计算机科学
COMPUTER SCIENCE

漏洞基准测试集构建技术综述

马总帅, 武泽慧, 燕宸毓, 魏强

引用本文

马总帅, 武泽慧, 燕宸毓, 魏强. 漏洞基准测试集构建技术综述[J]. 计算机科学, 2024, 51(1): 316-326.

MA Zongshuai, WU Zehui, YAN Chenyu, WEI Qiang. Survey of Vulnerability Benchmark Construction Technique [J]. Computer Science, 2024, 51(1): 316-326.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230300209

漏洞基准测试集构建技术综述

马总帅　武泽慧　燕宸毓　魏　强
信息工程大学数学工程与先进计算国家重点实验室　郑州 450001
(891489656@qq.com)

摘　要　随着软件漏洞分析技术的发展,针对不同漏洞的发现技术和工具被广泛使用。但是如何评价不同技术、方法、工具的能力边界是当前该领域未解决的基础性难题。而构建用于能力评估的漏洞基准测试集(Vulnerability Benchmark)是解决该基础性难题的关键。文中梳理了近20年漏洞基准测试集构建的相关代表性成果。首先从自动化的角度阐述了基准测试集的发展历程;然后对基准测试集构建技术进行了分类,给出了基准测试集构建的通用流程模型,并阐述了不同测试集构建方法的思想、流程以及存在的不足;最后总结当前研究的局限性,并对下一步研究进行了展望。

关键词:漏洞基准测试集;软件漏洞分析;评估指标
中图分类号　TP309

Survey of Vulnerability Benchmark Construction Technique

MA Zongshuai,WU Zehui,YAN Chenyu and WEI Qiang
State Key laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450001,China

Abstract　The development of technology for software vulnerability analysis has led to the widespread use of various techniques and tools for discovering vulnerabilities. Nevertheless,assessing the capability boundary of these techniques,methods,and tools remains a fundamental problem in this field. A vulnerability benchmark for capability assessment plays a pivotal role in solving this problem. The purpose of this paper is to review representative results related to the construction of benchmark test sets over the past 20 years. Firstly,it explains the developmental history of vulnerability benchmark from an automation perspective. Then,it classifies the techniques for constructing vulnerability benchmark and provide a general process model,explaining the ideas and processes of different construction methods and their limitations. Lastly,the limitations of current research are summarized and the future research is prospected.

Keywords　Vulnerability benchmark,Software vulnerability analysis,Evaluation metrics

1　引言

近年来,随着信息技术的飞速发展,软件系统在人们的生活中扮演着越来越重要的角色。与此同时,软件复杂程度的不断提升使得各类软件的正确性与安全性越来越难以保证,而软件漏洞因为其可以被攻击者恶意利用的特性,目前成为了软件系统安全的主要威胁之一[1]。为了有效检测出软件漏洞,研究人员提出了包括模糊测试、符号执行、污点分析以及人工智能检测等动态或静态方法。然而,由于不同的研究人员往往会选择不同的基准评估测试集,因此很难公正地评价漏洞检测工具的性能,同样也无法确定适用场景的漏洞检测工具。

为了使评估过程与结果标准化,漏洞测试集的构建技术逐渐引起了人们的关注,图1给出了漏洞测试集的发展过程。早期主要由人工编写带有漏洞的程序来构造漏洞基准测试集,这种方式的主要优点是可以提供精确的评估基准,缺点是结构简单,不能提供复杂真实的软件环境。之后研究人员基于静态分析或动态分析工具相继提出了基于真实软件的漏洞自动合成方法。合成漏洞的优点是可以提供真实的软件环境,并且具备可扩展性和低成本性,其缺点是无法精确地评估模糊测试工具。由于合成漏洞具有明显的特征,与真实漏洞相比更容易被漏洞检测工具发现[2],因此研究人员针对真实漏洞的注入方法进行了研究[3],并提出了相应的注入工具。真实漏洞可以很好评估漏洞检测工具,避免过拟合问题,缺点是真实漏洞的注入需要研究人员进行大量的前置分析,目前仍停留在半自动化阶段。本文对近年来漏洞基准程序构建领域的相关研究进行了归类和梳理,从漏洞构建和测试评估两个方面对已有的研究进行了阐述,并整理了目前可用于漏洞检测工具的公开基准测试集,在此基础上进一步展望了漏洞测试集构建技术所面临的问题和挑战。本文旨在帮助研究人员比较全面地了解漏洞测试集构建技术及主要进展,为该领域的研究人员提供技术参考,以推动该领域的进一步发展。

到稿日期:2023-03-27　返修日期:2023-07-28
基金项目:国家重点研发计划(2019QY0501)
This work was supported by the National Key Research and Development Program of China(2019QY0501).
通信作者:武泽慧(wuzehui2010@foxmail.com)

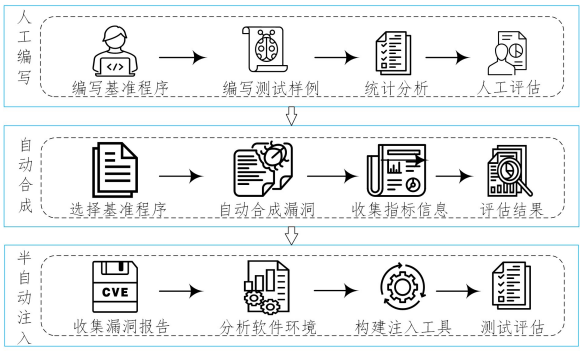


图 1 漏洞测试集发展过程

Fig. 1 Vulnerability benchmark development process

2 漏洞基准测试集分类

不同的基准测试集包含不同的基准程序集和具有不同构造方式的漏洞。本文将基准程序集和漏洞构造方式进行简单的分类，以供漏洞分析工具选择合适的测试集进行基准评估。根据程序集和漏洞来源的真实或合成属性进行分类，并且给出了目前主流基准测试集的所属类别，如表 1 所列。基准测试集中包含的程序可以是真实软件，也可以是合成软件。同样地，其包含的漏洞可以是真实漏洞（人类程序员编写的具有缺陷的代码），也可以是合成漏洞（由漏洞构建程序自动构建的不安全代码）。而表 2 则列出了目前已经开源的漏洞基准测试集分类。

表 1 基准测试集论文分类汇总

Table 1 Summary of vulnerability benchmark paper classification

测试集分类	真实软件基准	合成软件基准
真实漏洞	Magma Juliet SARD CGC logic bombs Fuzzbench Unifuzz Profuzzbench	
合成漏洞	EvilCoder AUTOCTF Apocalypse LAVA ASVG Rode0day BUG-INJECTOR AUTOFACTS SolidFI DRInject SemSeed FIXREVERTER BugAnaBench	CSmith HyperPUT Fuzzle

表 2 现有开源测试集统计

Table 2 Statistics of existing test suites

测试集	年份	语言	漏洞类型	下载地址
Juliet C/C++	2010	C/C++	真实漏洞	https://samate.nist.gov/SARD/test-suites/112
Juliet Java	2010	Java	真实漏洞	https://samate.nist.gov/SARD/test-suites/111
LAVA-M	2016	C/C++	合成漏洞	https://sites.google.com/site/steelix2017/home/lava
CGC	2016	C/C++	合成漏洞	https://github.com/CyberGrandChallenge/
logic-bombs	2017	C/C++	合成漏洞	https://github.com/hxuhack/logic_bombs
Rode0day	2019	C/C++	合成漏洞	https://rode0day.mit.edu/archive
ProFuzzBench	2021	C/C++	真实漏洞	https://github.com/profuzzbench/profuzzbench
Unifuzz	2021	C/C++	真实漏洞	https://github.com/unifuzz
Magma	2021	C/C++	真实漏洞	https://github.com/HexHive/magma
FuzzBench	2021	C/C++	真实漏洞	https://github.com/google/FuzzBench
RevBugBench	2022	C/C++	合成漏洞	https://github.com/SlaterLatiao/RevBugBench
Fuzzle	2022	C/C++	合成漏洞	https://github.com/SoftSec-KAIST/Fuzzle

