本中的噪音和异常数据也需要通过预处理步骤仔细筛选和处理,以确保输入模型的准确性和测试结果的可靠性.

(3)基于文档解析的语法语义模型提取.基于文档解析的语法语义模型提取是一种通过解析标准规范和文档来低成本构建输入模型的有效方法,主要应用自然预言处理等智能技术.例如,Ntfuzz[46]利用 Windows 函数应用程序编程接口(API)文档结合静态分析的结果,辅助生成测试样例来测试 Windows 内核.通过智能化迭代累加的方式,Ntfuzz 从有限的文档资源中能够补全并获得系统调用函数的完整参数信息.CarpetFuzz[47]发现不同的程序命令行参数组合对应着不同的分支路径,而这些参数信息往往蕴含于程序文档.它利用自然语言处理技术从文档中提取命令行参数并筛选出其中的有效参数组合用于测试启动,有效探索了参数相关的深层代码.此外,DocTer[48]和 TitanFuzz[24]采用了文档辅助的测试方法,从深度学习框架文档中提取框架API 输入所需满足的约束,包括参数个数和类型等,以指导不同的变异策略.类似的,COMFORT[49]针对 JavaScript(JS)引擎,参考 JS 语言规范,通过正则表达式获取一系列类似伪代码的 JS API 标准以验证各个厂商 JS 引擎解析的语义正确性.此类智能模糊测试技术虽然在提高测试效率和精确性方面表现出色,但它们依赖于文档的准确性和完整性.这种基于文档解析的智能模糊测试技术主要面临的挑战是无法充分应对实际实现与文档描述不符的情况.当软件实际行为与文档或规范存在偏差时,这种方法可能无法有效揭示软件中的漏洞或错误.此外,这种依赖文档的方法也可能忽略了由于开发人员的实现错误或误解规范而产生的非预期行为.

(4)结合大语言模型的语法语义模型提取.近年来以 GPT 系列为代表的大模型技术发展突飞猛进,在文本生成、知识问答、代码理解等方面显示了卓越的性能.已有研究结合大模型技术来提升模糊测试的智能度[24-26,28,49-50].最早的工作之一 COMFORT[49]收集大量 JS 程序作为语料库微调 GPT-

2 模型[51]，使其能够针对性地突变给定的函数. 随后的研究工作大多围绕 ChatGPT、GPT－3.5 展开. 例如，针对深度学习库 API，TitanFuzz[24] 专门结合输入规范设计提示文本，利用在线大模型生成了一批高质量种子，再利用本地大模型开展四种类型的突变. 研究结果表明，借助精心设计的提示文本，大模型直接生成的种子已经表现出强大的漏洞挖掘能力. 后续的 FuzzGPT[50]、Fuzz4all[28] 等仍然都遵循这种“大模型＋提示词”策略直接生成测试样例的范式. 除了生成输入，Zhang[26] 等人和 Liu 等人[25] 几乎同时创新性地提出了大模型驱动的测试驱动构建任务. 他们发现测试驱动的缺乏是开源软件持续性模糊测试工具等基础设施的瓶颈. 对此，他们设计了更加复杂详尽的提示文本，引导大模型自动化地为开源库 API 生成测试驱动，更好地支持下游测试基础设施.

(5)复杂输入的中间表示优化和生成算法. 现有的如 Syzlang 等系统调用描述语言在处理编程语言这类语法规范严格、语义空间复杂的输入时显示出局限性. 针对这一挑战，研究人员提出了基于中间表示的测试样例构建方法[52-56]，设计与语法等效的中间表示，将测试样例转化至中间表示后进行语法语义有效的突变，以平衡可靠性与多样性. 例如，针对数据库管理系统，SQUIRREL [53]引入了面向结构化查询语言的中间表示，采用分而治之的思想，基于智能翻译技术将结构化查询语句翻译为若干个中间表示的组合，每个中间表示是一条包含了赋值对象、常量和操作符的静态单赋值语句，通过更新中间表示本身或操作数执行语义有效的变异. 针对网页文档，Xu 等人[54]设计的基于上下文的智能中间表示方法维护了文档对象模型树数据依赖、层叠样式表规则和事件响应列表，能够生成语义正确的网页文档. 针对 JS 引擎，Fuzzilli[52]通过设计实现专门面向即时优化漏洞的中间表示，改善语法树层面突变缺乏语义感知的问题，显著提升对即时优化部分的测试能力. 上述方法大都适用于单一类型输入. Chen 等人[56]提出了面向通用语言的编译器测试方法 POLYGLOT. POLYGLOT 结合语法规范和语义注释，支持将多种编程语言转换为统一中间表示进行突变，减少为不同语言设计测试器的重复劳动. 最后，结构化输入通常按规则由基本片段(如词元、语法树节点、字段)组织而成. 针对这一特点，一个可行的方案便是通过分割、重组结构化输入获得合法且多样化的测试样例[57-61]，如 Montage[59]利用深度学习技术指导基本片段的组合，通过分割 JS 样本的语法树子树得到数据集，用以训练神经网络语言模型，最后由模型扩展上下文生成新的测试样例. 尽管基于片段重组的方法减少了对复杂领域知识的要求，但其测试样例生成的正确率通常低于基于输入模型的方法.

综上所述，针对结构化输入，基于语法语义建模的测试样例构建方法展示了其有效性. 在开源系统的处理上，研究人员通常通过分析系统的内部结构或解析相关规范和文档来构建这些输入模型. 对于闭源系统和物联网应用场景，获取输入模型相对困难，需利用额外的知识如分析配套应用等. 此外，选择合适的生成算法对测试效率至关重要，不同的工具已经提出了各种策略，旨在平衡测试样例的可靠性与多样性. 然而，如何设计高效的智能化技术如大语言模型、符号执行等，提高模糊测试工具在闭源场景下或针对具有复杂输入格式的测试对象的测试性能，仍是未来研究的重要方向.

4.2 高效约束求解

复杂测试输入生成的一个关键子问题是如何生成满足给定路径约束条件的输入数据，使得目标程序可以覆盖更多程序路径. 路径约束条件包括魔术字节、校验和以及嵌套约束，其中魔术字节是用于标识特定文件格式或数据结构的固定字节序列. 图 7 展示了涉及魔术字节和嵌套路径的示例. 目前，解决这一问题的智能模糊测试技术主要可以分为两种策略：基于符号执行和基于主动推测的方法.

```
1 …
2 read(fd, buf, size);
3 // notice the order of CMPs
4 if (buf[5] == 0xD8 && buf [4] == 0xFF)
5     … some useful code …
6 else
7     EXIT_ERROR(“Invalid file \n”);
```

魔术字节约束

```
1 …
2 read(fd, buf, size);
3 …
4 if (…) {
5     If (…) // nested IF
6         …
7 } else {
8     …
```

嵌套约束

图 7 魔术字节和嵌套路径问题[62]

(1)基于符号执行的智能模糊测试技术. 该技术结合了实际执行与符号化执行程序，用于收集程序路径的约束条件，并通过约束求解器为每条路径生成适当的测试样例. 尽管模糊测试能快速产生多样化的测试样例，它却难以通过随机变异直接生成满足复杂路径约束的用例；而符号执行，虽然在处理复杂分支条件上非常有效，却也面临着高开销和状态爆炸的挑战. 为了克服各自的缺点并利用它们的优势，研究人员提出了混合模糊测试方法：利用模糊测试的高效性来生成大量输入并快速探索程序的主要

路径,遇到复杂约束时使用符号执行进行精确求解.例如,Driller[63]结合了 AFL 与 angr[64],成为最早将符号执行辅助模糊测试求解的工作.继 Driller 之后,研究人员主要围绕约束求解方法和求解任务调度等提出了多种改进策略[65-71].特别地,在约束求解方面,Yun 等人[66]设计的 QSYM 引擎,采用动态二进制翻译、部分约束求解等方法,提高了符号执行的效率,更好地适应模糊测试的任务.DigFuzz[70]通过污点分析追踪执行指令,仅对关键指令执行符号模拟,极大简化了执行和约束求解的过程,并专注于解决实质性的复杂约束问题.此外,Huang 等人[69]指出现有混合模糊测试技术中的计算冗余问题,并提出了一种称为多边形路径抽象的方法来保留并复用探索过程中的约束求解结果,实现增量的混合模糊测试以提升求解效率.

(2)基于主动推测的智能模糊测试技术.针对未开源无法应用符号执行的测试对象,研究人员提出了基于主动推测的智能模糊测试技术,结合动静态分析和机器学习技术,帮助模糊测试工具在迭代测试过程中自动调整变异策略,以生成满足路径约束的测试样例,并进一步探索更深层次的路径.针对魔术字节和校验和问题,现有研究主要通过污点分析等技术识别魔术字节的位置和约束条件,通过关键字节插桩、覆盖率反馈等技术指导模糊测试生成能解决约束的测试样例[62,72-73].例如,VUzzer[62]通过动态污点分析,来确定输入中与魔术字节及错误处理代码相关的部分,并对这些部分进行针对性的变异操作,从而提高路径探索深度.此外,它还采用基于路径深度的种子选择策略,优先选择能深入探索的种子.虽然 VUzzer 可以有效支持闭源的目标程序的约束求解,但它对魔术字节的识别依赖于精确的污点分析和逆向工程,且不能处理由非连续输入字节影响的魔术字节问题.相比之下,steelix[73]采用轻量级的静态分析技术识别用于魔术字节比较的指令,并通过二进制插桩获取运行时比较值.这些值的变化指导模糊测试选择有助于解决约束的输入,并推测影响魔术字节变化的输入字节位置,从而指导变异策略.REDQUEEN[72]则通过跟踪比较指令和分析输入位置数据的变化对程序状态的影响来增加代码覆盖率.REDQUEEN 的设计减小了传统动态污点分析的开销,但其推测关键字节位置技术仅适用于输入和程序状态是简单映射关系的场景,无法求解复杂哈希索引的约束.针对嵌套约束问题,研究人员设计了控制流分析、自适应输入探索和复用历史数据等智能技术来优化种子变异策略[74-78].例如,Chen 等人[74]提出了 Matryoshka,使用静态分析技术识别条件语句之间的控制流依赖,并通过智能策略满足多层条件语句.GREYONE[75]则采用数据流敏感的模糊测试技术,通过字节级污点推断优化输入字节的变异策略.除了通过动静态技术分析依赖关系外,Angora[76]在字节级推断后,利用梯度下降自适应地搜索能触达未探索边的输入数据,来帮助满足约束条件.而 T-Fuzz[77]采用直接修改程序代码的方法来绕过难以解决的路径约束.

综上所述,现有研究通过结合符号执行技术显著提高了模糊测试在探索复杂路径上的智能化程度.然而,尽管有些研究尝试将这些约束求解方法应用于大型软件系统[79],这些策略大多还未在大规模软件环境中得到广泛验证.在大型软件系统中,运行时分析的成本可能非常高.因此,如何在较低开销下处理大型系统中复杂的数据流,并找到满足路径约束的输入,仍然是一个亟待解决的问题.此外,现有的基于主动推测的约束求解策略虽然支持学习多个离散输入与魔术字节之间的关系,但如何有效提升这些策略在学习约束条件与输入关系方面的效率,仍是一个开放的研究领域.

