CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 44 卷第 6 期 2022 年 6 月
Vol. 44, No. 6, Jun. 2022

文章编号:1007-130X(2022)06-1046-10

并行模糊测试综述*

顾涛涛,卢帅兵,李　响,况晓辉,赵　刚
(军事科学院系统工程研究院信息系统安全技术国防科技重点实验室,北京 100101)

摘　要:软件脆弱性已成为互联网安全的主要威胁来源,软件脆弱性分析技术的重要性日益突出。模糊测试是脆弱性分析的热点技术之一,通过持续生成测试用例、动态监控目标代码执行和反馈调节变异策略的方法尝试触发程序异常,具有部署便捷、适用性广和效果直观的优点。随着测试目标的复杂性增加,从业人员对模糊测试的效率提出了更高的要求。并行模糊测试通过并行执行、任务分解和共享信息等方法提高脆弱性分析的效率。首先,分析了基于覆盖反馈的模糊测试面临的主要挑战;之后,探讨了并行模糊测试的解决思路和方案,从系统结构、任务划分、语料库共享和崩溃去重等方面对并行模糊测试进行了综述;最后,总结了现有并行模糊测试的优缺点,并对未来发展方向进行了展望。

关键词:模糊测试;并行模糊测试;任务分发;语料库共享;崩溃去重
中图分类号:TP311.55　　**文献标志码**:A
doi:10.3969/j.issn.1007-130X.2022.06.012

Overview of parallel fuzzing

GU Tao-tao, LU Shuai-bing, LI Xiang, KUANG Xiao-hui, ZHAO Gang
(National Key Laboratory of Science and Technology on Information System Security, Institute of System and Engineering, Academy of Military Sciences, Beijing 100101, China)

Abstract: Software vulnerability has become the main threat of Internet security, so software vulnerability analysis technology has become increasingly prominent. As one of the hotspot technologies in vulnerability analysis, fuzzing triggers program exceptions by continuously generating test cases, dynamically monitoring the execution of target code, and implementing feedback adjusting variation strategies. Fuzzing has the advantages of convenient deployment, wide applicability and intuitive effect. However, the dynamic execution, variation and feedback mechanism of fuzzing is time-consuming, which affects the efficiency of vulnerability analysis. However, parallel fuzzing improves the efficiency of vulnerability detection with the help of parallel execution, task decomposition and information sharing. Firstly, the main challenges of fuzzing based on coverage feedback are analyzed. Besides, the ideas and solutions of parallel fuzzing are discussed. In addition, the system structure, task division, corpus sharing, crash de-duplication and other aspects of parallel fuzzing are summarized. Finally, the advantages and disadvantages of existing parallel fuzzing are summarized, and the future development direction is prospected.

Key words: fuzzing; parallel fuzzing; task division; corpus sharing; crash de-duplication

* **收稿日期**:2020-11-30;**修回日期**:2021-03-18
通信作者:况晓辉(xiaohui_kuang@163.com)
通信地址:100101 北京市朝阳区安翔北路 10 号
Address: 10 Anxiang North Road, Chaoyang District, Beijing 100101, P. R. China

1 引言

当今信息化时代,软件产业飞速发展,从国防技术、工业控制、航空航天、金融证券、邮电通信和医疗卫生等专业领域到日常生活中的衣食住行,软件都扮演了重要的角色。随着人们对软件功能需求越来越多,软件变得越来越复杂,代码的规模也大幅增大。不可避免地,软件的脆弱性问题也越来越多,软件的安全性问题日益凸显[1]。近年来,利用软件脆弱性进行恶意攻击的事件层出不穷。2014 年,美国银行摩根大通遭黑客攻击,泄露了超过 8 300 万个账户、7 600 万个家庭和 700 万家小企业的相关数据。2017 年 5 月 12 日被披露的勒索病毒 “WannaCry”,是利用一个名为“永恒之蓝”的 Windows 系统脆弱性[2]制作而成的。该勒索病毒席卷了全球 100 多个国家,对医疗、教育和银行等重要机构进行攻击,造成了巨大的经济损失。

软件脆弱性分析已经成为软件安全领域的一个重要研究方向[3]。安全研究人员使用各种方法来挖掘软件的脆弱性,例如代码审计、数据流分析、动态符号执行和模糊测试等。在挖掘软件脆弱性的研究中,以代码审计为代表的手动分析方法需要依赖专业人员的经验知识,难以大规模应用;数据流分析[4]等静态分析方法存在漏报率和误报率高的缺点,实际应用受限;动态符号执行技术[5]难以解决状态爆炸导致的约束求解困难等问题;而模糊测试[6]因自动化程度高、误报率低等优点,逐渐得到产业界和学术界的关注和认可。然而,真实程序规模庞大、复杂多样,利用模糊测试技术发现软件中潜在脆弱性时也面临着代码覆盖度低、测试效率低等诸多挑战。随着并行计算技术的发展,并行模糊测试技术已成为产业界和学术界的研究热点。早在 2016 年,微软[7]和谷歌[8]相继发布了各自的分布式模糊测试服务。微软的付费服务 Springfield 是一款基于云的脆弱性检测工具,结合模糊测试帮助开发人员发现其应用程序的脆弱性。谷歌的开源项目 OSS-Fuzz(continuous Fuzzing for Open Source Software)利用了最新的模糊测试技术和可扩展分布式技术,为开源软件进行持续性的模糊测试。截至 2020 年 6 月,OSS-Fuzz 已在 300 个开源项目中发现超过 20 000 个漏洞。近期,谷歌开放了 OSS-Fuzz 后端 ClusterFuzz[9]的源代码,微软也开源了基于 Microsoft Azure 云服务的模糊测试平台 OneFuzz[10]。在学术界,安全领域的四大顶级会议每年都接收大量关于模糊测试的论文[11-13],也有越来越多的研究团队开始关注模糊测试的效率问题。

本文在简述模糊测试技术及其面临的挑战的基础上,建立了并行模糊测试的模型框架,从系统结构、任务划分、语料库共享和崩溃去重等方面,分析了当前并行模糊测试技术的研究进展,从并行效率和并行有效性等方面对已有工作进行了评估,讨论了并行模糊测试技术的可能发展方向。