2.1 真实软件基准

真实软件指由人类程序员设计和实现的程序，可以反映现实世界软件的特征，因此大部分测试集都是基于真实软件进行构建的。

在漏洞基准测试集领域，最早的真实软件程序集工作体现在 Wilander 和 Kamkar 的论文中，2002 年，其提出了包含 44 个 C 函数调用的程序集[4]，以对 5 款漏洞静态分析工具进行评估。2004 年，Zitser 等[5]为了评估缓冲区溢出检测工具，选择了 3 款现实世界广泛使用的软件作为基准程序集。2005 年，NIST（National Institute of Standards and Technology）软件保证指标和工具评估（Software Assurance Metrics And Tool Evaluation，SAMATE）项目[6]花费大量精力收集了 86 864 个 C 和 Java 程序。2010 年，美国国家安全局（National Security Agency，NSA）软件质量中心为评估源码静态分析工具的检测能力，开发了面向 C/C++语言的已知缺陷测试集 Juliet C/C++[7]，该程序集包含 45 324 个 C 程序测试样例，随后经过两次扩充，2018 年，已经超过了 100 000 个文件[8]。作为漏洞分析领域基准评估的初步尝试，Juliet 测试集至今仍在更新。同样由 SAMATE 驱动的软件质量保证参考数据集（Software Assurance Reference Dataset，SARD）项目[9]中包含超过 170 000 个各种语言类型程序文件，其中有超过 7 000 个程序文件来自于十几个流行的基础应用程序。2016 年，国防高级研究计划局（Defense Advanced Research Projects Agency，DARPA）[10]举行了网络大挑战赛（Cyber Grand Challenge，CGC），CGC 中包含 247 个由安全专家精心设计的基准程序集。

尽管上述程序集包含大量的代码文件，但是单个 C 或 Java 程序代码量过小，而且程序结构也过于简单，这使得上述程序集所构成的基准测试集在评估静态检测工具时可以取得良好的评估结果，但是难以有效地评估模糊测试等动态分析工具。因此，研究人员提出了向真实软件中注入漏洞的方法，具体的技术可见本文第 3 章。下面介绍研究人员如何使用复杂真实软件作为基准程序集。

2016 年，Dolan-Gavitt[11]提出了一种基于动态污点分析

的漏洞注入技术 LAVA，他们选取 4 款程序文件构建了基准测试集 LAVA-M。同样地，Pewny 等[12]也创建了类似于 LAVA 的漏洞注入系统，称为 EvilCoder，并尝试使用 EvilCoder 将漏洞引入 libpng，wget 等 4 款开源程序，并在每一个程序中都可以发现数百条关键安全数据路径，这意味着可以插入数百个不同的漏洞。

2017 年，Xu 等[13]设计了 logic bombs 基准测试集用于对基于符号执行的模糊测试工具进行评估。Hulin 等[14]改进了 LAVA 漏洞注入系统并提出了 AUTOCTF，其基准程序集是由他们自己编写的两个程序 blecho 和 fifth。

2018 年，Roy 等[15]提出了 Apocalypse 技术，并选择了 30 款 GNU Coreutils 程序作为基准程序集。相应地，Sridhar[16]则对 LAVA 漏洞注入系统进行了更改，增加了 LAVA 的真实性和多样性，其使用 bufflo 和 bill 两个小程序作为程序集。Yang 等[17]则提出了漏洞构建技术 ASVG，并从 GitHub 中选择了 3 个开源项目作为软件程序集。

2019 年，为了呈现漏洞检测工具的缺陷和不足，由 MIT 的 Fasano 等[18]举办了名为 Rode0day 的新比赛，该比赛使用 9 款真实软件程序作为测试目标。而 Vineeth 等[19]则提出了新的漏洞注入技术 BUG-INJECTOR，并将其引入到 2 款真实软件程序中。为了保证 Openstack[20]等云管理系统的安全性，Cotroneo 等[21]选取 Openstack 的 3 个主要子系统作为基准程序集；基于相同的思想，Marques 等[22]提出了 FIT4Python，同样使用 Openstack 作为程序集。为了评估二进制文件静态分析工具的有效性，Machiry 等[23]提出了 AUTOFACTS，并从 GNU inetutls 中选取了 29 个程序作为程序集。

2020 年，为了评估针对智能合约的静态分析工具的效果，Ghaleb 等[24]提出了漏洞注入工具 SolidFI，他们从 Etherscan[25]库中选取了 50 个智能合约程序作为基准测试程序集。同样地，为了评估并发错误检测技术的效果，Liang 等[26]提出了工具 DRInject，可以自动向软件中注入数据竞争漏洞，并且选择了 10 款软件作为基准程序集。为了能够更加真实准确地评估漏洞检测工具，Ahmad 等提出了 Magma，其包含 7 个真实软件程序集。

2021 年，Google 开发了一个开源基准测试平台 Fuzzbench[27]，为用户提供基准测试服务，目前已经包含 24 款真实软件程序，并且可以轻易从 oss-fuzz[28]上进行扩展。而浙江大学的 Ji 等[29]提出了 Unifuzz 基准测试集，其选取 20 个软件作为基准程序集，并且集成了 35 款主流的模糊测试工具，包括 AFL[30]，Neuzz[31]，QSYM[32]，VUzzer[33]，MOPT[34]等。而 Natella 等[35]发现目前的基准程序集的种类大多集中在多媒体软件、数据压缩工具等无状态的库和软件上，针对有状态软件的程序集几乎没有，于是提出了一个新的基于网络协议状态的基准测试集 Profuzzbench。该程序集提供了一套开源的网络协议程序，Natella 等总共选择了 10 个协议，每个协议都有一个对应的真实软件（FTP 包含了两个软件）。Patra 等[36]提出了 SemSeed，可以向 JavaScript 编写的软件中注入漏洞，并从 GitHub 中选取 100 个流行的项目作为程序集。

2022 年，Zhang 等[37]提出了 FIXREVERTER，增加了注入漏洞的真实性，并使用 10 款开源程序作为程序集。而 Jiang 等[38]则提出了 BugAnaBench，采用 7 款真实软件作为基准程序集。

2.2 合成软件基准

合成软件程序指基于规则、模板或启发式算法等技术生成的基准程序，由于缺乏复杂性和真实性，在目前漏洞基准测试集中很少使用，本小节仅做简单总结。

CSmith[39]是一个程序合成工具，可以通过差分测试[40]来验证评估编译器，从而发现主流编译器框架中存在的安全问题[41-42]，包括 GCC[43]和 LLVM[44]。Kapus 等[45]则使用了基于语法的程序合成以及差分测试技术的组合，用于评估测试符号执行引擎存在的问题。

虽然上述程序可以用于构建基准程序集，但最直接的用处是进行差分测试，例如在差分测试中比较不同版本编译器的区别。而 HyperPUT[46]则继承了上述工具中基于语法程序合成的思想，通过递归的方式将一个初始简单程序转换为复杂的基准程序集。Lee 等[47]则提出了 Fuzzle，使用迷宫合成算法[48]来自动合成基准程序集，使用该算法生成的程序来递归方式链接函数，类似于现代编译器（GNU Compiler Collection，GCC）的递归下降解析器，接近人类编写的代码。

2.3 合成漏洞的构建

通过漏洞构建技术将错误自动植入程序后产生的漏洞被称为合成漏洞，目前合成漏洞已经逐渐成为一种主流的方法，相比人工编写，其具备可扩展性和可移植性，可以批量大规模构建漏洞测试集。给定基准程序集，漏洞程序的构建过程如下：1）首先通过静态分析或动态分析等技术手段确定源代码中可能存在的漏洞埋设位置；2）在每一个相应的位置，对漏洞点进行过滤，验证是否可以合成漏洞以及判断合成漏洞的类型；3）选择某一类型的漏洞，并将其植入到源代码中；4）最后根据测试集构建的评估指标和评估方法检测漏洞检测工具的性能。