4.3　依赖关系推断

依赖关系推断是复杂测试输入的关键子问题之一.针对操作系统内核、数据库等复杂测试对象,程序内共享数据状态的存在使得输入之间存在着依赖关系,特定代码路径的触发依然需要满足特定的输入序列.例如,内核系统调用的执行依赖先前系统调用所积累的内核状态,数据库 SQL 语句的执行依赖先前 SQL 语句所创建的数据模型.通过智能化的依赖关系推断,研究人员能够减少无效的输入序列,从而优化复杂测试输入的生成空间.

操作系统内核系统调用数量庞大,组合复杂多样,解决内核系统调用间的依赖关系一直是优化模糊测试输入生成的关键[33,79-88].IMF[80]和 SyzGen[33]通过追踪和分析系统调用执行日志,提出了面向闭源内核的依赖推断方法,主要捕捉系统调用间的显式依赖.显式依赖通过参数直接表达,隐式依赖的判定则更为复杂.MoonShine[81]和 HEALER[82]分析了 Linux 内核系统调用间的显、隐式依赖关系.MoonShine[81]使用静态方法分析系统调用参数和内核共享数据,以此推断系统调用间的显式和隐式依赖,但存在误报率高的问题,容易识别到不正确的

隐式依赖，且对指针分析不够准确. HEALER[82]结合了动态分析，通过参数分析和测试过程中覆盖率变化分别判定显式和隐式依赖，提高了依赖检测的准确性，并根据依赖指导变异. 除了内核之外，研究人员对应用程序 API 间的依赖推断做了研究[83-84,89-91]，覆盖了包括内核、浏览器、深度学习库、云服务等多种应用程序，显著改善了模糊测试对序列化空间输入的搜索能力. Zhang 等人[83]研究了研究闭源 SDK API 间的依赖，利用库的元信息和动态执行信息，解析并推测 API 间的显式数据依赖和隐式控制依赖，以辅助生成 SDK 的测试驱动程序. RESTler[92]将 API 依赖表示为集合关系，使用生产者－消费者模型，结合接口规范推断云服务 API 的依赖，据此构造请求序列. Ispoglou 等人[93]设计了库 API 测试的驱动生成工具 FuzzGen. 如图 8 所示，FuzzGen 利用库的头文件和使用代码来推断库的 API，构造抽象 API 依赖图，并基于依赖图生成测试驱动.

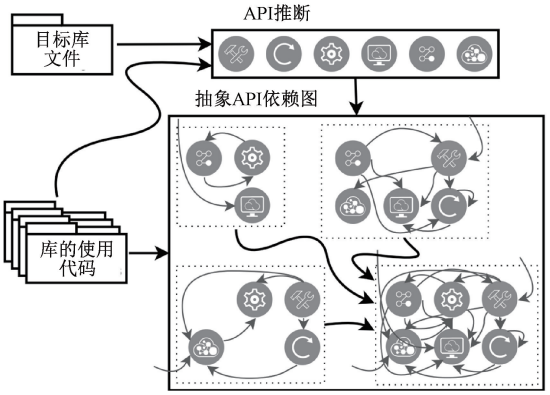

图 8 FuzzGen 构造的抽象 API 依赖图[93]

综上所述，研究人员已经在显隐式依赖挖掘、依赖关系建模等方面做出了许多依赖推断相关的工作. 如何将依赖推断相关的策略进一步推广到更加广泛的应用场景，如区块链交易序列、协议请求队列等，以辅助模糊测试，仍然需要进一步解决. 此外，表 1 对比了用于复杂测试样例生成的各种智能模糊测试技术.

表 1 用于复杂测试样例生成的经典智能模糊测试技术

名称	依赖框架	类别	测试目标	优化类别	关键技术	生成效果	不足之处
DIFUZE[31]	—	G	系统内核	Y	静态分析内核驱动，构建内核驱动接口模型	有效推断内核驱动接口信息	不支持闭源系统，误报漏报较高
Skyfire[40]	—	G	编译器	Y	从样本集合和语法中学习带概率的上下文有关文法，指导种子生成	优化种子生成；相比 AFL 提升 15%代码覆盖；	依赖输入语法；复杂格式输入的生成效果较差
QSYM[66]	AFL	M	通用程序	C	采用动态二进制翻译、部分约束求解等方法	显著提升执行效率；相比 Driller 提升 82.5%代码覆盖	仅支持 x86 架构
MoonShine[81]	Syzkaller	MG	系统内核	D	使用静态方法分析系统调用参数和内核共享数据，分别推断系统调用间显、隐式依赖	支持推断隐式依赖；相比 Syzkaller 提升 13%代码覆盖	不支持闭源内核；推断结果可能存在误报、漏报
CodeAlchemist[57]	—	G	编译器	Y	分割输入样本构建片段池，根据片段间约束条件实现语义感知的代码片段重组	相比 jsfunfuzz 提升了 470%漏洞发现	语义准确性较低
RESTler[92]	—	G	云服务	D	将依赖关系表示为集合，使用生产者－消费者模型并结合接口规范推断云服务 API 的依赖	自动化程度高；支持多种漏洞类型	对复杂测试序列的构造效果较差
DigFuzz[70]	AFL	M	通用程序	C	符号执行调度，将其更多用于求解真正的复杂路径约束	相比 Driller 提升 18.9%代码覆盖	增加了运行开销；路径排序时未考虑约束可解性
REDQUEEN[72]	AFL	M	通用程序	C	通过动态推测输入和约束条件的关系设计变异操作，生成满足约束条件的测试样例	相比符号执行和污点分析更加轻量化	无法处理输入和状态空间不对应问题
GREYONE[75]	—	MG	通用程序	C	通过监控分支变量，间接推断影响分支变量的输入字节，并据此调整输入字节和探索分支的优先级	相比 AFL 等工具增加了 1.2 倍的代码覆盖和 1.5 倍的漏洞发现	存在污染不足问题
FUZZGEN[93]	LibFuzzer	G	调用库	D	利用库的头文件和使用代码来推断库的 API，构造抽象 API 依赖图指导测试	自动化程度高；相比手写驱动提升 7%代码覆盖	API 依赖图存在误报；只支持单一库

（续表）

名称	依赖框架	类别	测试目标	优化类别	关键技术	生成效果	不足之处
FANS[30]	—	G	内核	YD	自动化扫描安卓服务,提取服务接口模型,并推断其中的接口依赖与变量依赖	支持对安卓原生服务测试接口的建模	不支持闭源对象,存在一定漏报
SQUIRREL[53]	AFL	M	数据库	Y	将 SQL 语句转换为静态单赋值语句的组合,设计基于中间表示的变异	输入有效性高;相比 AFL 等工具取得 2—10 倍覆盖率增长	对特定数据库的特定语法的支持度不够
SyzGen[33]	Syzkaller	GM	闭源内核	YD	迭代精炼推断闭源内核系统调用;追踪和分析系统调用的执行日志来推断显式依赖	相比 IMF 提升 15% 代码覆盖;支持闭源系统覆盖率获取	无法推断隐式依赖;依赖执行日志
COMFORT[49]	—	GM	编译器	Y	微调大型预训练模型,结合标准规范指导 JS 脚本的变异	输入有效性高;有效发现 Javascript 引擎不一致性漏洞	缺乏覆盖率引导;微调模型成本高
POLYGLOT[56]	—	M	编译器	Y	结合 BNF 语法和语义注释,将各种编程语言转换为统一中间表示进行变异	支持多种类型编译器;相比 AFL 取得 30 倍覆盖率增长	语法处理难以处理不一致性;变异规则受限
HEALER[82]	—	GM	系统内核	D	使用静态的参数分析和动态的覆盖率变化,分别判定系统调用间显、隐式依赖	相比 Syzkaller、Moonshine 提升 28%、21%代码覆盖	依赖人工书写的系统调用模板
TitanFuzz[24]	—	GM	深度学习库	Y	使用专门的提示文本引导大模型生成合法的深度学习库调用	覆盖更多 API 调用模型的构建	缺乏覆盖率支持

注:M=基于变异,G=基于生成,Y=语法语义建模,C=路径约束解决,D=依赖关系推断.

5　模糊测试循环的效率优化

模糊测试循环优化指的是通过测试过程中的反馈信息调整测试样例的生成策略,以提高测试效率,在定量的测试时间内发现尽可能多的漏洞.其衍生的关键子问题包括反馈机制优化、高效路径探索、自适应路径探索和复杂程序空间探索.

5.1　反馈机制优化

近年来模糊测试最主要的反馈指标是覆盖率.它衡量测试的广度和深度,以及对被测试软件的代码覆盖情况.常见覆盖率指标包括语句覆盖率、分支覆盖率、条件覆盖率等.针对反馈机制优化,可进一步将现有工作分为高效覆盖率收集和新反馈指标设计.前者通过解决哈希碰撞、插桩优化等技术优化覆盖率的计算速度和准确度,后者因特殊漏洞检测和复杂系统测试的需求,提出了资源消耗量、数据流、状态等新指标来引导模糊测试循环.表 2 对比了集成了不同反馈机制的模糊测试技术及框架.

(1)反馈收集机制优化.现有的智能模糊测试技术可分为覆盖准确率优化、插桩优化和硬件特征辅助追踪.Gan 等人[22]提出了一种智能化覆盖率计算方法,动态结合目标程序中的基本块关系计算分支哈希值,减少了哈希碰撞.进一步地,UnTracer[94]、Zeror[95]和 RIFF[96]采用动态删除已访问基本块的插桩或自修改技术,显著减少了测试的运行开销.此外,硬件辅助追踪技术,如 kAFL[97]、uAFL[98]和 SyzTrust[99]利用硬件特性进行非侵入式的覆盖率追踪,避免了传统插桩的兼容性问题.GDBFuzz[100]则利用 GNU Debugger(GDB)的特性,通过动态地对程序基本块设置断点,并检查测试样例触发的断点情况来收集覆盖率.

(2)新反馈指标设计.为了更深入地探索程序的复杂性,研究人员提出了执行时间[101-102]、内存访问数量[103]、程序结构特征[104]、程序状态[105]、漏洞触发条件[106]和漏洞潜在的位置[107]等新指标.以较难检测的资源消耗型漏洞为例,SlowFuzz[101]、Perffuzz[102]和 Memlock[103]在覆盖率的基础上专注于通过资源消耗指标来发现资源耗尽漏洞,如测试样例执行时触发的指令数量和资源的使用量等.在探索新的反馈指标的同时,研究人员还设计了多种智能算法和技术来引导种子调度和变异策略.本文将在后续章节中围绕这些智能模糊测试工作的科学问题进行总结和评估,主要从高效路径探索、自适应优化策略和复杂程序空间探索三个方面展开.