2 模糊测试技术面临的挑战

2.1 模糊测试的基本流程

模糊测试最早可追溯到 1988 年,在威斯康辛大学 Miller 教授[14]的计算机实验课上,模糊生成器(Fuzz generator)的概念被首次提出,用于测试 Unix 程序的健壮性,即用随机数据来测试程序直至程序崩溃。模糊测试一般流程是通过自动化或者半自动化的方法构造海量的测试用例并输入到目标测试程序;然后执行目标测试程序并监控执行过程,检查程序的行为有无异常;最后对引起程序异常行为的输入进行详细分析,确认目标程序的脆弱性。测试过程如图 1 所示。

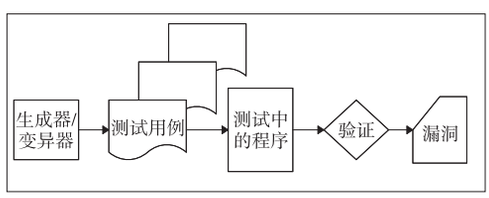

Figure 1 General flow of fuzzing

图 1 模糊测试一般流程

模糊测试技术分为基于生成(Generated-based)的模糊测试技术和基于变异(Mutation-based)的模糊测试技术。基于生成的模糊测试技术是模糊测试器(Fuzzer)利用目标程序输入规定的语法来自动生成符合目标程序输入的测试用例,而基于变异的模糊测试则是模糊测试器通过对程序输入进行变异(如位翻转、字节反转、算数增减和拼接等操作)生成大量的测试用例。传统的模糊测试工具采用完全随机的变异策略,测试效果很差,有研究人员把遗传算法引入了模糊测试,通过反馈机制能消除一部分随机性。其中覆盖度是被广泛采用的反馈内容,当前最流行的基于变异的覆盖度反馈模糊测试工具 AFL(American Fuzzy Lop)[15]在执行已经插桩的程序过程中,跟踪测试用例经过的程序路径并记录下来,优先保留能触发新路径的

测试用例,将其加入种子库,进行下一次迭代,通过这样的覆盖度反馈,能引导模糊测试快速探索更深的程序路径。相应的流程图如图2所示。由于基于覆盖度反馈方式是模糊测试领域的重要分支,下面着重介绍该技术面临的挑战。

2.2 基于覆盖度反馈模糊测试面临的主要挑战

基于覆盖度反馈的模糊测试的有效性,大量研究人员对图2中的初始种子构建、种子筛选、种子变异、待测目标程序执行环境的构建和测试任务的执行等环节进行了大量改进。基于覆盖度反馈的模糊测试技术面临的主要挑战如下所示:

(1)如何获得初始输入?最先进的基于覆盖度反馈的模糊测试技术都采用基于变异的方式生成测试用例。该策略使测试技术的有效性在很大程度上依赖初始种子的质量,良好的初始种子可以显著提高测试的效率和效果。当前关注这个问题的工作有很多。Wang等[16]提出了一种数据驱动的种子生成方案Skyfire,从输入中学习概率上下文敏感语法,然后在生成输入时,把学到的知识再利用起来。随着机器学习技术的迅速发展,来自微软研究院的Godefroid等[17]使用基于神经网络的统计机器学习技术来自动生成测试用例。

(2)如何从种子池中筛选种子?在测试过程中,执行引擎会重复地从种子库中选择种子,在新一轮循环开始之前会对选择的种子进行突变,如何从种子池中选择好的种子是模糊测试面临的另一个问题。Böhme等[18]提出了基于马尔科夫链的种子选择算法,为执行低频路径的种子分配更多的能量,更好地测试低频路径。导向型模糊测试[19]采用定向选择策略,将一些易受攻击的代码定义为目标位置,优先选择距离目标位置更近的测试用例。模糊测试器EcoFuzz[20]采用了一种自适应的种子调度算法,在AFL上运行时获得了极佳的节能优势。

(3)如何进行变异?基于覆盖度反馈模糊测试技术通过变异种子生成测试用例。然而,如何变异出能够覆盖更多程序路径和触发更多漏洞的测试用例是一个关键问题。Rawat等[21]提出了一种集成静态分析和动态污点分析的应用感知灰盒模糊测试工具Vuzzer。该工具先利用静态分析提取影响控制流的即时值、魔法值和其他特征字符串,再利用动态污点分析技术收集信息,影响程序控制流分支。Gan等[22]提出了基于数据流的模糊测试器GREYONE,利用数据流信息,设计了模糊测试驱动的污点推断模型FTI(Fuzzing-driven Taint Inference)模型,可以精准推断程序路径分支转移与测试用例偏移字节的关系。

(4)模糊测试技术如何适应多样化的目标程序场景?比如对操作系统进行模糊测试时,由于操作系统内核位于计算机体系底层,特权级高,不仅需要处理硬件管理事务,还要对上层程序的运行提供支持,这就导致监控内核运行时存在一定的技术困难,必须针对性地构建相应的测试环境和插桩,以及进行其他一些预处理工作。现在大部分的内核模糊测试都不同程度地利用了虚拟化技术来监控运行内核[23-27]。当前代表性的工作有kAFL、HFL、Razzer和JANUS[24-27]等等,文献[28]研究了如何测试虚拟机管理器。文献[29]研究了如何测试数据库。文献[30]将模糊测试技术应用到了无人车测试中。

(5)如何加快测试任务的执行速度?根据概率论的“大数定律”,只要重复次数够多,随机性够强,偶然的事件也会成为必然事件,模糊测试就是“大数定律”的典型应用。随着被测目标程序代码量的快速增长,想要获得更高的脆弱性发现率,需要快速执行大量的测试用例。在已有的研究[15]中,流行的AFL利用Linux内核中的fork调用,每一次执行测试用例时,复制一个已加载好的进程,避免了重复执行程序运行前的初始化工作,这样有效提升了单位时间执行测试用例的数量。在模糊测试中,跟踪程序的执行过程也是关键的一环。文献[31]利用Intel PT技术来跟踪二进制程序的模糊