前文中有大量采用合成漏洞构建的基准测试集，构建方法具备多样性（详情可见第 3 章）。LAVA 注入的漏洞类型包括缓冲区溢出、数组越界、空指针解引用。EvilCoder 则将注入漏洞类型集中在污点类型的漏洞[49]上，如缓冲区溢出、整数溢出等，这些漏洞的本质是不安全的数据流从用户输入流向敏感输出。AUTOCTF 则在 LAVA 的基础上新构建了两种类型的漏洞，即内存损坏和地址泄露，增加了合成漏洞的可利用性。Apocalypse 则关注如何将漏洞注入到代码更深处，Roy 等通过在程序中嵌入错误转移系统（Error Transition System，ETS）来完成，在注入漏洞类型上则未做过多描述。ASVG 仅仅关注 3 种漏洞类型：缓冲区溢出、数组边界溢出、整数溢出。Rode0day 比赛中使用 LAVA 和 Apocalypse 两种漏洞注入技术，在注入漏洞类型上并没有大的扩展。而 BUG-INJECTOR 则可以通过其在 Common Lisp 中定义的模板注入相应类型的漏洞。

Domenico 等提出的漏洞注入工具主要针对使用 Python 编写的 Openflow 工具。AUTOFACTS 通过修改 LLVM IR

指令合成漏洞。SolidFI 提取了智能合约中常见的 7 种类型漏洞用于构建智能合约静态分析工具的基准测试集。

DRInject 关注的则是并发漏洞测试集的构建，其注入了 600 个数据竞争型漏洞。SemSeed 针对 JavaScript 程序集可以在 1h 内生成超过 10 万个漏洞。FIXREVERTER 在 10 个程序集中通过撤销漏洞修复代码来完成漏洞注入，Zhang 等目前完成了 3 种相关的撤销修复模式。BugAnaBench 则注入了 37 个漏洞，涵盖 5 种漏洞类型，包括堆缓冲区溢出、栈缓冲区溢出等。

2021 年，Brendan 等对 Rode0day 和 LAVA-M 两个基准测试集上的 20 个目标程序进行了大量实验研究工作，发现合成漏洞目前存在以下问题：

1）基于动态分析的漏洞注入方法无法将漏洞注入到未覆盖的代码中。另一方面，无法精确评估注入漏洞是否为可触发的。

2）现有的漏洞注入方法仅支持有限类型的漏洞，这会限制基准测试集对漏洞分析工具的评估，因此添加新类型的漏洞十分重要。

3）注入漏洞与真实漏洞的差异过大，这导致基准测试集的评估结果不能展示出漏洞分析工具的真实能力。

Fuzzle 则改变了向真实软件中注入漏洞的思路，转而通过迷宫合成程序来评估模糊测试工具的路径探索能力，Lee 等在程序出口点设置漏洞，当触发漏洞时表明路径已经探索完成。

随着时间的推移，研究人员意识到合成漏洞必须能模拟真实漏洞环境，以此来真正评估漏洞检测工具对真实漏洞的探索能力，避免出现过拟合问题，当然也有研究人员选择使用真实漏洞作为基准测试集。

2.4 真实漏洞构建

本文将真实漏洞理解为由人类程序手工编写的带有缺陷的代码，真实漏洞相比合成漏洞而言具备真实性，在评估漏洞检测工具时不会出现较大偏差，但是扩展性差。

最早的真实漏洞体现在 Wilander 和 Kamkar 的工作中，其选择了可能触发缓冲区溢出或格式化字符串攻击的 23 个危险函数。Zitser 等则利用开源程序中已有的 14 个可利用缓冲区漏洞进行评估。

由 SAMATE 开发的 SARD 和由 NSA 开发的 Juliet 涵盖漏洞类型众多，包括但不限于缓冲区溢出、格式化字符串问题等 CWE 类型[50]的漏洞，具体详情可从官网获取[51]。

CGC 测试集的每个文件都包含一个或多个漏洞，这些漏洞是由安全专家设计的。美国国防高级研究计划局（Defense Advanced Research Projects Agency，DARPA）之后发布的 CGC 资格赛和决赛中使用的 141 个二进制文件和 17 个代表性示例二进制文件成为了应用广泛的测试集，用于评估各类漏洞检测工具的性能，在 VUzzer，Steelix[52]，Driller[53]等分析工具中有所应用。

logic bombs 的主要目的是评估符号执行工具应对符号推理挑战和路径爆炸挑战的能力，因为一些符号推理挑战会对符号推理的核心过程造成威胁。其作者编写了包括符号变量声明、符号内存等 11 款符号执行挑战测试集，最后对 Klee[54]，Triton[55]，angr[56]这 3 款符号执行工具进行了测试。

Fuzzbench 主要利用开源程序测试动态检测工具的路径探索能力，Profuzzbench 也基于同样的思路，区别在于选取的基准程序类型不同。Unifuzz 则利用开源程序中已有的真实漏洞进行评估，并且提供了一个工具，可以方便地分析 crash 与程序漏洞的对应关系。

Magma 提出了前向移植的方法，可以人工分析二进制程序的历史漏洞及漏洞的补丁程序，然后在最新的版本程序中引入该漏洞，保证漏洞的可利用性和真实性，其漏洞位置和漏洞呈现方式与真实的漏洞接近。由于注入的漏洞是历史上已经存在的，因此将其定义为真实漏洞，其中共移植了 118 个真实漏洞，涵盖数十种漏洞类型。

可以看到，真实漏洞的漏洞类型丰富，可以公平评估漏洞检测工具，避免出现过拟合问题，然而人工编写真实漏洞需要花费研究人员大量的时间和精力。前文的 FIXREVERTER 提出了修复模式撤销，Magma 则提出了前向移植，两者均是解决上述问题的尝试，而 Magma 在真实漏洞引入方面更有效，其构造的环境与真实漏洞并无较大差别。但是两者均需要研究人员对真实漏洞进行大量分析，目前仍停留在半自动化阶段，无法全自动引入真实漏洞。

3 漏洞基准测试集构建方法

本章对目前的漏洞构造技术进行了梳理和归类，并给出了目前测试集的构建流程图，如图 2 所示。另外还整理了目前已经开源的漏洞构造工具，如表 3 所列。

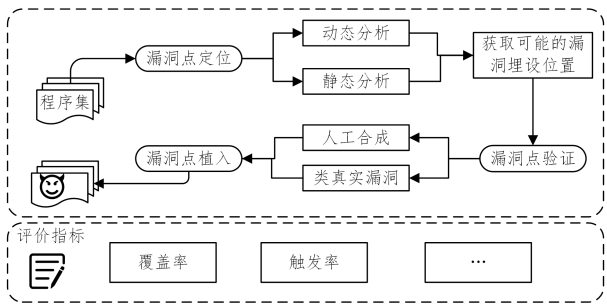


图 2 漏洞测试集构建流程图

Fig. 2 Flowchart of vulnerability benchmark construction

表 3 现有漏洞注入技术统计

Table 3 Statistics of existing vulnerability injection techniques

漏洞注入技术	支持语言	崩溃可触发	可利用	真实软件	真实漏洞
LAVA	C/C++	是	否	是	否
EVILCODER	C/C++	是	是	是	否
AUTOCTF	C/C++	是	是	是	否
APOCALYPSE	C/C++	是	是	是	否
ASVG	C/C++	是	是	是	否
BUG-INJECTOR	C/C++	是	是	是	否
MAGMA	C/C++	是	是	是	是
FUZZLE	C/C++	是	是	是	否
FIXREVERTER	C/C++	是	是	是	否
HyperPUT	C/C++	是	是	是	否
DRINJECT	C/C++	是	是	是	否
SolidFI	Solidity	否	是	是	否
SemSeed	JavaScript	否	是	是	否

3.1 基于污点分析的构建方法

LAVA 的漏洞注入是基于 Panda[57] 动态污点分析框架实现的，该框架建立在 Qemu[58] 模拟器上。通过 Panda 动态分析探索目标程序，定位用户控制的数据流以及潜在的漏洞注入位置，然后将原始代码修改为具有缓冲区溢出漏洞的代码。LAVA 提供了 PoC 用于验证合成漏洞，而且其构造方式具备简单、可控、可触发的优点，但是 LAVA 注入的漏洞具有明显的人工特征，如漏洞中使用“LAVA”命名开头的变量，因此合成的漏洞很容易被漏洞检测工具识别，目前绝大多数漏洞检测工具均可以识别出 LAVA-M 测试集中埋设的漏洞。

