无碰撞灰盒模糊测试方法研究

王　松，方　勇，贾　鹏
(四川大学网络空间安全学院，成都 610065)

摘　要：灰盒模糊测试技术已被证实是一种高效实用的漏洞挖掘技术，在漏洞挖掘领域应用广泛，有大量的高危漏洞都是通过灰盒模糊测试找到的. AFL 是灰盒模糊测试的经典代表之作，大量后续灰盒模糊测试都是在 AFL 的基础上根据不同条件进行改进得到的，可以说 AFL 是主流灰盒模糊测试的奠基之作. 但是 AFL 仍然存在一些问题，AFL 在对目标待测程序进行插桩时采用随机数代表桩点，在测试过程中采用两个桩点的随机数进行异或运算，得到结果用来表示一条边. 这种方式使得在进行边的统计时就会出现 HASH 碰撞问题，导致有一定的概率新边无法被发现，从而影响 AFL 的漏洞挖掘效率. 这个问题会随着待测程序的代码规模变大而显得愈发突出. 本文通过改进汇编级插桩的方式，将基本块敏感的插桩改为分支敏感的插桩，从而将程序的控制流图改变为二叉树的形式，并采用非随机编码来标记各个桩点，较好地解决了 HASH 碰撞的问题. 实验证明该方法有效，且由于该改进对于上层是透明的，可以应用于各个基于 AFL 的灰盒模糊测试工具中，从而提高模糊测试的效率.

关键词：灰盒模糊测试；AFL；插桩；HASH 碰撞；基本块；分支敏感
中图分类号：TP391.1　　**文献标识码**：A　　DOI：10.19907/j.0490-6756.2023.033004

Research on Collision-Free grey box fuzzing method

WANG Song, *FANG Yong*, *JIA Peng*
(School of Cyber Science and Engineering, Sichuan University, Chengdu 610065, China)

Abstract: Grey-box fuzzing technology has been proved to be an efficient and practical vulnerability mining technology. It is widely used in the field of vulnerability mining, and many high-risk vulnerabilities are found through grey-box fuzzing. American Fuzzy Lop(AFL) is a classic representative of grey-box fuzzing and many subsequent grey-box fuzzing are improved on the basis of AFL according to different conditions but AFL still faces certain issues. AFL uses random numbers to represent instrumentation points when performing instrumentation on target program, the random numbers of the two instrumentation points are used to perform the XOR operation in the testing process, and the result is used to represent an edge. This method can lead to HASH collision problems when performing edge statistics, which decreases the probability of discovering new edges and affects AFL's vulnerability mining efficiency, especially for larger code sizes.. In this paper, by improving the way of assembly-level instrumentation, the basic block-sensitive instrumentation is changed to branch-sensitive instrumentation, so that the control flow graph of the program is changed into a binary tree form, and non-random numbers are used to mark each instrumentation point, which is relatively well solved the problem of HASH collision. Experiments show that the propsosed method is effective, and since the improvement is transparent to

收稿日期：2023-01-06
作者简介：王松(1988—)，男，四川眉山人，硕士研究生，研究方向为网络信息对抗. E-mail:271974754@qq.com
通讯作者：贾鹏. E-mail：pengjia@scu.edu.cn

the upper layer, it can be applied to various AFL-based grey-box fuzzing tools, thereby improving the efficiency of the fuzzing test.

Keywords: Grey-box fuzzing; AFL; Instrumentation; HASH collision; Basic block; Branch sensitive

1　引　言

在漏洞挖掘领域,灰盒模糊测试[1](Greybox Fuzzing)是一种最具可扩展性和实用性的方法. 它是介于黑盒和白盒之间的一种更有效的模糊测试. 由于黑盒模糊测试缺乏对目标待测程序的分析,测试用例的生成过于盲目,灰盒模糊测试比黑盒模糊测试更具有效性;灰盒模糊测试对目标待测程序的分析相对于白盒模糊测试来说是轻量级的,它比白盒模糊测试有更高的效率[2-7]. 灰盒模糊测试广泛应用于大量软件测试、库以及内核代码的检测. 有大量的高危漏洞都是通过灰盒模糊测试找到的. 截止到 2022 年 7 月,谷歌的 OSS-Fuzz 平台利用几款灰盒模糊测试工具如 libFuzzer[8]、Honggfuzz[9]、American Fuzzy Lop(AFL)[10]和 AFL++[11]发现了 650 款开源软件中的超过 40 000 个漏洞[12].