综合上述分析,尽管现有研究显著提升了覆盖率的计算效率和测试的智能化程度,但在大型软件系统中,覆盖率收集仍面临效率低和精确度不足的

挑战.一个潜在的解决方向是采用分而治之的策略,通过模块化分解程序,并设计智能化算法合并各模块的反馈信息,实现高效的模糊测试.此外,对于硬件辅助技术和传统插桩技术的系统性评估仍不足,需进一步探讨如何将这些技术应用于新型软件系统,以提高覆盖率收集效率和计算准确度.

表 2 用于模糊测试反馈机制的经典技术及测试框架

名称	依赖框架	反馈指标	测试目标	优化类别	关键技术	优化效果	不足之处
CollAFL[22]	AFL	边覆盖率	通用程序	高效覆盖率收集	自适应覆盖率计算方法:根据基本块特征为哈希计算公式添加参数以避免哈希碰撞	比 AFL 降低了 75% 的哈希碰撞	(1)不支持闭源程序; (2)难以识别间接调用导致边覆盖率准确率降低
UnTracer[94]	AFL	基本块覆盖率	通用程序	高效覆盖率收集	测试过程中动态删除已经访问过的基本块插桩代码	针对闭源和开源程序分别减少 1300% 和 70%的开销	(1)不支持边覆盖率; (2)不支持命中次数的反馈; (3)兼容性受限于插桩技术
Zeror[95]	AFL	边覆盖率	通用程序	高效覆盖率收集	(1)针对二进制程序采用自修改代码删除已访问过的检测点; (2)自适应切换新机制和默认机制实现准确率和速度的平衡	(1)针对闭源程序,比 AFL 和 MOPT 等执行速度提升了 159%; (2)黑盒场景下比 Untracer 提升了 20.84 的覆盖率	兼容性受限于插桩技术
RIFF[96]	AFL	边覆盖率	通用程序	高效覆盖率收集	(1)基于静态分析技术将运行时的分支编号的计算等操作转换为编译时插桩方法; (2)将插桩代码缩减到单指令 6 字节	相比 AFL 和 MOpt,测试速度平均提升了 268%	不支持闭源程序
kAFL[97]	AFL	边覆盖率	系统内核	高效覆盖率收集	(1)基于 VT-x 虚拟化提高虚拟化效率 (2)基于 PT-Trace 收集覆盖率	(1)PT 解码速度比 Intel ptxed 提升近 30 倍; (2)执行速度相比于 TriforceAFL 提升近 48 倍	(1)兼容性受限于 CPU 对 Intel-PT 的支持; (2)解码 PT 数据包重建控制流代价高昂
uAFL[98]	AFL	基于 LCSAJ 的边覆盖率	固件驱动	高效覆盖率收集	(1)基于 ETM 硬件特性无侵入地收集覆盖率; (2)设计了基于 LCSAJ 的边覆盖率计算方法,避免重建控制流的开销	支持对闭源固件驱动的覆盖率收集	(1)兼容性受限于硬件开发板对 ETM 的支持; (2)设备上的测试受限于设备的计算速度,测试效率远低于基于仿真的测试框架
SyzTrust[99]	Syzkaller	基于 LCSAJ 的边覆盖率和状态覆盖率	可信内核	高效覆盖率收集	(1)采用 ETM 硬件特征及基于 LCSAJ 的边覆盖率计算方法收集覆盖率; (2)基于硬件调试器收集状态覆盖率	比 Syzkaller 在代码覆盖率上提升 66%,在状态覆盖率上提升 651%	(1)兼容性受限于硬件开发板对 ETM 的支持; (2)设备上的测试受限于设备的计算速度,测试效率远低于基于仿真的测试框架
GDBFuzzer[100]	LibFuzzer	基本块覆盖率	固件驱动	高效覆盖率收集	通过 GDB 设置断点收集覆盖率	基本块覆盖率比 Fuzzware 平均提升了 48%	(1)设备上的测试受限于设备的计算速度,测试效率远低于基于仿真的测试框架; (2)覆盖率准确性依赖逆向工具对控制流图分析的准确性; (3)边覆盖率的支持开销较大
SlowFuzz[101]	AFL	边覆盖率、指令数量与资源使用量	通用程序	新反馈指标设计	提出新时间消耗反馈指标,指引 Fuzzer 去产生占用资源更多、执行更多指令的输入	有效发现现实生活中应用程序的资源消耗型漏洞	(1)反馈较为单一,效率相对低; (2)检测的漏洞类型单一

续表

名称	依赖框架	反馈指标	测试目标	优化类别	关键技术	优化效果	不足之处
Perffuzz[102]	AFL	边覆盖率及程序特定行为统计值	通用程序	新反馈指标设计	提出新时间消耗反馈指标,包括分配字节数、I/O 操作数、缓存未命中数或页表错误数等	生成的总执行路径长度比 SlowFuzz 提升 1.9 到 24.7 倍	(1)运行时监控的反馈类型较多,导致开销较大; (2)检测的漏洞类型单一
Memlock[103]	AFL	边覆盖率及内存读写统计值	通用程序	新反馈指标设计	提出新空间消耗反馈指标,引导 Fuzzer 生成消耗更多空间资源的输入	和 AFL、AFLplus、PerfFuzz 等相比,检测到的空间资源漏洞数量提升 17.9%	(1)运行时监控的反馈类型较多,导致开销较大; (2)检测的漏洞类型单一

5.2　高效路径探索

高效路径探索指的是如何设计智能算法来利用程序执行知识,设计更好的种子调度和变异策略,从而快速探索程序的所有执行路径.现有智能模糊测试研究可以进一步分为路径优先探索和选择性路径探索.

(1)路径优先探索.模糊测试的路径探索过程中存在以下两个问题.首先,模糊测试可能会过度关注一些常见路径,导致程序中的一些路径很少被探索.其次,模糊测试可能缺少对一些重要路径的探索和测试.针对第一个问题,一大部分研究人员基于历史路径探索情况,通过改进种子选择、能量分配和变异策略来指导智能模糊测试增加对低频路径的探索.如 AFLFast[108] 和 CollAFL[22],会优先选择那些执行低频路径并且被选择次数较少的种子进行模糊测试,同时也对这些种子分配更多的能量. FairFuzz[109] 通过自适应的变异策略,使其产生的输入更容易遍历到程序的低频路径.针对第二个问题,研究人员在覆盖率度量的基础上提出了新的反馈指标,指导模糊测试增加对特殊路径的探索.如 Wang 等人[110] 指出单一的覆盖率度量过于粗粒度,无法准确的描述程序的执行状态,也无法有效区分不同路径的重要性.因此,他们提出了从函数、基本块和循环三个维度的度量方法,并以敏感内存操作的命中数作为种子选择的优先级指标. K-Scheduler[104] 则采用了图中心性分析的方法,用于评估当前种子在控制流图中到达的节点与整个控制流图之间的可达性关系,并将其作为种子选择的优先级指标.此外,AIFORE[44] 和 BeDivFuzz[111] 通过分析种子输入格式,分别设计了基于格式的能量分配算法和基于语法的编变异测类,分配更多能量给那些变异较少格式对应的种子或让突变策略目标偏向于有效性和触发程序行为的多样性.

(2)选择性路径探索.除了追求路径探索的全面性和均匀性,一部分研究通过智能定向模糊测试技术帮助模糊测试工具在测试过程中更加聚焦于目标程序的关键路径和敏感区域的探索,即选择性路径探索,从而帮助提高漏洞挖掘效率和准确性. AFLGo[107]、Hawkeye[113] 等是代表性的定向模糊测试方法,通过调整能量分配和变异策略,集中测试接近目标的种子.这些方法通过考虑程序的调用图和控制流信息来优化种子选择,从而提高了测试的目标性和效率.此外,研究人员提出了智能化的路径修剪算法,减少模糊测试工具在无法到达目标的路径上的探索,提高测试效率.如 FuzzGuard[114] 利用机器学习帮助过滤掉不可达目标的探索路径,Beacon[115] 设计了轻量级的静态分析算法来找出程序中的不可达分支并进行修剪,而针对操作系统等大型软件系统,G-Fuzz[116] 进一步优化了定向模糊测试的距离计算,通过设计轻量级距离计算,有效提升了定向模糊测试应用在大规模系统软件上的性能表现.如图 9 所示,最新研究 FishFuzz[112] 考虑了多目标的定向模糊测试,更符合实际情况中程序存在多个漏洞的情况. FishFuzz 设计了一种多目标的距离度量作为反馈指标,优先选择没有被足够探索且与潜在目标位置更近的种子,并给他们分配更多的能量.

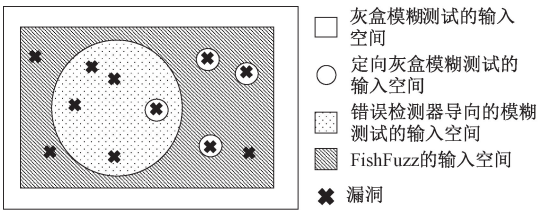

图 9　FishFuzz 和主流导向模糊测试的目标区别[112]

综上所述,为了提高模糊测试的路径探索效率,研究人员主要从历史路径探索情况、漏洞触发条件、程序控制流图、输入格式有效性以及潜在的漏洞位置角度出发引导模糊测试的种子选择、能量分配及变异策略,来实现路径探索的全面性和均匀性或实现关键路径的重点探索.然而,漏洞在程序内部的分

布是离散且稀疏的，面对大型操作系统和具有复杂嵌套格式的输入，如何在保持全面性的同时减少测试成本，仍然是一个挑战. 一个可行方案是有选择地对可能漏洞位置进行测试或通过识别出更有价值的多条路径来进行多目标的导向模糊测试. 在多目标导向模糊测试过程中，如何利用目标之间的关系以及如何平衡不同目标的探索仍值得进一步探索. 此外，针对一些特殊的对象，如物联网固件和可信执行环境等闭源的测试对象，如何设计高效的路径探索策略仍是待解决的问题.

5.3 自适应优化策略

自适应优化策略是指根据执行过程中的反馈信息自适应地切换不同的种子调度和变异策略，以提高模糊测试效率. 研究人员利用诸如多臂赌博机[117]等技术来辅助模糊测试工具自适应地选择和调整反馈机制、种子调度和变异策略.

(1)针对反馈机制. Cerebro[118]利用多目标优化算法来评价种子的质量和潜力，进一步动态调整种子选择和种子能量分配. 其中属性信息涉及的目标包括种子文件大小、种子执行时间，以及执行序列的特性(路径覆盖率、是否提升代码覆盖率和被覆盖的代码的复杂度).