EvilCoder 基于静态污点分析技术完成漏洞注入，首先确定注入位置、相关数据流和控制流，然后修改输入数据的安全检查，导致不合规的用户输入可能会到达一些易受攻击的函数上，从而出现污点类型漏洞。例如将”if($a<2$)”修改为“if($a<999$)”，这样不安全的输入到达一些敏感函数时可能会导致错误。因此，该方法不能保证漏洞构造的准确性，仅仅是在系统层面降低了软件的健壮性。

AUTOCTF 基于 LAVA 进行改造，仍然是基于动态分析框架 Panda 实现，其在 LAVA 中添加了嵌入式领域特定语言(Embedded Domain-Specific Language，EDSL)，可以快速进行源代码修改。

3.2 基于符号执行的构建方法

Apocalypse 基于符号执行来分析程序，根据原始程序的上下文，Apocalypse 通过注入额外的程序分支，以便用户输入需要通过更复杂的约束才能到达错误位置。然后，它将函数“*assert*()”注入到错误位置，而不是注入真正的错误。该方法旨在生成适合评价模糊路径探测能力的语料库，它能够提供 PoC 来触发漏洞。

ASVG 基于符号执行收集原始的输入集合，通过源代码程序的结构特点进行分析，定位潜在漏洞可插入点(寻找敏感函数、寻找条件语句、寻找读写函数)，采集程序运行时潜在漏洞点信息，最后修改函数将其转换为漏洞(修改敏感函数变量构造缓冲区溢出、删除或弱化条件语句、调整循环语句边界)，之后进行 PoC 验证。

3.3 基于模式匹配的构建方法

Bug-Injector 提出了一个以漏洞模板、代码融合、动态分析为基础的测试集生成方法，其需要至少一个漏洞用于提取相应的模板，漏洞模板需要以抽象形式描述，并提供基本代码、相关数据类型和注入条件。理论上使用该方法可以构建出多个不同类型的漏洞模板，但实际上生成各种 CWE(Common Weakness Enumeration)类型的良好漏洞模板十分困难，因为其注入条件需要用布尔谓词表达，这会花费研究人员大量的精力。

FIXREVERTER 则是通过观察开源软件的漏洞修复模式后归纳总结出了 3 种常用的漏洞修复模式，这 3 种修复模式都是与条件语句相关的，通过自动匹配并逆转程序中的修复模式达到注入漏洞的目的，但是提取模板是极为复杂的过程。研究人员观察了 6 个开源软件和 Magma 数据集总计 814 个 CVE 才完成漏洞修复模式的总结，使用该方法同样也会消耗研究人员大量的精力。

3.4 基于突变分析的构建方法

Gopinath 等[59] 提出了突变分析的概念，主要用于丰富测试指标，突变分析相对于覆盖率而言不会存在饱和问题而且要比各种形式的覆盖率更加健壮。突变分析的核心思想是将程序转换为一棵抽象语法树，然后选择树的一部分进行变异，产生的变异树再交给编译器测试是否通过，通过多个小变异漏洞的积累转化成大的漏洞。该模型与现实世界的漏洞相当一致，而且突变分析产生的漏洞不会受人类主观性的影响。

Domenico 等结合了突变分析理念，首先将 Python 代码自动转换为抽象语法树，通过扫描抽象语法树寻找可以注入的错误类型，然后通过删除或替换节点来修改抽象语法树，从而引入错误。同样地，SolidFI 首先将智能合约程序转化为抽象语法树，然后通过算法定位可能的漏洞，最后使用基于文本的代码转换修改代码。

3.5 其他构建方法

DRInject 提出的数据竞争漏洞基于动态调试技术进行构建，其由 3 个模块组成：静态分析工具、动态分析工具、代码注入工具。静态分析工具定位全局变量、函数入口点与出口点的位置，并生成可执行代码；动态分析工具调试可执行代码并找到会造成局部变量与全局变量冲突的位置，最后生成数据竞争代码并注入到相应的文件中。SemSeed 通过语义感知的方式构造漏洞，关键思想是通过学习历史补丁的语义模式，构造漏洞以适应上下文环境。

4 漏洞基准测试集的评估与应用

4.1 真实软件基准与合成软件基准实验评估

本小节分析讨论真实软件基准和合成软件基准的不同之处，并从函数平均代码行数(Average Non-Comment Lines of Code，Avg. NLOC)、平均圈复杂度(Average Cyclomatic Complexity Number，Avg. CCN)、函数平均标识符数量(Average token，Avg. token)以及函数数量(Function Count，Fun Cnt)4 个方面分析基准程序的特征，并给出了具体的实验结果，如表 4 所列。

表 4 基准程序复杂度分析

Table 4 Benchmark program complexity analysis

测试集	Avg. NLOC	Avg. CCN	Avg. token	Func Cnt
LAVA-M	34.8	8.9	180.9	47
Rode0day	27.2	9.0	253.7	26018
CGC	21.9	13.1	242.5	23975
RevBugBench	25.0	6.1	151.2	91867
Magma	25.4	6.6	167.9	65861
Fuzzle	16.9	3.5	105.1	153798

1) Avg. NLOC：平均每个函数的代码行数，它可以反映函数的长度和复杂度。

2) Avg. CCN：平均每个函数的圈复杂度，它可以反映代码中的控制结构嵌套层数和条件分支数量。

3) Avg. token：平均每个函数的标识符数，以标识符和关键字为单位进行计数。

4) Func Cnt：函数数量，即代码中函数的数量。

由表 4 可知，在 Avg. NLOC 方面，LAVA-M 的代码行数最长，Fuzzle 的代码行数最短。再分析 Fun Cnt 可知，原因是

Fuzzle 构造的函数数量过多，而在每个函数中的代码则过短，因此，通过对比可知，Fuzzle 合成的程序在程序编写上并不符合人类编写的特征。

在 Avg. CCN 方面，LAVA-M，Rode0day，RevBugBench，Magma 为 6.0～9.0，没有较大的偏差，而 CGC 程序的 CCN 则达到了 13.1，Fuzzle 的 CCN 仅有 3.5。本文认为出现如此大的偏差的原因是 CGC 的软件是由安全专家编写，只是为了评估漏洞分析工具的检测能力，并没有考虑软件的功能和实用性，不需要进行程序优化，而且复杂的控制结构更有利于工具的评估。而 Fuzzle 的基准程序则是由程序自动合成，过于复杂的控制结构必然不利于合成基准程序，因此控制结构简单。

在 Avg. token 方面，Fuzzle 的数量仍然是最少的，这表明 Fuzzle 的基准程序函数复杂度仍与真实程序存在较大差距。

因此，相比真实基准程序，合成基准程序 Fuzzle 在函数平均代码行数、平均圈复杂度、平均函数标识符方面均与真实基准程序不具可比性。

同时，基准程序应具有较强的场景适配性，需要判断其程序集是否可以实现不同的功能。多样的类型可以更好地评估漏洞检测工具的通用性，以及判断其是否可以应对各种场景。2.1 节对测试集使用的基准软件进行了介绍，本小节选取部分测试集进行汇总，表 5 列出了基准测试集具有的基准程序类型的数量。

表 5 基准程序类型统计

Table 5 Statistics of benchmark program types

测试集	基准程序类型数量
LAVA-M	1
Rode0day	9
CGC	—
RevBugBench	9
Magma	7
Fuzzle	—

通过分析可知，Rode0day，RevBugBench，Magma 这 3 个测试集使用具有场景应用的真实基准程序，可以保证其多样性，并具有较强的场景适配性；LAVA-M 使用一个项目的某个单个文件作为基准程序；CGC 由安全专家进行编写，并没有具体的使用场景；Fuzzle 则是合成程序，主要用于评估动态分析工具的路径探索能力。