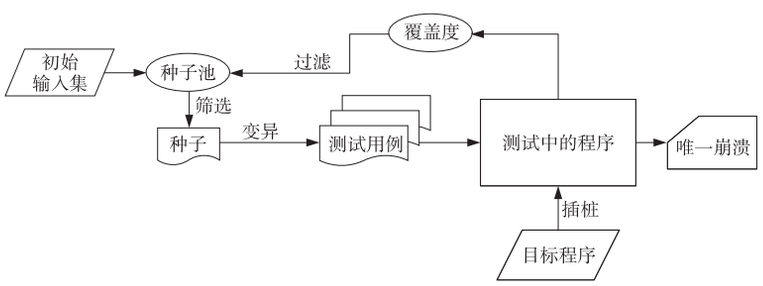

Figure 2 Fuzzing process based on coverage feedback

图2 基于覆盖度反馈模糊测试的流程

测试,大大降低了单次执行的开销。Nagy 等[12]提出了通过放弃跟踪不触发新的代码覆盖的执行过程,来降低模糊测试过程中的跟踪执行开销。Zhou 等[13]提出的 Zeror,在保证模糊测试过程中收集信息的精度的同时,提升了执行效率。

能快速地执行大量测试用例是模糊测试取得成功的主要原因之一。面对越来越复杂的测试任务,现有的模糊测试方法的效率遇到了许多瓶颈,如亟需发布的项目无法被快速有效地测试,影响产品发布时间,以及对关键脆弱性的快速查找效率低等问题。因此,越来越多的研究人员开始利用并行计算来提升模糊测试的效率和有效性。比如,谷歌利用云服务将其可扩展的模糊测试基础设施 ClusterFuzz[9]部署在 25 000 个核心上,这是典型的并行模糊测试设施。截至 2020 年 9 月,ClusterFuzz 已经在谷歌开发的产品中发现超过 25 000 个漏洞[32]。

3 并行模糊测试模型框架

并行模糊测试使用多个模糊测试器同时测试目标程序,是提升模糊测试效率的有效方法。并行模糊测试模型框架如图 3 所示,全局主节点负责分发测试任务,避免子节点重复工作,各个子节点之间通过分享种子等执行信息来协同探索目标程序,最后去除各个子节点发现的重复崩溃,得到全局不重复唯一崩溃集合。

当前并行模糊测试研究的重点主要包括:(1)并行模糊测试体系结构研究,包括控制器/调度器的设计,模糊测试器的同构/异构选择(各子节点是否采用不同的模糊测试器)等;(2)任务划分问题,有效划分能减少各个模糊测试器的重复工作,提升测试的有效性;(3)语料库共享问题,即共享什么,如何共享,共享时机;(4)崩溃去重问题,在大规模的并行模糊测试节点下,有效的崩溃去重方法能大大提升系统的可用性。本文在表 1 对现有并行模糊测试工作的特征进行了概述。

4 并行模糊测试研究进展

4.1 系统部署规模及结构

从体系结构角度看,已有研究工作可分为单机/多机并行、主从/对等结构和模糊测试器同构/模糊测试器异构等类型。

现在的单机多核资源丰富,有些研究[15,33-42]将并行模糊测试部署在单机多核上,其并行度不高,一般是 100 以内。比如 AFL[15]的并行模式和 LibFuzzer[40],模糊测试技术在并行时没有数据依赖,理想情况下,效率提升可随并行度增加而成倍增加,但是在以上 2 个 Fuzzer 并行度达到 16 及以上时,会出现单核性能下降的情况,即单个模糊测试器执行测试用例数大大减少。文献[33]指出这是由于在单机内,AFL 和 LibFuzzer 为加速而采用的 fork 系统调用和文件系统,在并行度较大时,频繁调用或读写种子会出现资源竞争,这严重降低了单核模糊测试的性能。文献[41]的 Honggfuzz 是