(2)针对种子调度. 研究人员应用强化学习技术来动态自适应地调整反馈指标、种子属性和选择策略. 例如 EcoFuzz[120]、AFL-HIER[119]、SLIME[121]和 Syz-Vegas[122]. Ecofuzz[120]提出了对抗性多臂赌博机的变体对控制流图建模，测试过程中会优先选择奖励概率高的种子. 如果种子发现了新的路径会赋予一定的奖励概率，并设计了自转移的概率估计方法来更新种子的奖励概率. 如图 10 所示，AFL-HIER[119]通过建立多级种子树，并利用多臂赌博机逻辑，动态管理种子库以平衡探索与利用，显著提高了种子的管理效率和测试深度. SLIME[121]则提出了一个程序敏感的能量分配方案，针对不同程序的自适应地为具有不同属性的种子分配能量. 属性队列选择可以看作一个多臂赌博机问题，属性自适应能量分配算法的目标是估计每个属性队列发现有价值种子的潜力，并根据潜力选择属性队列，然后根据不同属性的性能表现选择种子.

(3)针对变异策略. 现有研究工作主要为了解决不同目标的冲突关系以及不同程序对变异策略适配的问题. MobFuzz[97]考虑了多个目标之间的冲突关系，以及适合多目标协同引导的能量调度策略. MOpt[23]则对模糊测试过程中变异操作的选择设计了自适应优化的算法. 具体来说，MOpt 使用一个定制化的粒子群优化算法，在测试过程中自适应地执行不同的变异操作，帮助模糊测试工具减少花费在无效变异操作上的时间. 此外，Wu 等人[123]通过实证研究提出了对变异策略 havoc 的优化方法，并提出了一个多臂赌博机模型来自适应调整不同的变异策略，提高变异效率.

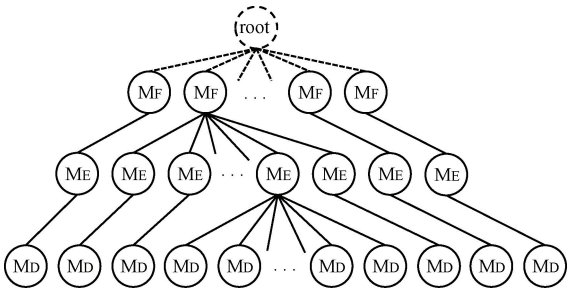

图 10 AFL-HIER 的多级种子树[119]

针对自适应优化策略，FuzzFactory[124]、FuzzBench[125]和 UniFuzz[126]提出了更为灵活和可扩展的测试框架，支持在多个领域内部署定制的模糊测试策略. 这些框架的设计考虑了多领域的需求，使得模糊测试技术不仅局限于特定类型的软件系统. 但目前数据集主要是小型通用应用程序，在对复杂程序如数据库、操作系统内核、物联网设备等的测试和评估上仍有一定局限性.

5.4 复杂程序空间探索

复杂程序空间探索是对所有程序路径探索优化方法的补充，通过对目标程序空间多角度的建模，添加新的反馈指标如数据流、状态等来指导种子调度和变异，避免因单纯探索程序的控制流图导致特殊漏洞检测效率低下. 为此，研究人员引入了包括漏洞特征、种子信息量、数据流和程序状态等新的反馈机制和对应算法. 表 3 对比了用于模糊测试循环的经典技术及测试框架.

基于漏洞特征和种子信息量分析的模糊测试技术提高了反馈的粒度. Medicherla 等人[106]评估测试输入与潜在内存漏洞触发条件的距离来优化模糊测试. 而 Manes 等人[127]通过分析不同输入引发的程序分支组合差异来优化种子评估. Coppik 等人[128]将内存操作纳入考量，保留触发新内存读写地址的种子. 然而，这些基于人工定义的程序特征的方法面临可扩展性问题. 更进一步，Böhme 等人[129]和 Wang 等人[130]则分别从种子信息量和漏洞潜在序列角度出发来指导模糊测试. Böhme 等人[129]提出了自适应的基于熵的智能种子能量分配策略 Entropic.

表 3　用于模糊测试循环优化的经典技术及测试框架

名称	依赖框架	类别	测试目标	优化类别	反馈指标	关键技术	优化效果	不足之处
AFLFast[108]	AFL	M	通用程序	P	分支覆盖率	优先选择执行低频路径且被选择次数较少的种子,并分配更多能量	发现漏洞的速度相比 AFL 提升 7 倍	针对路径及触发频率的计算开销较大
AFLGo[107]	AFL	M	通用程序	P	与目标位之间的距离	基于模拟退火算法的功率调度,随时间变化将更多能量分配给更靠近目标的种子	支持补丁和漏洞检测、持续性模糊测试、漏洞复现等	不支持闭源程序
FairFuzz[109]	AFL	M	通用程序	P	分支覆盖率、触发稀有路径分支触发情况	(1) 优先选择执行低频路径的种子; (2) 变异时保留触发稀有路径分支的字节,来增加对低频路径的探索	覆盖率增长速度优于 AFl;在具有嵌套条件结构的程序上能实现持续的覆盖率增长	模糊测试过程中没有命中的分支依然不会被探索
MOpt[23]	AFL	M	通用程序	PA	分支覆盖率	基于粒子群的智能优化算法自适应地选择变异操作	相比 AFL 多发现了 170% 的安全漏洞	粒子群全局最优算法开销较大,会使得测试速度降低
Entropic[129]	LibFuzzer	M	通用程序	PS	分支覆盖率和信息熵	优先选择具有大信息熵的种子并给其分配更多能量	覆盖率相比 LibFuzzer 有提升	仅考虑种子的信息熵
SyzVegas[122]	Syzkaller	MG	系统内核	PA	分支覆盖率	通过定制化的多臂赌博机智能算法来自适应调整种子选择和能量分配过程	覆盖率相比 Syzkaller 提升了 38.7%	偏向执行速度较快的系统调用,难以快速精简长系统调用序列输入
FuzzUSB[133]	Syzkaller	M G	系统驱动	PS	分支覆盖率、状态覆盖率	根据领域专有知识设计了四种状态变异操作来探索 USB 协议栈的深层状态空间	相比 Syzkaller,覆盖率提升了 3 倍	在真实设备上测试速度较慢,状态识别受限于静态分析的准确率,不支持闭源驱动
StateFuzz[105]	Syzkaller	M G	系统驱动	PS	分支覆盖率、程序状态	保存产生新覆盖率或新的状态的种子,优先选择那些触发低频状态的种子	相比 Syzkaller 代码覆盖率提升了 19%	不支持闭源驱动,状态的完整性受限于静态分析的准确率
EMS[78]	AFL	M	通用程序	PA	分支覆盖率	设计了一个可能性字节定向模型,复用历史数据中的高效变异操作及变异字节位置来优化变异策略	相比 AFL 发现的漏洞数量提升了 4.91 倍	变异算子及位置的推断粒度较粗
K-Scheduler[104]	AFL/LibFuzzer	M	通用程序	P	分支覆盖率和图中心性	评估当前种子在控制流图中到达的节点与整个控制流图的可达性关系,优先选择可以可达性更高的种子	在 FuzzBench 上覆盖率平均提升了 4.21%	不支持闭源程序,中心线分析受限于静态分析准确率

注:上述模糊测试技术均为灰盒模糊测试,M=基于变异,G=基于生成,P=高效路径探索,A=自适应优化策略,S=复杂状态空间探索.

该策略用熵来衡量一个输入种子对程序行为的影响,然后把更多的能量分配给能够影响程序行为的种子输入. Wang 等人[130]将操作序列覆盖率作为反馈来引导模糊测试工具逐步地生成覆盖这些操作序列的测试样例. 此外,他们部署了数据流分析来推断测试输入是如何影响程序状态的,并设计了有效的变异策略来防止不必要的变异.

此外,基于数据流分析的智能模糊测试技术帮助捕获到控制流无法观察到的程序行为,然而,以数据流覆盖率为引导的模糊器常常与重量级的程序分析工具(如污点分析和符号执行等)相结合,导致显著的运行时间开销,从而降低模糊器的处理效率. 近

期,研究人员提出了 dataFlow[131],采用轻量级数据流分析引导,通过对关键变量的定义和使用代码处进行插桩,使得模糊测试在测试过程中可以快速得到数据流覆盖率.其使用的数据流覆盖率较为粗粒度,仅考虑数据创建和使用.

最后,程序状态的理解对模糊测试至关重要,涉及程序在执行中的内部状态和外部环境的组合,如变量值、堆栈状态、函数调用顺序及交互输入输出.通过深入探索程序状态空间,智能模糊测试可以更有效地发现深层次漏洞.例如,Aschermann 提出了一种人机交互的智能模糊测试技术 IJON[132],将人对程序状态知识的理解反馈到模糊测试中,探索更多状态空间. StateFuzz[105] 观察到某些程序通过状态变量控制程序状态,针对 Linux 驱动程序提出了自动化状态变量识别及种子选择技术.该技术首先通过静态分析识别和评估状态变量的取值范围,然后对程序系统进行插桩以追踪状态变量的变化,并保存触发新状态的测试样例作为种子,为探索较少被测试的状态分配更多能量. FuzzUSB[133] 利用 USB 协议的消息接口作为状态节点,通过符号执行来识别这些节点的状态约束,从而指导更有效的状态探索.此外,GREBE[134] 和 ACTOR[135] 研究也为 Linux 内核状态识别提供了新方法,例如通过函数接口中的结构体和数值收集状态信息,以及识别可能触发漏洞的行为作为测试的反馈指标.

综上所述,研究者们已从信息熵、数据流和程序状态提出了比传统覆盖率更精细的反馈指标,提高了漏洞检测效率,它们在实施中仍面临着计算开销大和对测试盲目等挑战.特别是对于具有复杂状态的程序,如何有效理解和收集程序状态信息,并将这些信息用于指导模糊测试,仍然需要进一步研究.

6 测试预言

测试预言决定了模糊测试技术检测的漏洞类型和范围.研究人员主要围绕特定错误检测器、差分测试等角度加强测试预言的智能化.

(1)错误检测器.通过匹配程序行为和错误特征实现错误的捕获.因此,错误检测器发现的错误通常有着明确的错误范式.被广泛采用的 ASAN[136]、UBSAN[137]、KASAN[138] 等错误检测器通过插桩目标程序,在执行期间对内存访问进行检测和分析,能够捕获缓冲区溢出、空指针解引用、释放后使用等内存错误.但这类错误检测器主要支持 C/C++开源程序,且往往会引入额外的性能开销,后续工作致力于改善其效率与可用性. Jeon 等人[139] 为错误检测器提供了更加轻量化的内存管理元数据结构,并根据运行状况动态切换数据结构,提高了检测器的效率. Zhang 等人[140] 指出检查器可能会插入过多的检查指令而增加额外开销,可以通过消除其中不必要的检查指令改善其性能. Ba 等人[141] 提出了一种专门对基于 fork 的模糊测试进行优化的错误检测器,使用轻量化的随机标记而非传统元数据来标记对象边界,以此最小化程序启停的开销. Chen 等人[142] 和 Cho 等人[143] 拓展了错误检测器在闭源场景下的使用,引入硬件特性降低运行开销.除了内存检测器之外,Atlidakis 等人[144] 设计了一系列面向云服务 API 安全属性的错误检测器,通过分析服务请求的序列以识别信息泄露、越权访问等潜在漏洞. Chen 等人[145] 发现神经元激活和反向传播梯度偏离有效值可能与深度神经网络可执行文件的不安全动作相关,据此设计了检测器识别这一越界行为.上述检测器均依赖研究人员实现定义好的错误模式或规约.已有研究工作指出这种错误模式和规约的定义存在误报和漏报问题[146-148],难以覆盖所有的错误情况或正确情况.