因此，通过对合成基准程序和真实基准程序在程序复杂性和场景适配性上的讨论发现，相比真实基准程序，合成基准程序存在较大的差异，不能很好模拟真实软件环境，而由人工刻意编写的真实漏洞基准程序场景适配性差，无法进行公正的评估工作。

4.2 真实漏洞与合成漏洞实验评估

本小节讨论了真实漏洞与合成漏洞之间的差异。图 3 给出了各个测试集的漏洞类型，本文通过分析合成漏洞与真实漏洞类型的相似性，来判断其是否可以公平衡量漏洞检测工具的性能。

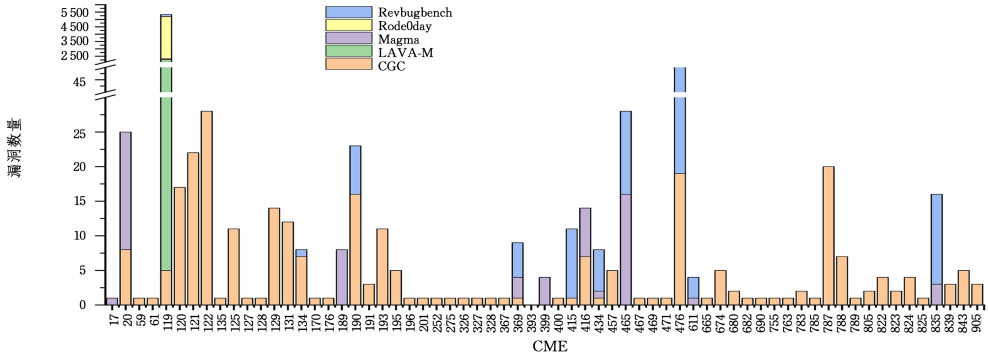


图 3 基准测试集 CWE 漏洞类型分类图

Fig. 3 Categorical CWE vulnerability types in benchmark

可以看到，合成漏洞的 CWE 类型较为单一，如 LAVA-M 和 Rode0day 仅包含 CWE-119（内存缓冲区操作的限制不当）一种类型，虽然漏洞数量庞大，但是不能准确地评估漏洞检测工具对各种类型漏洞的检测能力。因此目前合成漏洞与真实漏洞的区别较大，无法公正评估漏洞检测工具。

同时，本文认为在真实场景的软件程序中，其包含的漏洞类型也会随着适应场景的改变而有所侧重，因此设计了工具 CrawBenCompare，可以获取 CVE 网站中某个软件的所有 CVE 并对其进行 CWE 分类，然后与相应的测试集程序进行拟合性的对比，从而评估基准测试集的有效性。

由于测试集中 libpng 软件出现次数较多，因此本文选取该软件作为例子，该工具可以获取软件历史上所有的 CVE 信息，并与 CWE 建立对应关系，结果如图 4 所示。

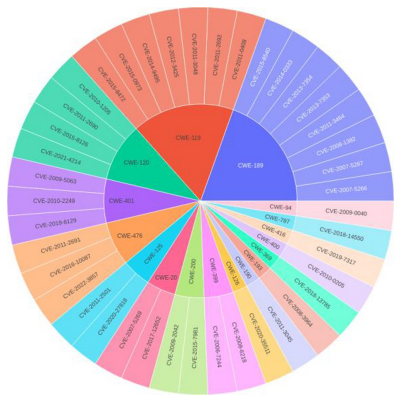


图 4 LibpngCWE 与 CVE 类型分类

Fig. 4 Classification of CWE and CVE types in Libpng

通过将爬取的 CWE 信息与测试集程序漏洞的 CWE 信息进行相似性比对，来量化度量测试集中漏洞构造的真实性，

相似性度量的计算式如下：

$$Cos_{similarity(S_{real},S_{ben})}=\frac{\boldsymbol{S}_{real}\cdot\boldsymbol{S}_{ben}}{\|\boldsymbol{S}_{real}\|\cdot\|\boldsymbol{S}_{ben}\|}$$

$\boldsymbol{S}_{real}$表示的是一组向量，其值是真实软件历史漏洞上各个CWE类型的漏洞数量占所有历史漏洞的比例。同样地，$\boldsymbol{S}_{ben}$则表示测试集中各个CWE类型的漏洞数量占所有漏洞的比例，通过求解两者的余弦相似度来度量测试集漏洞的真实性。

本文选取部分测试集程序进行相似性评估，结果如表6所列。

表6 测试集漏洞相似性评估表

Table 6 Vulnerability benchmark similarity assessment

	Magma	Revbugbench	Rode0day
Libpng	0.248	—	—
Libtiff	0.302	—	—
Libxml2	0.284	0.317	—
openssl	0.186	—	—
php	0.201	—	—
Libyaml	—	—	0.076
Libjpeg	—	—	0.102

由于Magma使用的是历史上的真实漏洞，因此相似性结果优于Rode0day，而Rode0day使用LAVA，Apocalypse漏洞构建技术植入单一的漏洞类型，因此相似性较差。Revbugbench使用Fixreverte构建技术，通过补丁撤销的方法引入漏洞，因此在效果上与Magma没有明显差异。

基准测试集中的漏洞应尽量与该软件的历史真实漏洞保持一致，而目前测试集只有Magma和RevBugBench考虑到这一问题，但是由于两者引入漏洞都需要进行前置分析工作，无法大量构建，因此测试集中的漏洞较为稀疏，得到的相似性评估结果较低。

4.3 基准测试集的应用

基准测试集指标的设计对评估漏洞分析工具起着不可替代的作用[60]，由于不同评估对象的漏洞分析监控方法、数据统计、分析目的等差异较大，本文从静态分析和动态分析两方面阐述基准评估指标设计的研究现状，并梳理出时间脉络，如图5所示。

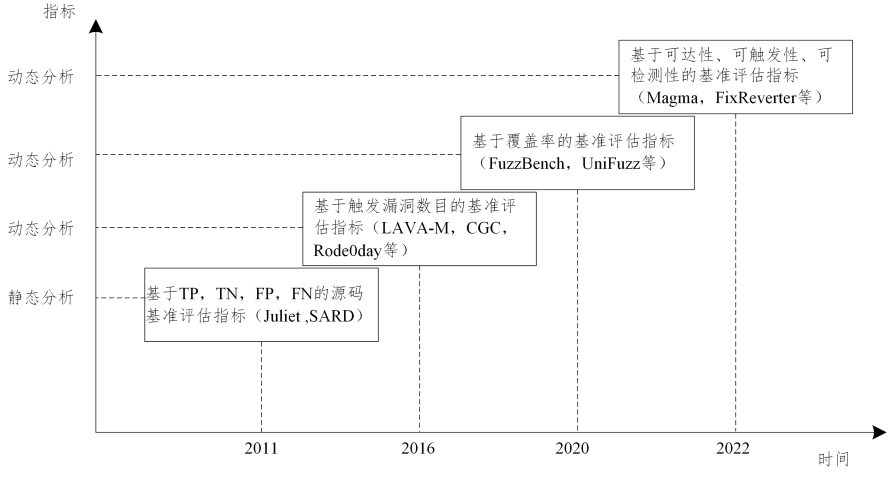


图5 漏洞测试集评估指标汇总

Fig. 5 Summary of vulnerability benchmark evaluation metrics

4.3.1 评估静态分析工具

使用静态分析工具进行评估时，主要通过对比缺陷报告结果和实例的实际数据进行，计算正报个数、误报个数、漏报个数以及真实的负例个数(TN)。表7列出了用来评估静态分析工具有效性的混淆矩阵。