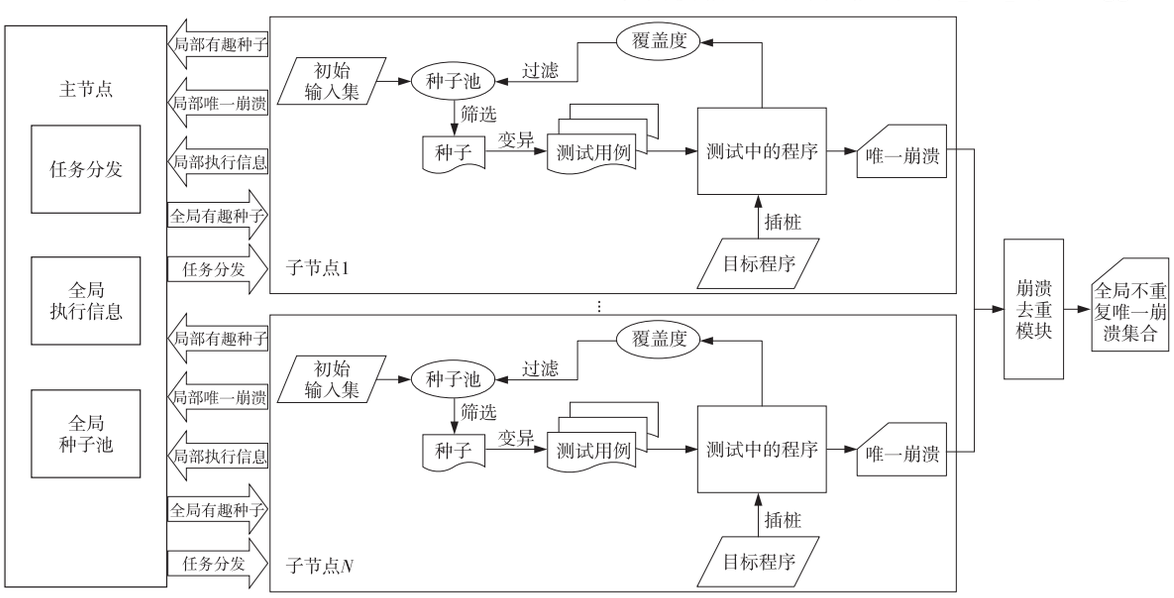

Figure 3 Framework of parallel fuzzing model

图 3 并行模糊测试模型框架

Table 1 Overview of the parallel fuzzers

表 1 并行模糊测试器的概述

模糊测试器	异构	开源	同步时间	同步方式	同步内容	单机/多机	崩溃去重
ClusterFuzz[9]	×	√	---	云计算资源	种子	多机多核	有
AFL-P[15]	×	√	fuzz_one()后	内存共享	种子	单机多核	无
Perf-Fuzz[33]	×	√	即时同步	内存共享	种子、代码覆盖度	单机多核	无
Large[34]	×	×	---	内存共享	种子、基本块执行流	多机多核	有
PAFL(Ye)[35]	×	×	按需同步	---	---	单机多核	无
OPAFL[36]	×	×	周期同步	内存共享	种子、代码覆盖度	单机多核	无
PAFL(Liang)[37]	×	×	周期同步	内存共享	种子、代码覆盖度	单机多核	无
EnFuzz[38]	√	√	周期同步	内存共享	种子、代码覆盖度	单机多核	有
CUPID[39]	√	×	周期同步	内存共享	种子、代码覆盖度	单机多核	无
LibFuzzer[40]	×	√	---	主节点	种子	单机多核	无
Honggfuzz[41]	×	√	---	主节点	种子	单机多核	无
P-Fuzz[42]	×	×	周期同步	数据库	种子、代码覆盖度	多机多核	无
UNIFUZZ[43]	×	√	即时同步	数据库	种子、代码覆盖度	多机多核	无
Grid[44]	×	×	---	---	---	多机多核	无
Roving[45]	×	√	周期同步	主节点	种子	多机多核	无
DisFuzz-AFL[46]	×	√	周期同步	主节点	种子	多机多核	无
Manul[47]	×	√	周期同步	主节点	种子	多机多核	无

自动支持多进程和多线程执行的模糊测试技术,不需要像 AFL 那样手动部署多个 Fuzzer 副本。也有大量研究工作[32,43-47]将并行模糊测试部署在多机多核上,称作分布式模糊测试技术。例如典型的 ClusterFuzz[32],它将并行模糊测试部署在约 25 000 个核心的谷歌云上,作为基础设施为开发人员提供软件模糊测试服务。Grid[44]利用网格计算资源,将分布在不同地方的模糊测试器通过网络通信向服务端申请和提交模糊测试任务。文献[45,46]中也是多机并行工作,在分布式节点上分别运行 AFL 的并行模式。Manul[47]是一个支持测试开源程序或黑盒二进制程序的分布式模糊测试工具,可以部署在 Windows、Linux 和 MacOS 上。文献[43]中的测试器 UNIFUZZ 也支持分布式部署,并评估验证了 128 个节点同时执行的有效性。

在并行模糊测试的框架选择方面,只有少数的工作[15,40,41]选择了对等结构,即整个并行模糊测试中的所有节点都只执行测试任务,不存在主节点去调控子节点的执行。更多的并行模糊测试工作都选择了主从结构,这是由于模糊测试本身的盲目性,主节点可以实现丰富的控制、调度和分析策略来对各个子节点进行调节,使各节点高效协同地工作。比如,文献[42,43]的测试器 P-Fuzz 和 UNIFUZZ 在主节点实现了很好的调度策略和任务分发策略来指导子节点的执行。

近来,越来越多的研究团队认为异构模糊测试器的并行更加有效。文献[38]的测试器 EnFuzz 从理论和工程上证明了各并行节点模糊测试器的多样性是提升模糊测试有效性的关键点,不同的模糊测试器在并行执行目标程序时,对不同路径约束的求解能力不一样,协作执行可以到达目标程序更深的路径。文献[39]中的测试器 CUPID 也采用异构模糊测试器并行工作。混合模糊测试[48-50]也可以看作是一种特殊的异构并行模糊测试技术。

4.2 任务划分

程序内部包含多种状态。使用并行模糊测试技术测试目标程序时,使每个模糊测试器分别测试目标程序的不同状态,能有效提高测试效率。任务划分是把一个测试任务精准地划分成不同的子任务,不同任务间程序状态不重复。已有研究工作可分为基于状态空间的任务划分和基于输入空间的任务划分 2 类。

基于状态空间的任务划分是指:被测目标程序内部有多条路径,表示不同的程序状态,根据路径将程序内部的状态划分为多个部分,不同的模糊测试器执行不同部分,完成并行测试。典型研究工作包括 Liang 等[37]开发的并行模糊测试器 PAFL,它采用了一种基于目标程序分支位图(Branch Bitm-

ap)的任务划分方法,将其分为 N 份,每个模糊测试器负责执行属于自己的目标程序分支。开始执行时,先根据种子命中的每个分支的命中次数来选择种子,在该种子命中的所有分支中,命中次数最少的分支是否属于当前模糊测试器任务范围,如果属于则正常执行,否则略过该种子并继续检查下一个种子。通过这种任务划分策略,每个模糊测试器分别测试程序的不同区域,增加了并行模糊测试的有效性。然而,Li 等[36]认为 PAFL 框架存在一定缺陷,且当测试器 FairFuzz[51]应用该框架时,只会执行命中最稀有分支的测试用例,命中其他有价值分支的用例会被抛弃,因此设计了更优的任务分发方式:把单个测试用例对应的多个分支都作为一个任务以备候选,这样就不会错过执行其他有价值的分支。

Ye 等[35]提出的 PAFL 也是采用类似的任务划分和分发策略,首先通过动态执行信息周期性筛选出有价值的程序状态(执行次数较少),这些程序状态之间的关联性较低;然后将其周期性地分发给各个子实例,各个实例通过"别针"技术(通过避免变异影响路径跳转的字节)使各节点执行分配到的程序状态,一定周期后,各子节点上传执行过的状态到主节点,更新全局执行状态,然后再重新进行状态分配,如此重复直至完成任务。Grid 框架[44]也是根据测试任务本身来划分,在其服务器端基于 SPIKE 的分割思想,把目标任务划分为不同的子任务,存储在 HTTP 服务器上,远程节点在 Zookeeper 中获取任务简述,然后到 HTTP 服务器去下载相应任务执行。

基于状态空间来划分任务的工作不多,各个实例专注于执行其分配到的任务,在提升模糊测试效率的同时,也增加了代码覆盖度。但是,由于其依赖于目标程序结构,使得模糊测试的并行度有限,不适用于大规模扩展。