(2)差分测试.则通过比较不同实现或版本的相同功能系统的输出来寻找潜在错误,特别是在缺少明确错误范式的场景如编译器测试中尤为有效.这种方法能够揭示一致性错误和语义错误,支持扩展到多个领域如深度学习库、网络协议[149-150]、区块链交易[151]、机器人操作系统[152] 的测试. Ye 等人[49] 选择不同厂商的 JS 引擎作为测试实例,输入相同脚本交叉验证不同实例的输出,检测其中存在的一致性错误. Bernhard 等人[153] 考虑编译优化前后程序应当保持相同的语义,将启用即时优化与未启用优化的 JS 引擎版本作为参照对象,比较相同脚本在两种情况下的执行结果,从而发现即使优化过程中引入的优化错误.如图 11 所示,LEMON[154] 将生成的神经网络模型输入到四个主流深度学习库,揭示了不同深度学习库之间存在的不一致性.然而,上述差分测试仍存在误报问题.如 Zhong 等人[155-156] 指出不同编译程序会导致不同的程序运行结果,但不应该被列为是漏洞.

综上所述,错误检测器和差分测试成功地帮助研究人员捕捉到数以千计的漏洞.然而,这些测试预言仍面临可扩展性、误报和测试盲点等问题.就基于

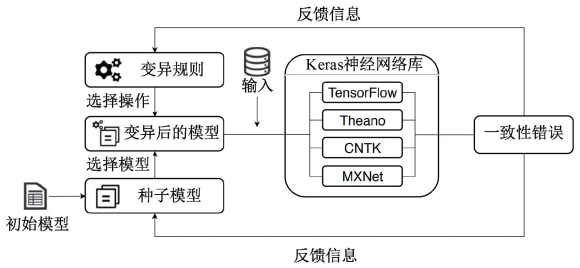

图 11　LEMON 采用差分测试捕获一致性漏洞[154]

错误检测器而言,现有工作大多致力于提升检测效率,对检测的误报和漏报关注较少,特别是内核测试过程中的误报.此外,差分测试固然能够弥补漏洞范式缺少的不足,但当缺乏类似的实例时这一方法可能无法奏效.如何在缺少参照的情况下检测语义、逻辑漏洞仍然是个挑战.针对新兴的场景和对象,如物联网设备、云系统和人工智能系统等复杂对象,如何挖掘潜在的规约和定义潜在的错误模式仍是一个开放问题.进一步地,考虑到系统的复杂性,如何应用上述规约和漏洞模式来设计相应的错误检测器也亟需进一步的研究.

7　未来研究方向

基于上述研究,如表 4 所示,本文进一步探讨了智能模糊测试面临的挑战及未来研究方向:

表 4　智能模糊测试领域的现实挑战及未来研究方向

现实挑战	未来研究方向	研究方向内涵
1.针对复杂测试测试生成,存在测试样例语法语义分析不足、测试样例多样化程度不高、嵌套约束求解难等挑战.	研究结合新型人工智能方法和程序分析技术,实现对测试样例空间的有效理解和高质量测试样例生成.	1.基于大语言模型的测试样例生成方法研究; 2.基于人工智能方法的程序约束求解优化方法研究.
2.针对模糊测试循环优化,存在对多样化目标软件系统分析难、对软件系统复杂程序空间探索不足等挑战.	研究针对不同目标软件系统的定制智能模糊测试技术,尤其是新型软件系统;研究智能化程序分析方法,提高智能模糊测试对程序信息的有效分析提取和程序状态高效探索的能力;研究人工智能和并行处理引导的模糊测试技术,优化模糊测试的执行过程,提高测试速率和吞吐量.	1.针对静态分析、符号执行技术在模糊测试场景下的精确度、效率、可扩展性等方面的优化研究; 2.结合神经网络等人工智能技术,实现对测试过程优化研究; 3.结合多种分析方法研究目标程序空间数据流和状态空间建模并设计轻量级的探索算法; 4.结合多目标优化、动态反馈信息归并等智能技术实现高效的并行模糊测试.
3.针对测试预言,存在检测漏洞类型单一、检测开销大等挑战.	研究针对不同漏洞类型的漏洞预言和模糊测试方法;研究轻量级的多类型漏洞检测方法.	1.针对语义漏洞、逻辑漏洞等新型漏洞的模糊测试方法研究; 2.轻量级模糊测试运行过程异常行为方法研究.

(1)复杂测试样例的生成仍是一个关键挑战,特别是在闭源、庞大代码体量及复杂依赖关系的软件系统中.当前智能模糊测试技术在测试样例的语法语义分析、多样化程度及嵌套约束求解方面仍存在一定局限性.因此,未来的关键研究方向包括运用高效的智能化技术如大语言模型和符号执行增强模糊测试的语义理解,生成更符合程序语义需求的高质量测试样例,同时降低处理复杂数据流和路径约束的开销.例如,利用大语言模型生成测试样例可以有效提高测试样例的质量,并在一定程度上避免了程序分析的开销.

(2)现有智能技术在提升模糊测试循环的效率仍面临多重挑战,尤其是在处理大型复杂软件系统时,存在对多样化目标软件系统分析难、对程序信息利用不充分不智能、程序执行过程效率低下等挑战.针对上述挑战,亟需针对特定目标软件系统定制智能模糊测试技术,尤其是那些新型软件系统,如智能合约和物联网固件等,以解决传统技术可能无法涵盖的复杂和独特问题;研究智能化程序分析方法,提高智能模糊测试对程序信息的有效分析提取和程序状态高效探索能力;研究结合人工智能和并行处理技术,优化模糊测试的执行过程,提高测试速率和吞吐量.具体策略包括深入研究静态分析和符号执行在模糊测试中的应用,提升其精确度、效率和可扩展性;结合神经网络等人工智能技术,实现对测试过程优化研究;探索融合多种分析方法的数据流和状态空间建模技术.此外,针对大型系统测试的效率问题,探索分而治之的策略,模块化程序分解及智能算法合并反馈.同时,研究硬件辅助和插桩技术的系统评估和优化,是提升覆盖率收集效率和精度的关键.最后,开发多目标导向的模糊测试策略,平衡不同测试目标的探索,以及为特殊测试对象如物联网固件设计有效的路径探索策略,将是推动模糊测试向更高智能化和效率化发展的重要方向.

(3)模糊测试通过监测程序执行过程中是否触发异常行为来发现程序中的漏洞.然而,在目前模糊测试错误检测优化方面,仍存在检测漏洞类型单一、检测开销大等挑战.模糊测试通常以程序崩溃作为一种常见的错误行为,或者对程序进行一些特殊的插桩(例如 Sanitizer 等)来发现一些内存漏洞等.因此,模糊测试通常只能发现一些与内存相关的漏洞类型,并且在使用 Sanitizer 等进行模糊测试时,会造成一定的运行开销.针对上述问题,一方面需要研究针对不同类型漏洞的检测范式和模糊测试方法,例如逻辑漏洞和条件竞争等漏洞类型.另一方面,需要研究轻量级的多类型漏洞检测方法,即研究如何在较低的程序运行开销下,实现对多种类型漏洞的有效检测.

8 结束语

随着近年来软件系统规模和复杂性的增加,安全漏洞的数量不断上升,其影响范围也逐步扩大.一大批来自学术界和学业界的学者对使用智能技术提升模糊测试性能的问题进行了广泛关注和深入研究,并且在模糊测试的复杂测试样例的生成、模糊测试循环的效率提升以及测试预言机三个关键优化问题上取得了许多瞩目的研究成果.然而,在新兴的时代背景下,现有智能模糊测试仍面临着许多关键的科学问题和挑战,包括物联网场景下的封闭测试环境和高度依赖的组件限制、针对复杂语法和嵌套约束的输入生成、对海量代码和异构架构的软件系统的性能瓶颈、复杂程序空间的深度探索以及高效的漏洞检测范式构建等.

参 考 文 献

[1] Lanzi A, Martignoni L, Monga M, Paleari R. A smart fuzzer for x86 executables//Proceedings of the 3rd International Workshop on Software Engineering for Secure Systems. Minnesota, USA, 2007: 7-7

[2] Luo L., Zeng Q., Yang B., Zuo F., Wang J. Westworld: Fuzzing-assisted remote dynamic symbolic execution of smart apps on iot cloud platforms//Proceedings of the 37th Annual Computer Security Applications Conference. Austin, USA, 2021: 982-995

[3] Pham VT, Böhme M, Santosa AE, Căciulescu AR, Roychoudhury A. Smart greybox fuzzing. IEEE Transactions on Software Engineering. 2019 16;47(9):1980-97

[4] Ren Zezhong, Zheng Han, Zhang Jiayuan, Wang Wenjie, Feng Tao, Wang He, Zhang Yuqing. A review of fuzzing techniques. Journal of Computer Research and Development, 2021, 58(5): 944-963(in Chinese)
(任泽众, 郑晗, 张嘉元, 王文杰, 冯涛, 王鹤, 张玉清. 模糊测试技术综述. 计算机研究与发展, 2021, 58(5): 944-963)

[5] Wang P, Zhou X, Lu K, Yue T, Liu Y. Sok: The progress, challenges, and perspectives of directed greybox fuzzing. arXiv preprint, 2020, arXiv:2005.11907

[6] Manès VJ, Han H, Han C, Cha SK, Egele M, Schwartz EJ, Woo M. The art, science, and engineering of fuzzing: A survey. IEEE Transactions on Software Engineering. 2019, 47(11): 2312-2331

[7] Wang Y, Jia P, Liu L, Liu J. A systematic review of fuzzing based on machine learning techniques. PIoS one, 2020, 15(8): e0237749

[8] Boehme M, Cadar C, Roychoudhury A. Fuzzing: Challenges and reflections. IEEE Software, 2021, 38(3): 79-86

[9] Zhu X, Wen S, Camtepe S, Xiang Y. Fuzzing: A survey for roadmap. ACM Computing Surveys, 2022, 54(11s): 1-36

[10] Yun J, Rustamov F, Kim J, Shin Y. Fuzzing of embedded systems: A survey. ACM Computing Surveys, 2022, 55(7): 137:1-137:33

[11] Li J, Zhao B, Zhang C. Fuzzing: A survey. Cybersecurity, 2018, 1(1): 6

[12] Serebryany K. OSS-Fuzz-Google's continuous fuzzing service for open source software. Vancouver, Canada: USENIX Association, 2017

[13] Miller BP, Fredriksen L, So B. An empirical study of the reliability of unix utilities. Communications of the ACM, 1990, 33(12): 32-44