1)真阳数(TP)：被工具报告为有缺陷且实际也有缺陷的个数，即正报个数。

2)假阳例(FP)：被工具报告为有缺陷但实际却是无缺陷的个数，即误报个数。

3)真阴例(TN)：被工具报告为无缺陷且实际也是无缺陷的个数。

4)假阴例(FN)：被工具报告为无缺陷但实际却是有缺陷的个数，即漏报个数。

表7 源代码静态分析指标的混淆矩阵

Table 7 Confusion matrix for source code static analysis metrics

函数类型	真实值	报告值
有缺陷(Bad类函数)	TP	FN
无缺陷(Good类函数)	FP	TN

得到以上4个基础指标后，将计算精确度、召回率、F-score、区分度、覆盖度这5个评价指标，如表8所列。该表列出了静态分析主要使用的测试集及其评估指标的计算式。

表8 静态分析工具使用的测试集及其评估指标

Table 8 Benchmark and evaluation criteria used in static analysis tool

	评估指标	计算式
SARD Juliet	精确度	$Precision=\frac{TP}{TP+FP}$
	召回率	$Recall=\frac{TP}{TP+FN}$
	F-score	$F\text{-}score=2*\left(\frac{Precison*Recall}{Precision+Recall}\right)$
	区分度	$Discrimination\_Rate=\frac{Discriminations}{TP+FN}$
	覆盖度	$Coverage\_CWE=\frac{Tool\_CWEs}{TestSuite\_CWEs}$

最后，表9列出了静态分析工具评估将面临的问题以及回答这些问题需要计算的评估指标。

表 9 工具评估问题及其对应的评估指标

Table 9 Evaluation issues of tools and their corresponding assessment indicators

问题	评估指标
1. 工具可以检测哪些 CWE 类型?	覆盖度
2. 工具检测 CWE 类型占所有 CWE 类型的比例是多少?	召回率和覆盖度
3. 工具报告噪音有多大?	精确度和区分度

通过评估指标解决表 9 中问题的方式如下:

1)可以通过该工具的缺陷报告得到该工具所检测到的缺陷类型。如 CWE 类型可在计算覆盖度的过程中得到。

2)工具检测缺陷占所有缺陷的比例,一方面需要根据召回率来计算基于缺陷个数的比例,另一方面也要根据覆盖度来得到基于缺陷类型的比例。

3)工具缺陷报告的噪音大小可以结合使用精确度和召回率得到。精确度越大,表明误报率越低;精确度越小,则噪音越大,缺陷报告中的误报干扰性越大。区分度可以得到静态分析工具有区分度的正报缺陷占所有缺陷的比例。

4.3.2 评估动态分析工具

由于二进制程序分析过程中,面临数据结构准确还原难、程序内部执行逻辑提取难、运行过程细粒度监控难等问题,难以实现类似源码静态分析方面的详细评价指标。

本节列举了目前模糊测试工具评估指标的研究方向以及模糊器指标构造的几种方式,主要包括对覆盖率统计的细粒度划分,以及细分基准测试程序和 bug 类型对不同领域的模糊测试进行评估等。汇聚结果如表 10 所列。

表 10 动态分析工具使用的测试集及其评估指标

Table 10 Benchmark and evaluation criteria used in dynamic analysis tool

测试集	评估指标
LAVA-M CGC Rode0day	Trigger bug numbers
Unifuzz	Quatity of unique bugs Quality of bugs Speed of finding bugs Stability of finding bugs Edge coverage Overhead
ProFuzzBench	Line coverage Find Line numbers Edge coverage Find Edge Numbers
Fuzzbench	Branch coverage
Magma RevBugBench	Reach bug numbers Triggerbug numbers Detectbug numbers
Logic-bombs Fuzzle	Pass/Fail

LAVA-M,Rode0day,CGC 等通常以触发的漏洞数量来作为验证模糊测试器性能的唯一指标,但是目前人工构造的漏洞基准测试程序真实性不可能到达真实漏洞的标准,合成漏洞要比真实漏洞更容易被发现[61]。

Klees 等[62]评估了 32 款模糊测试器,并研究了它们的实验评价。结果表明,发现漏洞的数量是检验模糊器性能的第一标准,覆盖率可以作为衡量的第二标准,但是选择不同的目标程序,很难比较不同工具或算法间的性能。并且,由于许多模糊测试器并没有统计漏洞而是统计 unique crashes,因此需要考虑 crash 去重的问题[63]。

Profuzzbench,Fuzzbench 依赖单一的覆盖率作为评估指标。Unifuzz 提出了一套更加全面可靠的评估体系,弥补了国内评估体系的空白。Fuzzle 则通过覆盖率、探索时间来衡量模糊测试的路径探索能力。Logic bombs 将符号执行工具是否通过测试样例以及通过测试样例的时间长短作为衡量工具性能的指标。

Wang 等[64]分析了灰盒模糊测试过程中覆盖率度量的影响,目的是研究是否存在其他针对灰盒模糊测试的评估方法。覆盖率度量可以分为两大类,一类是代码覆盖率,另一类是数据覆盖率[65]。他们通过实验分析是否存在某一类型的覆盖率可以完全引导模糊测试发现更多的漏洞,最后得出结论,即不同的覆盖率度量指标可以对应不同的漏洞集[66]。这为以覆盖率为指标的基准测试集提供了新思路,但是以覆盖率为评估指标存在的问题是代码的覆盖率在模糊测试运行一段时间后就会趋于饱和,因此具有局限性。

之后,Magma 对目前的测试集进行了系统性的分析,针对之前工作对漏洞进行计数的粒度不够精细的问题,Magama 将漏洞分为了 3 种状态:reach,trigger,detect。reach 指标可以用来衡量模糊器的路径探索能力,trigger 和 detect 指标则可以用来衡量模糊器的约束求解能力。之后,FIXREVERTER 沿用了 Magma 对漏洞的细粒度划分。

最后,Böhme[67]总结了模糊测试工具在输入、状态性、目标等方面的多样性,发现已有的基准测试集难以评估如此丰富多样的工具。针对这一问题,他们提出的解决方法是细分不同技术领域进行针对性评估,包括基于覆盖的模糊测试技术[68]、基于目标导向的模糊测试技术[69-70]、基于约束求解的模糊测试技术[71-72]等,每个领域又可以按照模糊测试适用的漏洞和软件类型进行进一步的划分[73]。

5 研究挑战与展望

从现有研究来看,围绕漏洞基准测试集的构建、应用与评估已经取得一系列成果。同时,目前该研究还处于发展阶段,漏洞基准测试集仍然难以达到准确评估漏洞分析工具的要求。因此,漏洞测试集的构建与评估技术目前依然面临巨大挑战。

5.1 研究挑战

本小节根据之前相关研究的介绍,对目前面临的问题与挑战进行了总结。

1)合成漏洞测试集存在过拟合问题,真实漏洞测试集存在稀疏性问题。合成漏洞测试集漏洞预置点以及合成漏洞的基准程序形式和结构都过于单一,导致不同测试项在漏洞触发路径、触发条件、代码覆盖等方面等都存在一定的相似性(称为测试集"过拟合问题"),因此更容易被发现。真实漏洞测试集虽然不存在过拟合问题,但是目前的真实漏洞引入方法限制了测试集的规模,导致测试集漏洞过于稀疏,无法全面评估分析工具的性能。

2)缺乏内部测试过程分析和测试指标的体系化设计。

通过文献分析可知，现有测试集在评估中，普遍以“能否检测到漏洞、检测到几个漏洞”等为主，均是判断“是和否”的验证性测试。在测试过程中，则以可观测的现象为主，如测试项是否崩溃、测试过程是否终止、漏洞代码是否执行等，测试者往往忽略了对测试过程的总结和分析。

3）现有漏洞测试集测试策略单一，缺乏场景化设计。漏洞挖掘与软件结构、类别紧密相关，不同基准程序的逆向分析、输入构造、异常监控、代码执行环境等差异较大。在实际测评过程中，现有测试集普遍通过重复测试取平均值，未结合漏洞分析工具特征设计有针对性的测试方法和测试策略；同时缺乏场景化设计，漏洞挖掘过程具有典型的场景性特点，现有方法鲜有基于场景化的测试设计，难以适配到实际应用场景中。