灰盒模糊测试的基本过程是将初始种子文件按照一定的策略进行变异后得到大量程序输入,并将这些输入放到目标待测程序中运行,同时跟踪记录并反馈程序的运行状态信息来指导后续的测试,最后对引起程序崩溃的输入进行分析,从而找到程序的漏洞.

AFL 是灰盒模糊测试的经典代表之作,其实用性得到了广大学者的青睐. 大量后续的灰盒模糊测试都是基于 AFL 在不同的条件下改进得到的,如 AFLGO[13]、AFLFast[14]、AFL++等. 可以说,AFL 是主流灰盒模糊测试的奠基之作.

但是 AFL 这类主流 Fuzzer 仍然存在一些不可忽视的问题,例如 HASH 碰撞问题[15]. HASH 碰撞问题会导致待测程序在被测试中反馈的覆盖率信息有错,进一步导致有价值的测试用例被抛弃. 这个问题将随着程序代码规模的变大而变得更加严重.

本文将采用改进插桩代码的方式,通过新的统计方法来解决 HASH 碰撞问题,实现待测程序的覆盖率提高,从而提高漏洞挖掘的效率.

2　背　景

2.1　AFL 简介

以 AFL 为代表的主流灰盒模糊测试大多都是基于边覆盖率引导来进行测试的,即 CGF(Coverage-guide Greybox Fuzzing). CGF 一般会在待测程序源代码中进行插桩(桩代码),从而可以在每一次测试程序时获得程序路径覆盖率信息的反馈,并对出现过的程序路径信息进行记录汇总,产生总的程序路径信息位图(bitmap). 用单个测试用例的覆盖程序路径信息和总的 bitmap 进行对比,通过覆盖率是否提高进行判断,筛选出有价值的测试用例,再赋予不同的能量(即变异时间)对有价值的测试用例进行重点测试,以提高模糊测试的效率. 图 1为 AFL 的工作流程图.

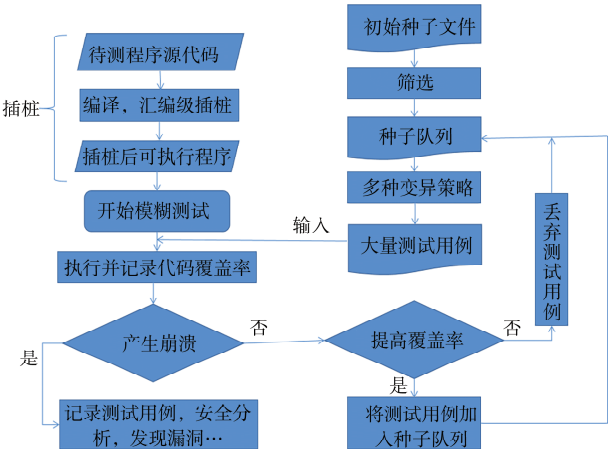


图 1　AFL 工作流程图

Fig. 1　AFL workflow chart

这样做的目的是在有限的时间内覆盖尽可能多的程序路径. 基于覆盖率引导的测试思想对后续研究有着深远的影响. 其原理和核心思想是基于被测试到的程序代码不一定能发现漏洞,但是没被测试到的程序代码肯定不能发现漏洞. 所以在测试过程中要尽量去发现新的程序路径(即新边),有新边出现的测试用例将被认定为好的种子,获得更多的变异可能和测试时间,从而增大发现漏洞的概率.

2.2　HASH 碰撞问题

HASH 碰撞问题会导致待测程序在被测试后,不同的边被视为相同的边. 这样将会有一定的概率导致新边被误认为是旧边,进一步导致测试用例被抛弃. 这和 AFL 的核心思想是相矛盾的——命中了新边的测试用例反而被抛弃将大大影响漏洞的挖掘效率.

对这个问题有很多学者都进行了相关研究，例如 BigMap[16]、INSTRIM[17]、CollAFL[18] 等. 有的是采用扩大内存的方法，有的是采用改变边覆盖为基本块覆盖的方法，本文将采用改进汇编级插桩的方式并通过新的统计方法来尝试解决这个问题.

2.2.1　问题成因　AFL 在对待测程序进行插桩时会在每个插桩点放置一个随机数（范围：0～65 535），在测试过程中使用前一个桩点随机数右移一位和当前桩点随机数的异或值来表示这条程序边，并对每条边的命中次数进行统计，结果放入 trace_bits 数组中，用以反馈边覆盖率信息，过程示意如下.