[14] Kaksonen R. A functional method for assessing protocol[Licentiate Thesis]. Aalto University, Finland. 2001

[15] Aitel, D. An introduction to spike, the fuzzer creation kit//Proceedings of the Black Hat USA. Las Vegas, USA, 2002

[16] Sutton M, Greene A. The art of file format fuzzing//Proceedings of the Black Hat USA. Las Vegas, USA, 2005

[17] Google. Peach fuzzer: effective fuzzing. Las Vegas: Google. 2013

[18] Mozilla Fuzzing Security. Funfuzz: A collection of fuzzers in a harness for testing the spidermonkey javascript engine. USA: Mozilla Fuzzing Security. 2008

[19] Zalewski M. American fuzzy lop. USA: Google. 2018

[20] Vyukov D, Konovalov A. Syzkaller: An unsupervised coverage-guided kernel fuzzer. USA: Google. 2015

[21] Google. ClusterFuzz: Scalable fuzzing infrastructure. USA: Google. 2019

[22] Gan S, Zhang C, Qin X, Tu X, Li K, Pei Z, Chen Z. CollAFL: Path sensitive fuzzing//Proceeding of the 2018 IEEE Symposium on Security and Privacy. San Francisco, USA, 2018: 679-696

[23] Lyu C, Ji S, Zhang C, Li Y, Lee W-H, Song Y, Beyah R. MOPT: Optimized mutation scheduling for fuzzers//Pro-

ceeding of the 28th USENIX Security Symposium. Santa Clara, USA, 2019: 1949-1966

[24] Deng Y, Xia CS, Peng H, Yang C, Zhang L. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models//Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. Seattle, USA, 2023: 423-435

[25] Liu D, Metzman J, Chang O. AI-powered fuzzing: Breaking the bug hunting barrier. USA. 2023

[26] Zhang C, Bai M, Zheng Y, Li Y, Xie X, Li Y, Ma W, Sun L, Liu Y. Understanding large language model based fuzz driver generation. arXiv preprint arXiv:2307.12469, 2023

[27] Yang C, Zhao Z, Zhang L. KernelGPT: Enhanced kernel fuzzing via large language models. arXiv preprint arXiv: 2401.00563, 2023

[28] Xia CS, Paltenghi M, Tian JL, Pradel M, Zhang L. Universal fuzzing via large language models. arXiv preprint arXiv: 2308.04748, 2023

[29] Kitchenham B, Charters S. Guidelines for performing systematic literature reviews in software engineering. Technical Report: EBSE 2007-001, Keele University and Durham University, UK, 2007

[30] Liu B, Zhang C, Gong G, Zeng Y, Ruan H, Zhuge J. FANS: Fuzzing android native system services via automated interface analysis//Proceeding of the 29th USENIX Security Symposium. Boston, USA, 2020: 307-323

[31] Corina J, Machiry A, Salls C, Shoshitaishvili Y, Hao S, Kruegel C, Vigna G. DIFUZE: Interface aware fuzzing for kernel drivers//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, USA, 2017: 2123-2138

[32] Sun H, Shen Y, Liu J, Xu Y, Jiang Y. KSG: Augmenting kernel fuzzing with system call specification generation//Proceeding of the 2022 USENIX Annual Technical Conference. 2022: 351-366

[33] Chen W, Wang Y, Zhang Z, Qian Z. SyzGen: Automated generation of syscall specification of closed-source macos drivers//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. Republic of Korea, 2021: 749-763

[34] Wang J, Chen B, Wei L, Liu Y. Superion: Grammar-aware greybox fuzzing//2019 IEEE/ACM 41st International Conference on Software Engineering. Montreal, Canada, 2019: 724-735

[35] Park S, Xu W, Yun I, Jang D, Kim T. Fuzzing javascript engines with aspect-preserving mutation//Proceeding of the 2020 IEEE Symposium on Security and Privacy. San Francisco, USA, 2020: 1629-1642

[36] Hao Y, Li G, Zou X, Chen W, Zhu S, Qian Z, Sani AA. SyzDescribe: Principled, automated, static generation of syscall descriptions for kernel drivers//Proceeding of the 2023 IEEE Symposium on Security and Privacy. San Francisco, USA, 2023: 3262-3278

[37] Mu T, Zhang H, Wang J, Li H. CoLaFUZE: Coverage-guided and layout-aware fuzzing for android drivers. IEICE Transactions on Information and Systems, The Institute of Electronics, Information and Communication Engineers, 2021, E104-D(11): 1902-1912

[38] Chen J, Diao W, Zhao Q, Zuo C, Lin Z, Wang X, Lau WC, Sun M, Yang R, Zhang K. IoTFuzzer: Discovering memory corruptions in iot through app-based fuzzing//Proceedings of the 2018 Network and Distributed System Security Symposium. San Diego, USA, 2018

[39] Redini N, Continella A, Das D, De Pasquale G, Spahn N, Machiry A, Bianchi A, Kruegel C, Vigna G. Diane: Identifying fuzzing triggers in apps to generate under-constrained inputs for iot devices//Proceeding of the 2021 IEEE Symposium on Security and Privacy. San Francisco, USA, 2021: 484-500

[40] Wang J, Chen B, Wei L, Liu Y. Skyfire: Data-driven seed generation for fuzzing//Proceeding of the 2017 IEEE Symposium on Security and Privacy. San Jose, USA, 2017: 579-594

[41] Feng X, Sun R, Zhu X, Xue M, Wen S, Liu D, Nepal S, Xiang Y. Snipuzz: Black-box fuzzing of IoT firmware via message snippet inference//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. Korea, 2021: 337-350

[42] Lyu C, Ji S, Li Y, Zhou J, Chen J, Chen J. Smartseed: Smart seed generation for efficient fuzzing. arXiv preprint arXiv:1807.02606, 2018

[43] Godefroid P, Peleg H, Singh R. Learn&Fuzz: Machine learning for input fuzzing. arXiv preprint arXiv:1701.0732, 2017

[44] Shi J, Wang Z, Feng Z, Lan Y, Qin S, You W, Zou W, Payer M, Zhang C. AIFORE: Smart fuzzing based on automatic input format reverse engineering//Proceedings of the 32nd USENIX Security Symposium. Anaheim, USA, 2023: 4967-4984

[45] Hu Z, Shi J, Huang Y, Xiong J, Bu X. GANFuzz: A gan-based industrial network protocol fuzzing framework//Proceedings of the 15th ACM International Conference on Computing Frontiers. 2018: 138-145

[46] Choi J, Kim K, Lee D, Cha SK. NtFuzz: Enabling type-aware kernel fuzzing on windows with static binary analysis//Proceedings of the 2021 IEEE Symposium on Security and Privacy. San Francisco, USA, 2021: 677-693

[47] Wang D, Li Y, Zhang Z, Chen K. CarpetFuzz: Automatic program option constraint extraction from documentation for fuzzing//Proceedings of the 32nd USENIX Security Symposium. Anaheim, USA, 2023: 1919-1936

[48] Xie D, Li Y, Kim M, Pham HV, Tan L, Zhang X, Godfrey MW. DocTer: Documentation-guided fuzzing for testing deep learning api functions//Proceedings of the 31st ACM SIG-

SOFT International Symposium on Software Testing and Analysis. Korea, 2022: 176-188

[49] Ye G, Tang Z, Tan SH, Huang S, Fang D, Sun X, Bian L, Wang H, Wang Z. Automated conformance testing for javascript engines via deep compiler fuzzing//Proceedings of the 42nd ACM SIGPLAN Conference on Programming Language Design and Implementation. Virtual, Canada, 2021: 435-450

[50] Deng Y, Xia CS, Yang C, Zhang SD, Yang S, Zhang L. Large language models are edge-case fuzzers: Testing deep learning libraries via fuzzgpt. arXiv preprint arXiv:2304.02014, 2023

[51] Min B, Ross H, Sulem E, Veyseh AP, Nguyen TH, Sainz O, Agirre E, Heintz I, Roth D. Recent advances in natural language processing via large pre-trained language models: A survey. ACM Computing Surveys. 2023 Sep 14;56(2):1-40

[52] Groß S, Koch S, Bernhard L, Holz T, Johns M. FUZZILLI: Fuzzing for javascript jit compiler vulnerabilities//Proceedings 2023 Network and Distributed System Security Symposium. San Diego, USA, 2023

[53] Zhong R, Chen Y, Hu H, Zhang H, Lee W, Wu D. SQUIRREL: Testing database management systems with language validity and coverage feedback//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. USA, 2020: 955-970

[54] Xu W, Park S, Kim T. FREEDOM: Engineering a state-of-the-art dom fuzzer//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. USA, 2020: 971-986

[55] Liang Y, Liu S, Hu H. Detecting logical bugs of dbms with coverage-based guidance//Proceedings of the 31st USENIX Security Symposium. Boston, USA,2022: 4309-4326

[56] Chen Y, Zhong R, Hu H, Zhang H, Yang Y, Wu D, Lee W. One engine to fuzz 'em all: Generic language processor testing with semantic validation//Proceedings of the 2021 IEEE Symposium on Security and Privacy. San Francisco, USA, 2021: 642-658

[57] Han H, Oh D, Cha SK. CodeAlchemist: Semantics-aware code generation to find vulnerabilities in javascript engines//Proceedings of the 2019 Network and Distributed System Security Symposium. San Diego, USA, 2019

[58] Holler C, Herzig K, Zeller A. Fuzzing with code fragments//Proceedings of the 21st USENIX Security Symposium. Bellevue, USA, 2012: 445-458

[59] Lee S, Han H, Cha SK, Son S. Montage: A neural network language model-guided javascript engine fuzzer//Proceedings of the 29th USENIX Security Symposium. Boston, USA, 2020: 2613-2630

[60] Veggalam S, Rawat S, Haller I, Bos H. IFuzzer: An evolutionary interpreter fuzzer using genetic programming//Proceedings of the 21st European Symposium on Research in Computer Security. Heraklion, Greece, 2016, 9878: 581-601

[61] Mansur MN, Christakis M, Wüstholz V, Zhang F. Detecting critical bugs in smt solvers using blackbox mutational fuzzing//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. USA, 2020: 701-712

[62] Rawat S, Jain V, Kumar A, Cojocar L, Giuffrida C, Bos H. VUzzer: Application-aware evolutionary fuzzing//Proceedings of the 2017 Network and Distributed System Security Symposium. San Diego, USA, 2017

[63] Stephens N, Grosen J, Salls C, Dutcher A, Wang R, Corbetta J, Shoshitaishvili Y, Kruegel C, Vigna G. Driller: Augmenting fuzzing through selective symbolic execution//Proceedings of the 2016 Network and Distributed System Security Symposium. San Diego, USA, 2016

[64] 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//Proceedings of the IEEE symposium on security and privacy. San Francisco, USA, 2016:138-157

[65] Cho M, Kim S, Kwon T. Intriguer: Field-level constraint solving for hybrid fuzzing//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. London, UK, 2019: 515-530