并行模糊测试工作大都是基于输入空间来划分任务。基于输入空间任务划分的依据是不同的输入测试用例能触发的程序状态不一样,因此每个模糊测试器在执行触发不同路径的测试用例时,也能保证并行模糊测试的有效性。这方面的代表性工作有 UNIFUZZ 和 P-Fuzz,它们都采用了主从结构,通过主节点的调度策略,分配触发不同路径的种子到不同的从节点。具体来说,当各个子节点发现有趣的测试用例时,将其传送到主节点,然后在全局视角下,主节点进一步筛选出有趣种子,再通过"请求-回应"机制(子节点实例完成各自的任务后,主动向主节点申请任务,主节点回应后,分发种子,这在一定程度上实现了负载均衡)分发任务到各子节点。大部分基于输入空间来划分任务的并行模糊测试研究[33,42,43,47]都需要一个主节点从全局的角度去调度种子,而 AFL、LibFuzzer 和 Honggfuzz 直接共享所有的种子,导致了大量重复工作。基于异构模糊测试器(比如 EnFuzz、CUPID)的任务划分也是基于输入空间划分的,各测试器之间差异很大,对不同路径约束求解能力不一,变异种子的方式不同,因此即使被分配了相同的种子,也有可能触发不同的目标程序状态,这样就保证了资源利用的有效性。当然给并行异构模糊测试器加入主节点,以确定哪一个模糊测试器可以从一个特定的种子受益,更能提升异构模糊测试器并行时的有效性。梁洪亮等[52]将符号执行与模糊测试的随机变异方式相结合,提出了一种适用于并行化环境的路径取反算法来生成测试用例,可使各模糊测试器触发不同的程序状态。

良好的任务划分策略不仅能高效地利用硬件资源,而且还能很好地提升测试效果。基于状态空间划分任务的方式,更为直观,利用被测任务程序的内部特征,符合我们对常规并行计算的理解。但是,由于模糊测试任务的特性,基于状态划分的并行工作使实例间的语料库共享带来的收益不大。另外,由于现有的大部分工作的划分方式都依赖位图实现,而位图不能全面准确地代表程序状态空间,因此精准地划分任务也是一个难点。基于输入空间划分任务,即依赖种子来划分任务,是大规模并行测试采用较多的方式,在大规模节点下,同时借助全局和局部的执行信息能较好地避免任务冲突,提升效率。

4.3 语料库共享

并行模糊测试中各模糊测试器之间的有效共享信息能提高各个节点对测试程序路径的探索效率。本节集中讨论并行模糊测试中语料库共享的内容、时机和开销。

各子节点之间共享有趣的种子(覆盖新路径的种子)是促进节点间良好协作的关键。早期的研究直接共享有趣种子[15,45,46],每个模糊测试器遍历其他模糊测试器输出目录中的有趣种子,判断其对自己来说是否为有趣新种子,如果有趣则加入本地种子库,完成共享操作。LibFuzzer 在并行工作时,每个模糊测试器在独立的进程中运行,共享同一个输出目录,每个模糊测试器直接共享其他实例发现的有趣种子。Honggfuzz 自动部署并行模式,主进程

负责模糊测试执行环节以外的所有工作,子进程并行执行多个测试用例,也通过同一个输出目录实现语料库共享。ClusterFuzz的基本执行单位bot直接共享有趣种子,每个基本执行单位把本地的有趣种子上传到谷歌云存储平台中,然后再定期从云存储中的全局语料库下载种子。在近期的一些改进工作中,共享内容不仅包括有趣种子,还有各节点的其他信息,这部分信息主要用来指导节点间更好地共享种子。共享种子以外的其他信息的并行工作都采用主从结构,文献[34,37,42,43,47]都设计了一种全局/本地的语料库共享机制,全局状态由所有模糊测试器共同维护,本地状态自己维护,这样可以从全局的角度出发判断哪个种子该分配到哪个子节点。Liang等[37]提出的PAFL就是在全局/本地框架下通过Upload、Update和Pull操作实现种子共享的,首先通过Upload操作上传各实例的本地指导信息(如分支覆盖信息)到主节点,接着通过Update操作来更新全局信息,最后通过Pull操作为各实例获取当前的全局信息和种子,同时更新自己的本地信息和种子。P-Fuzz和UNIFUZZ在语料库共享过程中,当各实例发现有趣的种子时,就把有趣种子和本地实例的代码覆盖位图传送到主节点,然后主节点合并更新全局代码覆盖位图,最后根据全局覆盖位图为各实例分发种子,同时用全局位图更新本地位图。Manul也在执行过程中共享了代码覆盖位图。Li等[34]认为在大规模并行共享时,代码覆盖度的载体位图会成为共享时的瓶颈,因此通过静态分析收集基本块的跳转,设计了一个更适合大规模并行时共享的信息载体。UNIFUZZ在共享代码覆盖信息和种子的同时,还分享了模糊测试过程中的信息(比如该种子的执行次数),这个信息伴随有趣种子上传。UNIFUZZ是基于AFL开发的,主节点利用种子的执行次数来评价种子,进而筛选出更优的种子。在未来的研究工作中,还可以共享各子节点间更丰富的测试信息来互相协作。

语料库共享时机的选择取决于并行度和共享内容。由于模糊测试自动执行大量测试用例,大规模实例并行时需要考虑共享时的竞争问题,对于共享代码覆盖信息的工作,由于子节点的覆盖信息更新很快,频繁地共享指导信息会导致性能下降。因此,大部分研究[9,15,37,40,45-47]都是周期性共享信息。也有部分工作[44,45]基于主从结构实现即时共享,当各子实例发现有趣种子时,即时将覆盖信息和种子上传到主节点,通过加锁/解锁来避免竞争,当各子节点完成任务时,即时从主节点获取任务和种子。在异构模糊测试器[38,39]并行时,由于各子节点执行速度不一致,因此不适合周期性共享,采用了全局同步-本地异步的共享方式,当各子节点发现有趣种子时,即时上传到主节点,然后主节点周期性地向各子节点共享有趣种子。也有Ye等[35]的PAFL根据其设计的共享机制采取了按需请求(On-Demand)的方式,当子节点没有特定的种子时,才向主节点申请相应的种子。