CurBlock_Num=Random(0, 65 535)

Edge=(PreBlockNum>>1)^CurBlock_Num

Trace_bits[edge]++

由于异或操作造成了桩点信息损失（且随机数本身就有一定的可能直接相同），即 HASH 碰撞问题，例如：

Block_A = 0×02　Block_B = 0×011

$Edge_{AB}$=(0×02>>1)^0×11=0×10

Block_C = 0×22 Block_D = 0×01

$Edge_{CD}$=(0×22>>1)^0×01=0×10

AB 和 CD 是完全不同的两条程序边，但是在统计命中次数时会被视为等同，产生了 HASH 碰撞.

2.2.2　问题影响　在程序规模较小时，这个问题影响不大，但是它会随着程序规模的增加而变得越来越严重. 在对不同边规模的程序进行 100 次重复模拟碰撞测试后，取平均值得到的程序边规模和总体碰撞率的关系如表 1 所示.

程序边规模为 N 时，总体碰撞率计算公式经过本文归纳总结可表示为

$$P=\frac{\sum_{i=2}^{N}(|R_i|-|R_{i-1}|)}{N} \tag{1}$$

其中，$R_i=\{r_1, r_2, \cdots\cdots, r_i\}$；$r_i$ 是代表每条边的随机数.

在模糊测试过程中，对靠后的测试用例进行测试时，由于已经产生了很多旧边，若此时再有新边出现，单条新边的碰撞率会更高，如表 2 所示.

单条新边在程序边规模为 N 时碰撞概率公式为

$$P=\frac{1}{N} \tag{2}$$

较高的单条新边碰撞概率，会造成模糊测试越往后进行越难以发现有价值的测试用例（即命中新边的测试用例），这与边覆盖率引导测试的核心思想是相违背的.

表 1　程序规模和总体碰撞率关系

Tab. 1　Relationship between program size and overall collision rate

程序边规模	碰撞次数	碰撞概率/%
1000	7.69	0.77
2000	30.54	1.54
5000	186.99	3.74
10 000	723.14	7.23
20 000	2764.67	13.82
30 000	5946.41	19.82
40 000	10 053.86	25.13
50 000	15 019.37	30.04

表 2　程序规模和单新边碰撞率关系

Tab. 2　Relationship between program size and single new edge collision rate

已出现程序边规模	单新边碰撞概率/%
1000	1.53
2000	3.05
5000	7.63
10 000	15.26
20 000	30.52
30 000	45.78
40 000	61.04
50 000	76.30

3　优化方法

如果完整的记录起点终点的随机数，不做异或操作，则可以避免 HASH 碰撞问题. 但是这样做会造成记录边信息的存储空间爆炸. 当完整地记录每条边的起点终点随机数用以区分每条边时，要考虑 65 536×65 536 种可能，共享内存为 2^{32} byte，开销太大. 共享内存过大会造成寻址速度大大降低，进行 bitmap 比对时也会耗费更多的时间，将严重影响漏洞挖掘的效率. 如图 2 所示.

但是通过分析可以注意到每一个桩点的跳转可能性是有限的，即以每一个桩点为起点的边的数目是有限可控的. 基于此点，可以尝试找到一种方法适当增大共享内存，以较小代价解决 HASH 碰

撞问题.

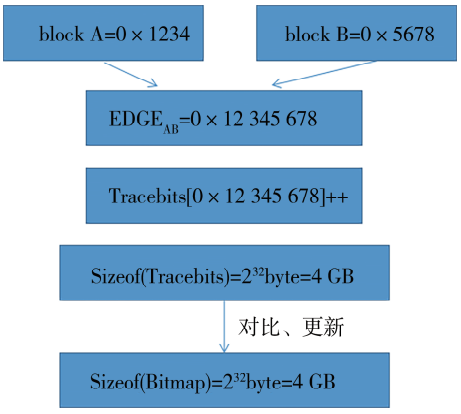


图 2　简单取消异或所需空间示意图

Fig. 2　Schematic diagram of the space required for simple cancellation of XOR

3.1　基于分支敏感的插桩

以 AFL 为代表的灰盒模糊测试一般采用的插桩策略是以基本块为最小单位的,这样就会在各基本块的起始位置进行插桩,例如在函数头、各跳转分支点、标号进行插桩并产生程序的边信息,如图 3 所示.

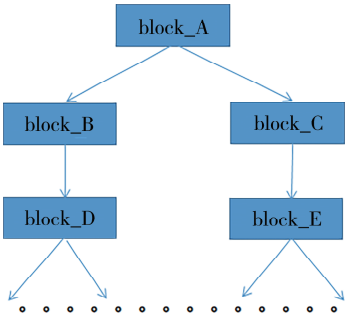


图 3　基本块敏感的插桩

Fig. 3　Basicblock sensitive instrumention

这样做的坏处是会产生一些多余的边. 可以将它改进为基于分支敏感的插桩策略,取消必经点的插桩,因为测试过程只对程序的不同路径感兴趣,如图 4 所示. 通过对比,可以看到相同的程序结构,在必经点(B、C 都是必经点)处不再插桩后,程序边数有所减少,这对漏洞挖掘工作是有利的.