[66] Yun I, Lee S, Xu M, Jang Y, Kim T. QSYM: A practical concolic execution engine tailored for hybrid fuzzing//Proceedings of the 27th USENIX Security Symposium. Baltimore, USA, 2018: 745-761

[67] Chen J, Wang J, Song C, Yin H. JIGSAW: Efficient and scalable path constraints fuzzing//Proceedings of the 2022 IEEE Symposium on Security and Privacy. San Francisco, USA, 2022: 18-35

[68] Liew D, Cadar C, Donaldson AF, Stinnett JR. Just fuzz it: Solving floating-point constraints using coverage-guided fuzzing//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Tallinn, Estonia, 2019: 521-532

[69] Huang H, Yao P, Wu R, Shi Q, Zhang C. Pangolin: Incremental hybrid fuzzing with polyhedral path abstraction//Proceedings of the 2020 IEEE Symposium on Security and Privacy. San Francisco, USA, 2020: 1613-1627

[70] Zhao L, Duan Y, Yin H, Xuan J. Send hardest problems my way: Probabilistic path prioritization for hybrid fuzzing//Proceedings of the 2019 Network and Distributed System Security Symposium. San Diego, USA, 2019

[71] Chen Y, Ahmadi M, Wang B, Lu L. MEUZZ: Smart seed scheduling for hybrid fuzzing// Proceedings of the 23rd International Symposium on Research in Attacks, Intrusions and Defenses. San Sebastian, Spain, 2020: 77-92

[72] Aschermann C, Schumilo S, Blazytko T, Gawlik R, Holz T. REDQUEEN: Fuzzing with input-to-state correspondence//

Proceedings of the 2019 Network and Distributed System Security Symposium. San Diego, USA, 2019

[73] Li Y, Chen B, Chandramohan M, Lin S-W, Liu Y, Tiu A. Steelix: Program-state based binary fuzzing//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Paderborn, Germany, 2017: 627-637

[74] Chen P, Liu J, Chen H. Matryoshka: Fuzzing deeply nested branches//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. London, UK, 2019: 499-513

[75] Gan S, Key S, Engineering M, Computing A, Symposium US, Gan S, Zhang C, Chen P, Zhao B, Qin X, Wu D, Chen Z. Greyone: Data flow sensitive fuzzing//Proceedings of the 29th USENIX Security Symposium. Boston, USA, 2020: 2577-2594

[76] Chen P, Chen H. Angora: Efficient fuzzing by principled search//Proceedings of the 2018 IEEE Symposium on Security and Privacy. San Francisco, USA, 2018: 711-725

[77] Peng H, Shoshitaishvili Y, Payer M. T-fuzz: Fuzzing by program transformation//Proceedings of the 2018 IEEE Symposium on Security and Privacy. San Francisco, USA, 2018: 697-710

[78] Lyu C, Ji S, Zhang X, Liang H, Zhao B, Lu K, Beyah R. EMS: History-driven mutation for coverage-based fuzzing//Proceedings of the 2022 Network and Distributed System Security Symposium. San Diego, USA, 2022

[79] Kim K, Jeong DR, Kim CH, Jang Y, Shin I, Lee B. HFL: Hybrid fuzzing on the linux kernel//Proceedings of the 2020 Network and Distributed System Security Symposium. San Diego, USA, 2020

[80] Han H, Cha SK. IMF: Inferred model-based fuzzer//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, USA, 2017: 2345-2358

[81] Pailoor S, Aday A, Jana S. MoonShine: Optimizing os fuzzer seed selection with trace distillation//Proceedings of the 27th USENIX Security Symposium. Baltimore, USA, 2018: 729-743

[82] Sun H, Shen Y, Wang C, Liu J, Jiang Y, Chen T, Cui A. HEALER: Relation learning guided kernel fuzzing//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. Germany, 2021: 344-358

[83] Zhang C, Lin X, Li Y, Xue Y, Xie J, Chen H, Ying X, Wang J, Liu Y. APICraft: Fuzz driver generation for closed-source sdk libraries//Proceedings of the 30th USENIX Security Symposium. Vancouver, Canada, 2021: 2811-2828

[84] Zhou C, Zhang Q, Wang M, Guo L, Liang J, Liu Z, Payer M, Jiang Y. Minerva: Browser api fuzzing with dynamic mod-ref analysis//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022: 1135-1147

[85] Peng H, Yao Z, Sani AA, Tian DJ, Payer M. GLeeFuzz: Fuzzing webgl through error message guided mutation//Proceedings of the 32th USENIX Security Symposium. Anaheim, USA, 2023: 1883-1899

[86] Liang J, Chen Y, Wu Z, Fu J, Wang M, Jiang Y, Huang X, Chen T, Wang J, Li J. Sequence-oriented dbms fuzzing//Proceedings of the 2023 IEEE 39th International Conference on Data Engineering. Anaheim, USA, 2023: 668-681

[87] Su J, Dai H-N, Zhao L, Zheng Z, Luo X. Effectively generating vulnerable transaction sequences in smart contracts with reinforcement learning-guided fuzzing//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. Rochester, USA, 2022: 1-12

[88] Xu J, Zhang X, Ji S, Tian Y, Zhao B, Wang Q, Cheng P, Chen J. MOCK: Optimizing kernel fuzzing mutation wit context-aware dependency//Proceedings of the 2024 Network and Distributed System Security Symposium. San Diego, USA, 2024

[89] Deng Y, Yang C, Wei A, Zhang L. Fuzzing deep-learning libraries via automated relational api inference//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022: 44-56

[90] Jiang J, Xu H, Zhou Y. RULF: Rust library fuzzing via api dependency graph traversal//Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering. Melbourne, Australia, 2021: 581-592

[91] Liu Y, Li Y, Deng G, Liu Y, Wan R, Wu R, Ji D, Xu S, Bao M. Morest: Model-based restful api testing with execution feedback//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 1406-1417

[92] Atlidakis V, Godefroid P, Polishchuk M. RESTler: Stateful rest api fuzzing//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineerin. Montreal, Canada, 2019: 748-758

[93] Ispoglou K, Austin D, Mohan V, Payer M. FuzzGen: Automatic fuzzer generation//Proceedings of the 29th USENIX Security Symposium. Boston, USA, 2020: 2271-2287

[94] Nagy S, Hicks M. Full-speed fuzzing: Reducing fuzzing overhead through coverage-guided tracing//Proceedings of the 2019 IEEE Symposium on Security and Privacy. San Francisco, USA, 2019: 787-802

[95] Zhou C, Wang M, Liang J, Liu Z, Jiang Y. Zeror: Speed up fuzzing with coverage-sensitive tracing and scheduling//Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. Australia, 2020: 858-870

[96] Wang M, Liang J, Zhou C, Jiang Y, Wang R, Sun C, Sun J. RIFF: Reduced instruction footprint for coverage-guided fuzzing//Proceedings of the 2021 USENIX Annual Technical Conference. 2021: 147-159

[97] Schumilo S, Aschermann C, Gawlik R, Schinzel S, Holz T.

KAFL: Hardware-assisted feedback fuzzing for os kernels//Proceedings of the 26th USENIX Security Symposium. Vancouver, Canada, 2017: 167-182

[98] Li W, Shi J, Li F, Lin J, Wang W, Guan L. MAFL: Non-intrusive feedback-driven fuzzing for microcontroller firmware//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 1-12

[99] Wang Q, Chang B, Ji S, Tian Y, Zhang X, Zhao B, Pan G, Lyu C, Payer M, Wang W, Beyah R. SyzTrust: State-aware fuzzing on trusted os designed for iot devices//Proceedings of the 2024 IEEE Symposium on Security and Privacy. San Francisco, USA, 2024

[100] Eisele M, Ebert D, Huth C, Zeller A. Fuzzing embedded systems using debug interfaces//Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis. Seattle, USA, 2023: 1031-1042

[101] Petsios T, Zhao J, Keromytis AD, Jana S. SlowFuzz: Automated domain-independent detection of algorithmic complexity vulnerabilities//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas Texas, USA, 2017: 2155-2168

[102] Lemieux C, Padhye R, Sen K, Song D. PerfFuzz: Automatically generating pathological inputs//Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. Amsterdam, Netherlands, 2018: 254-265

[103] Wen C, Wang H, Li Y, Qin S, Liu Y, Xu Z, Chen H, Xie X, Pu G, Liu T. MemLock: Memory usage guided fuzzing//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Seoul, Korea, 2020: 765-777

[104] She D, Shah A, Jana S. Effective seed scheduling for fuzzing with graph centrality analysis//Proceedings of the 2022 IEEE Symposium on Security and Privacy. San Francisco, USA, 2022: 2194-2211

[105] Zhao B, Li Z, Qin S, Ma Z, Yuan M, Zhu W, Tian Z, Zhang C. StateFuzz: System call-based state-aware linux driver fuzzing//Proceedings of the 31st USENIX Security Symposium. Boston, USA, 2022: 3273-3289

[106] Medicherla RK, Komondoor R, Roychoudhury A. Fitness guided vulnerability detection with greybox fuzzing//Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. Seoul, Korea, 2020: 513-520

[107] Böhme M, Pham V-T, Nguyen M-D, Roychoudhury A. Directed greybox fuzzing//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, USA, 2017: 2329-2344

[108] Böhme M, Pham V-T, Roychoudhury A. Coverage-based greybox fuzzing as markov chain//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Vienna, Austria, 2016: 1032-1043

[109] Lemieux C, Sen K. FairFuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage//Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. Montpellier, France, 2018: 475-485

[110] Wang Y, Jia X, Liu Y, Zeng K, Bao T, Wu D, Su P. Not all coverage measurements are equal: fuzzing by coverage accounting for input prioritization//Proceedings of the 2020 Network and Distributed System Security Symposium. San Diego, USA, 2020

[111] Nguyen HL, Grunske L. BeDivFuzz: integrating behavioral diversity into generator-based fuzzing//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 249-261

[112] Zheng H, Zhang J, Huang Y, Ren Z, Wang H, Cao C, Zhang Y, Toffalini F, Payer M. FISHFUZZ: Catch deeper bugs by throwing larger nets//Proceedings of the 32nd USENIX Security Symposium. Manheim, USA, 2023: 1343-1360

[113] Chen H, Xue Y, Li Y, Chen B, Xie X, Wu X, Liu Y. Hawkeye: Towards a desired directed grey-box fuzzer//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto, Canada, 2018: 2095-2108

[114] Zong P, Lv T, Wang D, Deng Z, Liang R, Chen K. Fuzzguard: Filtering out unreachable inputs in directed grey-box fuzzing through deep learning//Proceedings of the 29th USENIX Security Symposium. Boston, USA, 2020: 2255-2269

[115] Huang H, Guo Y, Shi Q, Yao P, Wu R, Zhang C. Beacon: Directed grey-box fuzzing with provable path pruning//Proceedings of the 2022 IEEE Symposium on Security and Privacy. San Francisco, USA, 2022: 36-50

[116] Li Y, Chen Y, Ji S, Zhang X, Yan G, Liu AX, Wu C, Pan Z, Lin P. G-fuzz: A directed fuzzing framework for gvisor. IEEE Transactions on Dependable and Secure Computing, 2023: 1-18

[117] Vermorel J, Mohri M. Multi-armed bandit algorithms and empirical evaluation//Proceedings of the 16th European Conference on Machine Learning, Porto, Portugal. 2005: 437-448

[118] Li Y, Xue Y, Chen H, Wu X, Zhang C, Xie X, Wang H, Liu Y. Cerebro: Context-aware adaptive fuzzing for effective vulnerability detection//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Tallinn, Estonia, 2019: 533-544

[119] Wang J, Song C, Yin H. Reinforcement learning-based hierarchical seed scheduling for greybox fuzzing//Proceedings of the 2021 Network and Distributed System Security Symposium. 2021

[120] Yue T, Wang P, Tang Y, Wang E, Yu B, Lu K, Zhou X.