并行模糊测试框架引入语料库共享环节带来的共享开销是研究人员研究的又一个重点。共享开销与共享机制和共享途径相关。对于各个子实例都是平等结构,没有主节点,各个实例之间共享时的复杂度为$O(n^2)$,比如AFL的并行模式,因此,文献[33]的Perf-Fuzz针对AFL的共享开销问题设计了内存共享,在内存中维护一个(Testcase_log)循环队列,每个日志存储测试用例的文件名、大小、哈希值和代码覆盖位图,各个子节点从队列中依次取出种子简要信息,将其中代码覆盖位图和自己的位图对比,决定是否执行该测试用例。主从结构框架的复杂度为$O(n)$,显然更小。从共享途径来看,单机内的工作[33,37,49]可以采用内存共享来降低开销。多机分布式部署的工作[9,45,47-49],都是通过高性能网络通信,基于文件系统传输有趣种子。其中Manul[49]在单机内使用共享内存存储共享信息,在分布式环境下,采用虚拟共享内存存储共享信息。UNIFUZZ[45]和P-Fuzz[44]采用主从结构,在主节点使用数据库存储种子摘要信息,通过摘要信息去调度种子,进而降低共享开销。

4.4 崩溃去重

模糊测试的崩溃去重环节是提高测试结果可用性的关键[53]。基于反馈的算法会把触发崩溃的测试用例保存为种子,进一步对其变异执行,此时新的测试用例很容易执行相似的函数,出现相同的错误,导致重复的崩溃,没有崩溃去重环节会严重影响对测试结果的判断。当前流行的单核模糊测试实例崩溃去重方法有:堆栈回溯哈希、基于覆盖的重复数据删除和语义感知的重复数据删除。并行模糊测试技术的崩溃去重是新引入的问题,当模糊测试实例并行时,各实例仅在各自内部进行崩溃去重,但并行节点之间也会出现重复崩溃,特别当节点数过多时,这个问题会变得不可忽视。当前的并行模糊测试研究中,只有少部分工作提出了改进,EnFuzz在主节点维护了一个全局崩溃列表,利

用全局代码覆盖位图和内存检测工具去除全局重复崩溃,并对崩溃进行分类;Li 等[34]基于基本块的跳转设计了适用于大规模并行的数据结构来简化执行信息表示,同时利用测试用例触发的基本块跳转序列和自身的堆栈校验和过滤去除重复崩溃;ClusterFuzz 部署在超过了 25 000 个并行实例,其内部通过谷歌云强大的算力来统计全局崩溃信息,利用内存错误检测工具和漏洞相关联的堆栈跟踪信息去除重复崩溃。在大规模并行模糊测试技术中,如何精准且低开销地去除重复崩溃是该技术面临的重要挑战。

4.5 评估

当前的并行模糊测试技术通过对硬件资源的有效利用,大大提升了其模糊测试的效率。Perf-Fuzz 从操作系统层面改进了 AFL 和 LibFuzzer,消除文件系统和 fork 调用瓶颈,引入内存共享机制和更轻量级的启动任务策略,验证了在单机内部署 120 个实例时该改进工作的有效性。P-Fuzz 和 UNIFUZZ 在到达代码覆盖度(单核模糊测试的代码覆盖度)上限之前,随着资源增加,有效地减少了执行时间。也有部分工作相对于单核代码覆盖度,不仅减少了执行时间,还增加了覆盖度。在 PAFL[35,37]的评估环节中,由于其采用基于状态空间的任务划分策略,各实例专注于执行程序各部分,使验证环节中的程序代码覆盖度上有一定提升,AFLFast[18] 和 FairFuzz[53] 应用并行框架 PAFL[37]部署在 4 个实例上,分别提升了 8%和 17%的代码覆盖度,多触发了 79%和 52%的唯一崩溃。作为第一个异构并行模糊测试器,EnFuzz 采用基于输入空间的任务划分策略,从其评估数据来看,有效地提高了代码覆盖度,比参考对象平均多发现了 93%的唯一漏洞,多执行了 48.25%的路径,多覆盖了 17.8%的分支。

5 结束语

并行模糊测试技术的效率提升程度与其并行结构、任务分发策略和语料库共享策略相关。时间效率方面,由于各实例间无数据相关依赖,随着并行实例数的增加,测试时间理论上应线性减少,但是由于模糊测试固有的随机性以及并行框架引入的其他开销,依然存在一定的时间效率损失。大部分并行工作选择了主从结构,主节点拥有全局的执行信息,其任务分发机制、语料库共享机制和崩溃去重机制都能在主节点中更好地实现。从部分并行模糊测试技术的评估结果来看,其效率提升与任务分发策略和语料库共享策略相关。好的策略能减少各节点间的重复工作,提高资源利用率,也能降低节点间通信的额外开销。语料库共享策略方面,在主节点对种子进行筛选比较的方式比各实例遍历各自输出目录的种子更有效;使用内存共享语料库比通过文件系统共享语料库开销低,但实现更复杂;在共享时机的选择上,周期性共享比即时共享更容易实现,但周期设置不当也会导致各部分信息无法及时共享,进而影响有效性。任务分发策略方面,基于输入空间的任务划分策略可扩展性强,实现简单,能有效缩短测试时间,但是会遭遇无法突破代码覆盖度的瓶颈;基于状态空间的任务划分策略使得每一个测试实例专注于执行各自划分到的程序路径,能充分测试该程序状态,在缩短测试时间的同时也提升了代码覆盖度,但会遇到可扩展性问题。从评估结果看,由于各异构模糊测试器对同一问题有不同的解决能力,能探索更深的路径,异构模糊测试器并行技术能提高代码覆盖度,另外由于采用基于输入空间的任务划分策略,因此可以大规模部署。因此,在未来的大规模分布式模糊测试技术中,采用异构模糊测试器是很好的方案。

在未来的异构并行模糊测试技术工作中,可以关注以下几个研究方向:

(1)由于异构模糊测试器的差异性,其各自包含的执行信息有异同之处,因此如何选择更合适的语料库共享策略是可以研究的问题。通过在主节点处对需共享的信息进行统一的转换后再进行判断和共享,可以初步解决这个问题。