采用基于分支敏感的插桩策略后,在插桩时,不再对函数头这类必经点进行插桩,找到分支点(跳转指令和标号)进行插桩即可. 由于实现的是汇编级插桩,根据汇编语言的特点,例如 jz 等语句产生的后续分支都只有两种可能,这样得到的程序所有分支点构成一个二叉树结构,即每一个桩点的后续节点只有两种可能,这对后续的统计方法改进有着至关重要的作用,可以避免路径爆炸问题,如图 5所示.

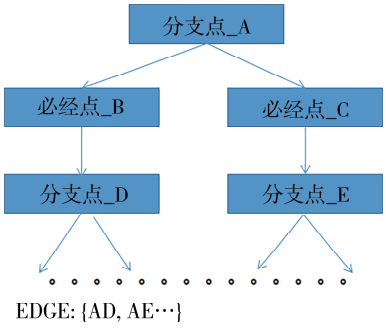


图 4　分支敏感的插桩

Fig. 4　Branch sensitive instrumention

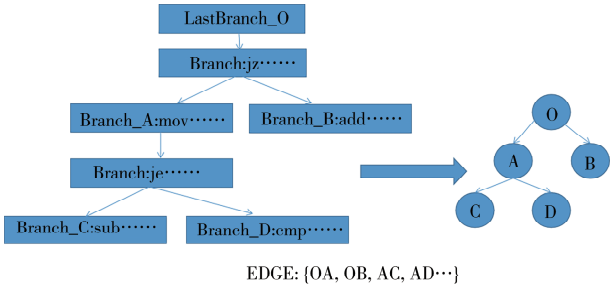


图 5　基于分支敏感的二叉树结构

Fig. 5　Branch sensitive binary tree structure

如果不取消函数头这类必经点的插桩,则可能出现一个函数跳转到多个函数的情况(即一个桩点的后续桩点有多种可能),造成程序边路径爆炸,这种情况下的桩点结构远比基于分支敏感的桩点结构要复杂得多,将导致记录边信息需要的内存过大而无法实现有效改进,如图 6 所示.

3.2　分支节点顺序编码

采用顺序编码对各分支点进行插桩标记,即从 1 开始依次递增标记每个桩点. 这样做可以避免随机数生成时造成的随机数相同而导致的碰撞(两个不同的桩点随机数相同,进而导致边表示可能相同),且各个桩点的规律性和可辨识度更高.

3.3　扩大共享内存并改变统计方法

采用基于分支敏感的插桩策略后,程序插桩点是二叉树结构,即单一桩点的后继节点只有两种可能. 可以通过扩大一定的内存来记录一条程序边的起点及终点的完整信息,而不再采用异或的方式来表示边.

将记录程序边命中次数的 64 kB 的 trace_bits

内存扩大为 128 kB，即边的可能性由 65 536 条变为 65 536×2 条. 再增加维护另一个数组 edgeinformation，即边的信息数组，大小为 65 536×2×2 byte=256 kB，此时共享内存大小为 384 kB. 在统计时记录桩点的后继节点信息用以区分两条边的不同，如图 7 所示.