EcoFuzz: Adaptive energy-saving greybox fuzzing as a variant of the adversarial multi-armed bandit//Proceedings of the 29th USENIX Security Symposium. Boston, USA, 2020: 2307-2324

[121] Lyu C, Liang H, Ji S, Zhang X, Zhao B, Han M, Li Y, Wang Z, Wang W, Beyah R. SLIME: Program-sensitive energy allocation for fuzzing//Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. Seoul, Republic of Korea, 2022: 365-377

[122] Wang D, Zhang Z, Zhang H, Qian Z, Krishnamurthy SV, Abu-Ghazaleh N. SyzVegas: Beating kernel fuzzing odds with reinforcement learning//Proceedings of the 30th USENIX Security Symposium. Vancouver, Canada, 2021: 2741-2758

[123] Wu M, Jiang L, Xiang J, Huang Y, Cui H, Zhang L, Zhang Y. One fuzzing strategy to rule them all//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 1634-1645

[124] Padhye R, Lemieux C, Sen K, Simon L, Vijayakumar H. FuzzFactory: Domain-specific fuzzing with waypoints//Proceedings of the ACM on Programming Languages, 2019, 3: 1-29

[125] Metzman J, Szekeres L, Simon LMR, Sprabery RT, Arya A. FuzzBench: An open fuzzer benchmarking platform and service//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, USA, 2021

[126] Li Y, Ji S, Chen Y, Liang S, Lee W-H, Chen Y, Lyu C, Wu C, Beyah R, Cheng P, Lu K, Wang T. UNIFUZZ: A holistic and pragmatic metrics-driven platform for evaluating fuzzers//Proceedings of the 30th USENIX Security Symposium. Vancouver, Canada, 2021: 2777-2794

[127] Manès VJM, Kim S, Cha SK. Ankou: Guiding grey-box fuzzing towards combinatorial difference//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Seoul, Republic of Korea, 2020: 1024-1036

[128] Coppik N, Schwahn O, Suri N. MemFuzz: Using memory accesses to guide fuzzing//Proceedings of the 2019 12th IEEE Conference on Software Testing, Validation and Verification. Xi'an, China, 2019: 48-58

[129] Böhme M, Manès VJM, Cha SK. Boosting fuzzer efficiency: An information theoretic perspective//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. USA, 2020: 678-689

[130] Wang H, Xie X, Li Y, Wen C, Li Y, Liu Y, Qin S, Chen H, Sui Y. Typestate-guided fuzzer for discovering use-after-free vulnerabilities//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Seoul, Republic of Korea, 2020: 999-1010

[131] Herrera A, Payer M, Hosking AL. DatAFLow: Toward a data-flow-guided fuzzer. ACM Transactions on Software Engineering and Methodology, New York, USA, 2022

[132] Aschermann C, Schumilo S, Abbasi A, Holz T. Ijon: exploring deep state spaces via fuzzing//Proceedings of the 2020 IEEE Symposium on Security and Privacy. San Francisco, USA, 2020: 1597-1612

[133] Kim K, Kim T, Warraich E, Lee B, Butler KRB, Bianchi A, Jing Tian D. FuzzUSB: Hybrid stateful fuzzing of usb gadget stacks//Proceedings of the 2022 IEEE Symposium on Security and Privacy. San Francisco, USA, 2022: 2212-2229

[134] Lin Z, Chen Y, Wu Y, Mu D, Yu C, Xing X, Li K. GREBE: Unveiling exploitation potential for linux kernel bugs//Proceedings of the 2022 IEEE Symposium on Security and Privacy. San Francisco, USA, 2022: 2078-2095

[135] Fleischer M, Das D, Bose P, Bai W, Lu K, Payer M, Kruegel C, Vigna G. ACTOR: Action-guided kernel fuzzing//Proceedings of the 32nd USENIX Security Symposium. Anaheim, USA, 2023: 5003-5020

[136] Serebryany K, Bruening D, Potapenko A, Vyukov D. AddressSanitizer: A fast address sanity checker//Proceedings of the USENIX annual technical conference. 2012: 309-318

[137] Hans Wennborg. Using Clang for fun and profit-Examples from the Chromium project. Aarhus,Denmark: Google, 2012.

[138] Song D, Lettner J, Rajasekaran P, Na Y, Volckaert S, Larsen P, Franz M. SoK: Sanitizing for security//Proceedings of the 2019 IEEE Symposium on Security and Privacy. San Francisco, USA, 2019:1275-1295

[139] Jeon Y, Han W, Burow N, Payer M. FuZZan: Efficient sanitizer metadata design for fuzzing//Proceedings of the 2020 USENIX Annual Technical Conference. 2020: 249-263

[140] Zhang J, Wang S, Rigger M, He P, Su Z. SANRAZOR: Reducing redundant sanitizer checks in c/c++ programs//Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation. 2021: 479-494

[141] Ba J, Duck GJ, Roychoudhury A. Efficient greybox fuzzing to detect memory errors//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. Rochester, USA, 2022: 1-12

[142] Chen X, Shi Y, Jiang Z, Li Y, Wang R, Duan H, Wang H, Zhang C. MTSan: A feasible and practical memory sanitizer for fuzzing cots binaries//Proceedings of the 32nd USENIX Security Symposium. Manheim, USA, 2023: 841-858

[143] Cho M, An D, Jin H, Kwon T. BoKASAN: Binary-only kernel address sanitizer for effective kernel fuzzing//Proceedings of the 32nd USENIX Security Symposium. Manheim, USA, 2023: 4985-5002

[144] Atlidakis V, Godefroid P, Polishchuk M. Checking security properties of cloud service rest apis//Proceedings of the 2020 IEEE 13th International Conference on Software Testing, Validation and Verification. Porto, Portugal, 2020:

387-397

[145] Chen Y, Yuan Y, Wang S. OBSan: An out-of-bound sanitizer to harden dnn executables//Proceedings of the 2023 Network and Distributed System Security Symposium. San Diego, USA, 2023

[146] Zhong H, Mei H. An empirical study on api usages. IEEE Transactions on Software Engineering, 2019, 45(4): 319-334

[147] Legunsen O, Hassan WU, Xu X, Rou G, Marinov D. How good are the specs? A study of the bug-finding effectiveness of existing java api specifications//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. New York, USA, 2016: 602-613

[148] Robillard MP, Bodden E, Kawrykow D, Mezini M, Ratchford T. Automated api property inference techniques. IEEE Transactions on Software Engineering, 2013, 39(5): 613-637

[149] Liu J, Lin J, Ruffy F, Tan C, Li J, Panda A, Zhang L. NNSmith: Generating diverse and valid test cases for deep learning compilers//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2. Vancouver, Canada, 2023: 530-543

[150] Zou Y-H, Bai J-J, Zhou J, Tan J, Qin C, Hu S-M. TCP-fuzz: Detecting memory and semantic bugs in tcp stacks with fuzzing//Proceedings of the 2021 USENIX Annual Technical Conference. 2021: 489-502

[151] Yang Y, Kim T, Chun B-G. Finding consensus bugs in ethereum via multi-transaction differential fuzzing//Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation. 2021: 349-365

[152] Kim S, Kim T. RoboFuzz: Fuzzing robotic systems over robot operating system (ros) for finding correctness bugs//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, Singapore, 2022: 447-458

[153] Bernhard L, Scharnowski T, Schloegel M, Blazytko T, Holz T. JIT-picking: Differential fuzzing of javascript engines//Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. Los Angeles, USA, 2022: 351-364

[154] Wang Z, Yan M, Chen J, Liu S, Zhang D. Deep learning library testing via effective model generation//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. USA, 2020: 788-799

[155] Zhong H. Enriching compiler testing with real program from bug report//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. New York, USA, 2023: 1-12

[156] Zhong H. Which exception shall we throw? //Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. New York, USA, 2023: 1-12

WANG Qin-Ying, Ph. D. candidate. Her research interests include IoT security, software and system security.

XU Jia-Cheng, Ph. D. candidate. His research interests include fuzz testing and orating system security.

LI Yu-Wei, Ph. D., associate professor. Her interested research directions include system and software security, fuzz testing and vulnerability detection.

PAN Zu-Lie, Ph. D., professor. His current research interests include network security and software security.

ZHANG Yu-Qing, Ph. D., professor. his current research interests include network and system security.

ZHANG Chao, Ph. D., associate professor. His current research interests include software and system security.

JI Shou-Ling, Ph. D., professor. His current research interests include AI security, data driven security, and software and system security.

Background

This research belongs to the software and system security area, focusing on the research progress of smart fuzzing in the field of vulnerability detection area. With the increasing scale and complexity of software systems in recent years, along with the continuous growth in the number of security vulnerabilities and their expanding impact, the global security situation remains challenging. Fuzzing is a promising way to find security bugs in practice and has attracted extensive attention from academia and industry. Thus far, intensive research with advanced techniques has been devoted to improving the effectiveness of fuzzers in different software and systems.

In recent years, several scholars have conducted systematic studies on fuzz testing spanning from 2016 to 2023. These studies predominantly focus on specific technologies or a particular test target in the fuzzing domain. Moreover, they do not address the problems fuzzing encounters within the context of emerging trends and scenarios, thus falling short of summarizing the current advancements in smart fuzzing. Despite the diverse and considerable efforts invested in improving fuzz testing by researchers and practitioners in recent years, the rapid increase in workload makes it chal-

lenging to develop a comprehensive and consistent understanding of fuzz testing.

In this survey, we introduce a taxonomy specifically designed around the problems posed by smart fuzzing techniques. In addition, we provide a methodical and analytical review of the current landscape of smart fuzzing research, highlighting the strengths and weaknesses inherent in existing methodologies. Our critical evaluation not only synthesizes the state-of-the-art but also casts light on potential avenues for future inquiry and practical applications within the domain of smart fuzzing.

This work was partly supported by the Joint Key Program of the National Natural Science Foundation of China (Grant No. U1936215), the Program of the Young Scientists Fund of the National Natural Science Foundation of China (Grant No. 62202484), and the Joint Key Program of the National Natural Science Foundation of China (No. U2336203).