5.2 未来展望

针对基准测试集存在的上述问题进行分析，本文将未来研究的展望总结如下。

1）潜在漏洞点的定位。测试集构建中的漏洞点定位技术所起的作用是返回代码中的可插入漏洞点。在实际的漏洞点定位过程中存在类型单一、误报率高等问题，而引入真实漏洞则需要根据相应漏洞信息对漏洞点进行人工定位。利用符号执行和模式匹配提取漏洞约束模型对漏洞点进行筛选是一种有效的方法，然而如何提取出通用的约束模型作为有效指导，目前仍缺乏探索。

2）漏洞模板的扩充。基于模式匹配的测试集构建方法通常依赖于人工定义好的模板，在漏洞构建阶段，可以在获取的漏洞点上进行逐一尝试，例如撤销一个条件语句、替换一个变量等。但是对于具有复杂逻辑的漏洞并不容易提取模板，因此，理论上，已有的方法只能构建一些简单的漏洞。FIXREVERTER提出的补丁撤销模板用于构建具有真实漏洞特征的漏洞测试集，但是有限的补丁撤销模板是限制测试集漏洞数量的一个重要原因。而利用机器学习方法从漏洞代码或修复代码自动提取高质量漏洞模板或者补丁撤销模板并应用于漏洞构建是值得研究的方向。

3）基准评估过程精细化。对漏洞分析能力进行基准评估，一方面需要获取准确的内部执行数据，另一方面需要从深度、广度等方面设计评价指标，并对评价指标的合理性进行推演证明，确保指标的科学性、客观性。从测试数据统计的准确性和指标设计的全面性两个方面，统一基准能力评估的度量衡。然而，如何设计此类评估指标，是尚未解决的难题。

4）基准测试集程序分类与多指标评估。目前单一类型的基准测试集已经不足以评估现有的漏洞分析工具，因此在使用测试集评估前需要分析漏洞分析工具的特征，得出漏洞分析工具的优势，如擅长约束求解或擅长路径探索，然后根据分析工具的优势，针对性地选取测试程序进行评估。对于漏洞分析工具的衡量标准应该是多维度的，在实际测试过程中如何根据漏洞分析工具的特征智能分配出测试的性能指标是尚未解决的挑战。

结束语 漏洞基准测试集在评估漏洞分析工具时起着不可替代的作用。本文详细阐述了漏洞测试集的构建流程，并且介绍了漏洞构造技术以及面临的挑战。为了有效提高基准测试集评估的可信度与漏洞构建的普遍性，安全研究人员需要设计和实现更加真实的漏洞，需要在测试集中模拟出真实漏洞的效果。虽然目前的研究工作已经积累了较多的研究成果，但仍存在一些问题与挑战，这也成为了该领域未来的研究方向。

参 考 文 献

[1] MANES V J M, HAN H S, HAN C, et al. Fuzzing: Art, science, and engineering[J]. arXiv: 1812.00140, 2018.

[2] BUNDT J, FASANO A, DOLAN-GAVITT B, et al. Evaluating synthetic bugs[C]// Asia Conference on Computer and Communications Security. 2021: 716-730.

[3] HAZIMEH A, HERRERA A, PAYER M. Magma: A ground-truth fuzzing benchmark[J]. ACM on Measurement and Analysis of Computing Systems, 2020, 4(3): 1-29.

[4] WILANDER J, KAMKAR M. A comparison of publicly available tools for static intrusion prevention[C]// Nordic Workshop on Secure IT Systems(NordSec). 2002: 108.

[5] ZITSER M, LIPPMANN R, LEEK T. Testing static analysis tools using exploitable buffer overflows from open source code [C]// ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering. 2004: 97-106.

[6] BLACK P E. Software Assurance Metrics and Tool Evaluation [C]// Software Engineering Research and Practice. 2005: 829-835.

[7] SHRESTHA J. Static Program Analysis [D]. Uppsala: Uppsala University, 2013.

[8] BLACK P E, BLACK P E. Juliet 1.3 test suite: Changes from 1.2[M]. US Department of Commerce, National Institute of Standards and Technology, 2018.

[9] BLACK P E. A software assurance reference dataset: Thousands of programs with known bugs[J]. Journal of research of the National Institute of Standards and Technology, 2018, 123: 1-3.

[10] THOMPSON M F, VIDAS T. Cyber Grand Challenge(CGC) monitor: A vetting system for the DARPA cyber grand challenge[J]. Digital Investigation, 2018, 26: S127-S135.

[11] DOLAN-GAVITT B, HULIN P, KIRDA E, et al. Lava: Large-scale automated vulnerability addition[C]// 2016 IEEE Symposium on Security and Privacy(SP). IEEE, 2016: 110-121.

[12] PEWNY J, HOLZ T. EvilCoder: automated bug insertion[C]// Annual Conference on Computer Security Applications. 2016: 214-225.

[13] XU H, ZHAO Z, ZHOU Y, et al. Benchmarking the capability of symbolic execution tools with logic bombs[J]. IEEE Transactions on Dependable and Secure Computing, 2018, 17(6): 1243-1256.

[14] HULIN P, DAVIS A, SRIDHAR R, et al. AutoCTF: Creating Diverse Pwnables via Automated Bug Injection[C]// WOOT. 2017.

[15] ROY S, PANDEY A, DOLAN-GAVITT B, et al. Bug synthesis: Challenging bug-finding tools with deep faults[C]// European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018: 224-234.

[16] SRIDHAR R. Adding diversity and realism to LAVA, a vulnera-

bility addition system[D]. Massachusetts Institute of Technology,2018.

[17] YANG J,ZHOU P,NI Y. ASVG:Automated Software Vulnerability Sample Generation Technology Based on Source Code [C]// Broadband and Wireless Computing,Communication and Applications (BWCCA-2018). Springer International Publishing,2019:316-325.

[18] FASANO A,LEEK T,DOLAN-GAVITT B,et al. The rode day to less-buggy programs[J]. IEEE Security & Privacy,2019,17(6):84-88.

[19] KASHYAP V,RUCHTI J,KOT L,et al. Automated customized bug-benchmark generation[C]// International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE,2019:103-114.

[20] OpenStack[EB/OL]. (2017-11-26)[2021-08-17]. http://www.openstack.org/.

[21] COTRONEO D,DE SIMONE L,LIGUORI P,et al. How bad can a bug get? an empirical analysis of software failures in the openstack cloud computing platform[C]// ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019:200-211.

[22] MARQUES H,LARANJEIRO N,BERNARDINO J. Injecting software faults in Python applications: The OpenStack case study[J]. Empirical Software Engineering,2022,27(1):20.

[23] MACHIRY A,REDINI N,GUSTAFSON E,et al. Towards automatically generating a sound and complete dataset for evaluating static analysis tools[C]// Workshop on Binary Analysis Research(BAR). 2019.

[24] GHALEB A,PATTABIRAMAN K. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection[C]// ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020:415-427.

[25] BUTERIN V. A next-generation smart contract and decentralized application platform[J]. White Paper,2014,3(37):2-1.

[26] LIANG H,LI M,WANG J. Automated data race bugs addition [C]// European Workshop on Systems Security. 2020:37-42.

[27] METZMAN J,SZEKERES L,SIMON L,et al. Fuzzbench:an open fuzzer benchmarking platform and service[C]// ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021:1393-1403.

[28] SEREBRYANY K. OSS-Fuzz-Google's continuous fuzzing service for open source software[C]// USENIX Security Symposium. USENIX Association,2017.

[29] LI Y,JI S,CHEN Y,et al. UNIFUZZ:A Holistic and Pragmatic Metrics-Driven Platform for Evaluating Fuzzers[C]// USENIX Security Symposium. 2021:2777-2794.

[30] PRAKASH R K,VASUDEVAN I,INDHUJA I,et al. Hardiness sensing for susceptibility using American fuzzy lop[C]// ITM Web of Conferences. EDP Sciences,2021,37:01003.

[31] SHE D,PEI K,EPSTEIN D,et al. Neuzz:Efficient fuzzing with neural program smoothing[C]// 2019 IEEE Symposium on Security and Privacy(SP). IEEE,2019:803-817.