(2)异构模糊测试器对测试任务的表现能力不同,从资源调度的角度,对更优的模糊测试器分配更多的资源,从而提升效率。

(3)异构并行模糊测试技术的崩溃去重问题。在并行模糊测试技术的崩溃去重的基础上,各异构模糊测试器可以采用不同的崩溃去重方案,如何在主节点精准地去重是可以研究的问题。

(4)异构并行模糊测试技术的任务分发问题。虽然异构模糊测试器对不同约束的求解能力不一样,但是在主节点分发测试用例时,如何更准确地判断哪一个模糊测试器将会从某个测试用例中受益是可以研究的问题。

(5)异构模糊测试器之间的组合问题。已有部分工作研究如何选择最优的异构模糊测试器组合,但是都是静态选择。考虑到执行过程中需面对程序路径的变化,如何动态调整异构模糊测试器的组

合也是可以研究的方向。

参考文献:

[1] Yu Zhao-hui. A picture to understand the overview of China's Internet network security situation in 2019 [J]. Civil-Military Integration on Cyberspace,2020,35(4):29-30. (in Chinese)

[2] Microsoft security bulletin MS17-010-Critical | Microsoft Docs [EB/OL]. [2020-10-23]. https://docs. microsoft. com/en-us/securityupdates/sec-uritybulletins/2017/ms17-010.

[3] Liu B C, Shi L, Cai Z H, et al. Software vulnerability discovery techniques: A survey[C]//Proc of the 4th International Conference on Multimedia Information Networking and Security,2012:152-156.

[4] Khedker U, Sanyal A, Sathe B. Data flow analysis: Theory and practice[M]. Boca Raton:CRC Press,2017.

[5] Baldoni R, Coppa E, D'elia D C, et al. A survey of symbolic execution techniques[J]. ACM Computing Surveys,2018,51(3):1-39.

[6] Li J, Zhao B, Zhang C. Fuzzing: A survey[J]. Cybersecurity, 2018, 1(1): 1-13.

[7] Microsoft previews project Springfield, a cloud-based bug detector—The AI blog[EB/OL]. [2020-10-22]. https://blogs. microsoft. com/ai/microsoft-previews-project-springfield-cloud-based-bug-detector/.

[8] Serebryany K. Announcing OSS-Fuzz: Continuous fuzzing for open source software[EB/OL]. [2016-12-11]. https://testing. googleblog. com/2016/12/announcing-oss-fuzz-continuous-fuzzing. html.

[9] Google online security blog: Open sourcing ClusterFuzz[EB/OL]. [2020-10-23]. https://security. googleblog. com/2019/02/open-sourcing-clusterfuzz. html.

[10] Project OneFuzz—Microsoft research[EB/OL]. [2020-10-23]. https://www. microsoft. com/en-us/research/project/project-onefuzz/.

[11] Wen C, Wang H J, Li Y K, et al. Memlock: Memory usage guided fuzzing[C]//Proc of the 42nd International Conference on Software Engineering,2020:765-777.

[12] Nagy S, Hicks M. Full-speed fuzzing: Reducing fuzzing overhead through coverage-guided tracing[C]//Proc of 2019 IEEE Symposium on Security and Privacy,2019:787-802.

[13] Zhou C, Wang M, Liang J, et al. Zeror: Speed up fuzzing with coverage-sensitive tracing and scheduling[C]//Proc of the 35th IEEE/ACM International Conference on Automated Software Engineering, 2020:858-870.

[14] Miller B. Project list[EB/OL]. [1988-12-10]. https://pages. cs. wisc. edu/~bart/fuzz/CS736-Projects-f1988. pdf.

[15] Zalewski M. American fuzzy lop[EB/OL]. [2014-12-11]. https://lcamtuf. coredump. cx/afl/.

[16] Wang J J, Chen B H, Wei L, et al. Skyfire: Data-driven seed generation for fuzzing[C]//Proc of 2017 IEEE Symposium on Security and Privacy,2017:579-594.

[17] Godefroid P, Peleg H, Singh R. Learn & fuzz: Machine learning for input fuzzing[C]//Proc of the 32nd IEEE/ACM International Conference on Automated Software Engineering,2017:50-59.

[18] Böhme M, Pham V T, Roychoudhury A. Coverage-based greybox fuzzing as Markov chain[J]. IEEE Transactions on Software Engineering,2017,45(5):489-506.

[19] Böhme M, Pham V T, Nguyen M D, et al. Directed greybox fuzzing[C]//Proc of the 2017 ACM SIGSAC Conference on Computer and Communications Security,2017:2329-2344.

[20] Yue T, Wang P F, Tang Y, et al. EcoFuzz: Adaptive energy-saving greybox fuzzing as a variant of the adversarial multi-armed bandit[C]//Proc of the 29th USENIX Security Symposium, 2020: 2307-2324.

[21] Rawat S, Jain V, Kumar A, et al. VUzzer: Application-aware evolutionary fuzzing[C]//Proc of the 24th Annual Network and Distributed System Security Symposium,2017:1-14.

[22] Gan S T, Zhang C, Chen P, et al. GREYONE: Data flow sensitive fuzzing[C]//Proc of the 29th USENIX Security Symposium,2020:2577-2594.

[23] Li He, Zhang Chao, Yang Xin, et al. Survey of OS kernel fuzzing[J]. Journal of Chinese Computer Systems,2019,40(9):1994-1999. (in Chinese)

[24] Schumilo S, Aschermann C, Gawlik R, et al. kAFL: Hardware-assisted feedback fuzzing for OS kernels[C]//Proc of the 26th USENIX Security Symposium, 2017:167-182.

[25] Kim K, Jeong D R, Kim C H, et al. HFL: Hybrid fuzzing on the Linux kernel[C]//Proc of the 27th Annual Network and Distributed System Security Symposium, 2020:1.

[26] Jeong D R, Kim K, Shivakumar B, et al. Razzer: Finding kernel race bugs through fuzzing[C]//Proc of 2019 IEEE Symposium on Security and Privacy,2019:754-768.

[27] Xu W, Moon H, Kashyap S, et al. Fuzzing file system via two-dimensional input space exploration[C]//Proc of 2019 IEEE Symposium on Security and Privacy,2019:818-834.

[28] Schumilo S, Aschermann C, Abbasi A, et al. HYPER-CUBE: High-dimensional hypervisor fuzzing[C]//Proc of the 27th Annual Network and Distributed System Security Symposium,2020:23-26.