[32] YUN I,LEE S,XU M,et al. {QSYM}:A practical concolic execution engine tailored for hybrid fuzzing[C]// 27th {USENIX} Security Symposium({USENIX} Security 18). 2018:745-761.

[33] RAWAT S,JAIN V,KUMAR A,et al. Vuzzer:Application-aware evolutionary fuzzing[C]// NDSS. 2017,17:1-14.

[34] LYU C,JI S,ZHANG C,et al. MOPT:Optimized Mutation Scheduling for Fuzzers[C]// USENIX Security Symposium. 2019:1949-1966.

[35] NATELLA R,PHAM V T. Profuzzbench:A benchmark for stateful protocol fuzzing[C]// ACM SIGSOFT International Symposium on Software Testing And analysis. 2021:662-665.

[36] PATRA J,PRADEL M. Semantic bug seeding:a learning-based approach for creating realistic bugs[C]// ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021:906-918.

[37] ZHANG Z,PATTERSON Z,HICKS M,et al. {FIXREVERTER}:A Realistic Bug Injection Methodology for Benchmarking Fuzz Testing[C]// 31st USENIX Security Symposium(USENIX Security 22). 2022:3699-3715.

[38] JIANG Z,LI R,TANG C. BugAnaBench:benchmark for software vulnerability analysis and its construction method[C]// Second International Symposium on Computer Technology and Information Science(ISCTIS 2022). SPIE,2022:40-45.

[39] YANG X,CHEN Y,EIDE E,et al. Finding and understanding bugs in C compilers[C]// ACM SIGPLAN Conference on Programming Language Design and Implementation. 2011:283-294.

[40] MCKEEMAN W M. Differential testing for software[J]. Digital Technical Journal,1998,10(1):100-107.

[41] MARCOZZI M,TANG Q,DONALDSON A F,et al. Compiler fuzzing:How much does it matter? [C]// Proceedings of the ACM on Programming Languages. 2019,3(OOPSLA):1-29.

[42] EVEN-MENDOZA K,CADAR C,DONALDSON A F. Closer to the edge:Testing compilers more thoroughly by being less conservative about undefined behaviour[C]// IEEE/ACM International Conference on Automated Software Engineering. 2020:1219-1223.

[43] POP A,POP S,JAGASIA H,et al. Improving GNU compiler collection infrastructure for streamization[C]// Proceedings of the 2008 GCC Developers' Summit. 2008:77-86.

[44] LATTNER C,ADVE V. LLVM:A compilation framework for lifelong program analysis & transformation[C]// International Symposium on Code Generation and Optimization,2004(CGO 2004). IEEE,2004:75-86.

[45] KAPUS T,CADAR C. Automatic testing of symbolic execution engines via program generation and differential testing[C]// 2017 32nd IEEE/ACM International Conference on Automated Software Engineering(ASE). IEEE,2017:590-600.

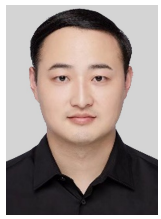
[46] FELICI R,POZZI L,FURIA C A. HyperPUT:Generating Synthetic Faulty Programs to Challenge Bug-Finding Tools[J]. arXiv:2209.06615,2022.

[47] LEE H,KIM S,CHA S K. Fuzzle:Making a Puzzle for Fuzzers [C]// 37th IEEE/ACM International Conference on Automated Software Engineering. 2022:1-12.

[48] SHAH M S H,MOHITE M J M,MUSALE A G,et al. Survey paper on maze generation algorithms for puzzle solving games [J]. International Journal of Scientific & Engineering Research,2017,8(2):1064-1067.

[49] YAMAGUCHI F, GOLDE N, ARP D, et al. Modeling and discovering vulnerabilities with code property graphs[C]// 2014 IEEE Symposium on Security and Privacy. IEEE, 2014: 590-604.
[50] MARTIN R A, BARNUM S. Common weakness enumeration (cwe) status update[J]. ACM SIGAda Ada Letters, 2008, 28(1): 88-91.
[51] NIST Software Assurance Reference Dataset Project[EB/OL]. https://samate.nist.gov/SRD/.
[52] LI Y, CHEN B, CHANDRAMOHAN M, et al. Steelix: program-state based binary fuzzing[C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017: 627-637.
[53] STEPHENS N, GROSEN J, SALLS C, et al. Driller: Augmenting fuzzing through selective symbolic execution[C]// NDSS. 2016, 16(2016): 1-16.
[54] CADAR C, DUNBAR D, ENGLER D R. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs[C]// OSDI. 2008, 8: 209-224.
[55] SAUDEL F, SALWAN J. Triton: Concolic execution framework [C]// Symposium Sur La sécurité Des Technologies De l'Information ET DES Communications(SSTIC). 2015.
[56] SPRINGER J, FENG W. Teaching with angr: A symbolic execution curriculum and {CTF}[C]// 2018 {USENIX} Workshop on Advances in Security Education({ASE} 18). 2018.
[57] DOLAN-GAVITT B, HODOSH J, HULIN P, et al. Repeatable reverse engineering with PANDA[C]// Program Protection and Reverse Engineering Workshop. 2015: 1-11.
[58] DOLAN-GAVITT B, HODOSH J, HULIN P, et al. Repeatable reverse engineering with PANDA[C]// Program Protection and Reverse Engineering Workshop. 2015: 1-11.
[59] GOPINATH R, GÖRZ P, GROCE A. Mutation analysis: Answering the fuzzing challenge[J]. arXiv: 2201.11303, 2022.
[60] ZHIVICH M, LEEK T, LIPPMANN R. Dynamic buffer overflow detection[C]// Workshop on the Evaluation of Software Defect Detection Tools. 2005.
[61] LUK C K, COHN R, MUTH R, et al. Pin: building customized program analysis tools with dynamic instrumentation[J]. ACM Sigplan Notices, 2005, 40(6): 190-200.
[62] KLEES G, RUEF A, COOPER B, et al. Evaluating fuzz testing [C]// ACM SIGSAC Conference on Computer and Communications Security. 2018: 2123-2138.
[63] GAO X, MECHTAEV S, Crash-avoiding program repair[C]// Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2019: 8-18.
[64] WANG J, DUAN Y, SONG W, et al. Be sensitive and collaborative: Analyzing impact of coverage metrics in greybox fuzzing [C]// Research in Attacks, Intrusions and Defenses(RAID'19). 2019.
[65] CADAR C, DUNBAR D, ENGLER D R. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs[C]// OSDI. 2008, 8: 209-224.
[66] LIN G, WEN S, HAN Q L, et al. Software vulnerability detection using deep neural networks: a survey[J]. Proceedings of the IEEE, 2020, 108(10): 1825-1848.
[67] BÖHME M, CADAR C, ROYCHOUDHURY A. Fuzzing: Challenges and reflections[J]. IEEE Software, 2020, 38(3): 79-86.
[68] BÖHME M, PHAM V T, NGUYEN M D, et al. Directed greybox fuzzing[C]// ACM SIGSAC Conference on Computer and Communications Security. 2017: 2329-2344.
[69] WANG P, ZHOU X, LU K, et al. The progress, challenges, and perspectives of directed greybox fuzzing[J]. arXiv: 2005.11907, 2020.
[70] ZHANG Z, CHEN L, WEI H, et al. Binary-level Directed Symbolic Execution Through Pattern Learning[C]// 2022 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). IEEE, 2022: 50-57.
[71] BALDONI R, COPPA E, D'ELIA D C, ET AL. et al. A survey of symbolic execution techniques[J]. ACM Computing Surveys (CSUR), 2018, 51(3): 1-39.
[72] ZHANG T, JIANG Y, GUO R, et al. A survey of hybrid fuzzing based on symbolic execution[C]// Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies. 2020: 192-196.
[73] ZHU X, WEN S, CAMTEPE S, et al. Fuzzing: a survey for roadmap[J]. ACM Computing Surveys (CSUR), 2022, 54(11S): 1-36.

MA Zongshuai, born in 1999, postgraduate. His main research interest is software security analysis.



WU Zehui, born in 1988, Ph. D. His main research interest is software and system vulnerability analysis.

(责任编辑:喻藜)