[29] Zhong R, Chen Y, Hu H, et al. SQUIRREL: Testing database management systems with language validity and coverage feedback[C]//Proc of the 2020 ACM SIGSAC Conference on Computer and Communications Security,2020:955-970.

[30] Kim T, Kim C H, Rhee J, et al. RVFUZZER: Finding input validation bugs in robotic vehicles through control-guided testing[C]//Proc of the 28th USENIX Security Symposium, 2019:425-442.

[31] Zhang G, Zhou X, Luo Y Q, et al. PTfuzz: Guided fuzzing with processor trace feedback[J]. IEEE Access, 2018, 6: 37302-37313.

[32] ClusterFuzz [EB/OL]. [2020-10-23]. https://google. github. io/clusterfuzz/.

[33] Xu W, Kashyap S, Min C, et al. Designing new operating primitives to improve fuzzing performance[C]//Proc of the 2017 ACM SIGSAC Conference on Computer and Communications Security,2017:2313-2328.

[34] Li Y, Feng C, Tang C. A large-scale parallel fuzzing system

[C]//Proc of the 2nd International Conference on Advances in Image Processing,2018:194-197.

[35] Ye J X,Zhang B,Li R L,et al. Program state sensitive parallel fuzzing for real world software[J]. IEEE Access,2019,7:42557-42564.

[36] Li S S,Li R L,Ye J X,et al. Adaptive parallel fuzzing with multi-candidate task scheduling[J]. Journal of Physics:Conference Series,2020,1619:012019.

[37] Liang J,Jiang Y,Chen Y L,et al. PAFL:Extend fuzzing optimizations of single mode to industrial parallel mode[C]// Proc of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering,2018:809-814.

[38] Chen Y, Jiang Y, Ma F, et al. EnFuzz: Ensemble fuzzing with seed synchronization among diverse fuzzers[C]//Proc of the 28th USENIX Security Symposium,2019:1967-1983.

[39] Güler E, Görz P, Geretto E, et al. CUPID: Automatic fuzzer selection for collaborative fuzzing[C]//Proc of Annual Computer Security Applications Conference, 2020:360-372.

[40] Serebryany K. libFuzzer - A library for coverage-guided fuzz testing[EB/OL]. [2015-12-11]. https://llvm.org/docs/LibFuzzer.html.

[41] Swiecki R. Honggfuzz[EB/OL]. [2016-11-12]. http://code.google.com/p/honggfuzz.

[42] Song C,Zhou X,Yin Q,et al. P-Fuzz:A parallel grey-box fuzzing framework[J]. Applied Sciences,2019,9(23):5100.

[43] Zhou X,Wang P F,Liu C Y F,et al. UniFuzz:Optimizing distributed fuzzing via dynamic centralized task scheduling [J]. arXiv:2009.06124,2020.

[44] Yan X. Using grid computing for large scale fuzzing[D]. Lisbon:Universidade Nova de Lisboa,2010.

[45] Richo/Roving[EB/OL]. [2020-10-23]. https://github.com/richo/roving.

[46] MartijnB/disfuzz-afl:Distributed fuzzing for afl[EB/OL]. [2020-10-23]. https://github.com/MartijnB/disfuzz-afl.

[47] Mxmssh/manul:Manul is a coverage-guided parallel fuzzer for open-source and blackbox binaries on Windows,Linux and MacOS[EB/OL]. [2020-10-23]. https://github.com/mxmssh/manul.

[48] Stephens N, Grosen J, Salls C, et al. Driller: Augmenting fuzzing through selective symbolic execution[C]//Proc of the 23rd Annual Network and Distributed System Security Symposium,2016:1-16.

[49] Yun I, Lee S, Xu M, et al. QSYM: A practical concolic execution engine tailored for hybrid fuzzing[C]//Proc of the 27th USENIX Security Symposium,2018:745-761.

[50] Chen Y H,Li P,Xu J,et al. SAVIOR:Towards bug-driven hybrid testing[C]//Proc of 2020 IEEE Symposium on Security and Privacy,2020:1580-1596.

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

[52] Liang Hong-liang, Yang Xiao-yu, Dong Yu, et al. Parallel smart fuzzing test[J]. Journal of Tsinghua University (Science and Technology),2014,54(1):14-19. (in Chinese)

[53] Li Y,Ji S,Chen Y,et al. UNIFUZZ:A holistic and pragmatic metrics-driven platform for evaluating fuzzers[J]. arXiv:2010.01785,2020.

附中文参考文献:

[1] 于朝晖.一图读懂《2019年我国互联网网络安全态势综述》[J].网信军民融合,2020,35(4):29-30.

[23] 李贺,张超,杨鑫,等.操作系统内核模糊测试技术综述[J].小型微型计算机系统,2019,40(9):1994-1999.

[52] 梁洪亮,阳晓宇,董钰,等.并行化智能模糊测试[J].清华大学学报(自然科学版),2014,54(1):14-19.

作者简介:

顾涛涛(1995-),男,四川绵阳人,硕士生,研究方向为模糊测试。**E-mail**:gutaotao1995@qq.com

GU Tao-tao, born in 1995, MS candidate, his research interest includes fuzzing.

卢帅兵(1990-),男,河南周口人,硕士,助理研究员,研究方向为操作系统和二进制翻译。**E-mail**:datadancer@163.com

LU Shuai-bing, born in 1990, MS, assistant research fellow, his research interests include operation system, and binary translation.

李响(1983-),女,河北涿州人,硕士,高级工程师,研究方向为软件脆弱性分析和恶意代码检测。**E-mail**:ideal_work@163.com

LI Xiang, born in 1983, MS, senior engineer, her research interests include software vulnerability analysis, and malicious code detection.

况晓辉(1975-),男,湖南新化人,博士,研究员,研究方向为网络与信息安全。**E-mail**:xiaohui_kuang@163.com

KUANG Xiao-hui, born in 1975, PhD, research fellow, his research interest includes network & information security.

赵刚(1969-),男,河北保定人,博士,研究员,研究方向为网络与信息安全。**E-mail**:bisezhaog@163.com

ZHAO Gang, born in 1969, PhD, research fellow, his research interest includes network & information security.