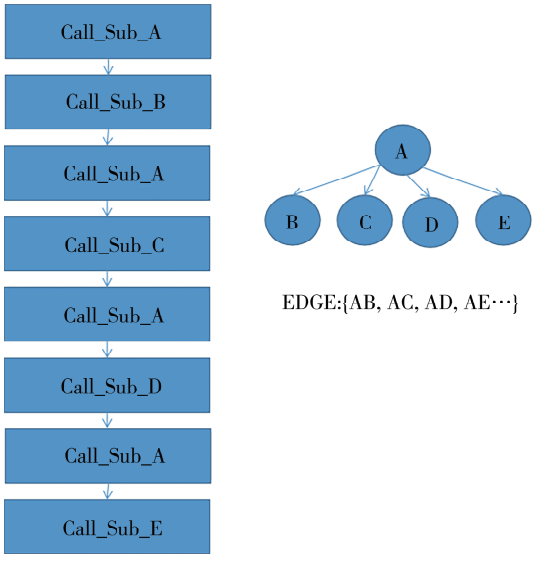


图 6　函数头桩点造成路径爆炸

Fig. 6　Function header instrumention causes path explosion

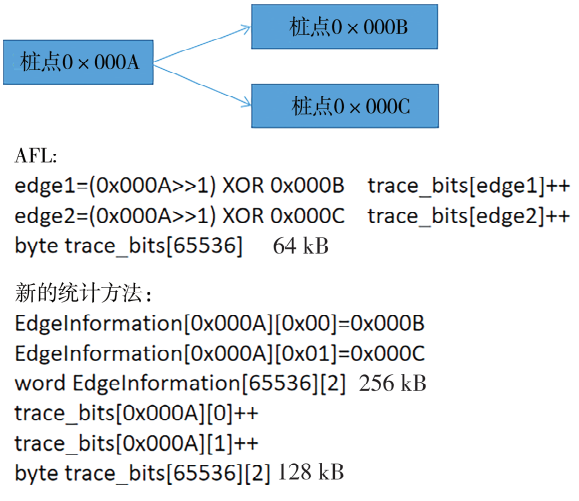


图 7　新旧统计方法对比

Fig. 7　Comparison of old and new statistical methods

因为原始方法的边命中次数数组（trace_bits）地址即代表边的有序排列，所以在对比边命中信息时只需要依次按字节进行对比即可. 改进后的方法则需要根据记录的边的信息将地图稍做调整即可（即将二叉树的左右后继节点位置调整得相同）.

改进后的插桩统计函数以汇编代码实现在桩代码中，其流程如图 8 所示.

```
Instrument Code
1.  前一桩点为 Prebranch_Num  当前桩点为Curbranch_Num
2.  IF          Edge=Prebranch_Num-XXX从未出现
3.  Then        记录Curbranch_Num为Prebranch_Num的左后继节点
4.              Edge=Prebranch_Num-Curbranch_Num命中次数加1
5.  Else IF     Curbranch_Num是Prebranch_Num的左后继节点
6.              Edge=Prebranch_Num-Curbranch_Num命中次数加1
7.  Else        Curbranch_Num必然是Prebranch_Num的右后继节点
8.              记录Curbranch_Num为Prebranch_Num的右后继节点
9.              Edge=Prebranch_Num-Curbranch_Num命中次数加1
```

图 8　统计函数流程伪代码

Fig. 8　Statistical function flow pseudo code

4　实验及分析

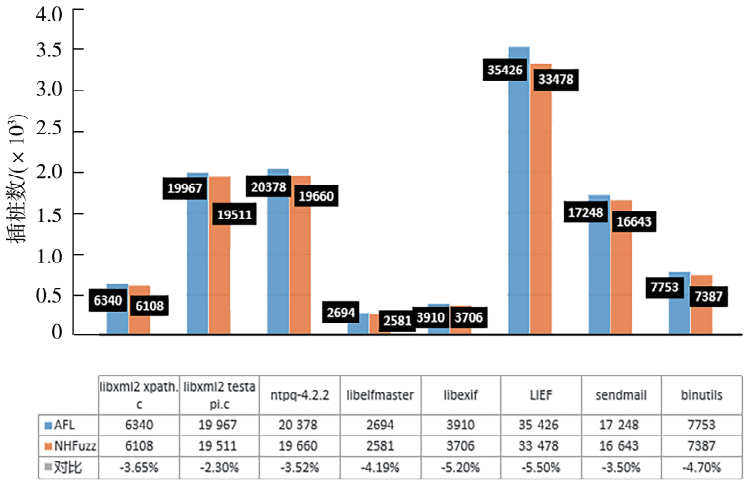
本文在 AFL2.52b 的基础上实现了模糊测试工具 NHFuzzer，并进行了实验对比. 实验选取 libxml2、ntpq、libelfmaster、LIEF、sendmail、binutils 和 libexif 作为目标，实验主机 CPU 型号为 Intel Xeon CPU E3-1231 v3@3.4 GHz，系统版本为 Ubuntu18.04.5. 每次实验运行 24 h，分别运行 3 次后取平均值得到实验结果.

4.1　插桩数对比

如图 9 所示，各个测试对象的插桩数都有明确减少但是减少幅度不大，都在 6% 以内，因为函数头类必经点占比规模不大. 插桩数减少可以小幅度减少待测程序的边规模，但是去掉函数头类必经点的最大意义是使得汇编级的二叉树分支图得以呈现，后续的边信息可以得以全存储.

4.2　覆盖率、有价值种子数及崩溃数对比

如表 3 所示，通过实验可以看到 NHFuzz 使得待测试程序的边覆盖率、有价值的种子数以及独特崩溃数三个主要指标都得到了有效提升. 在 libexif、LIEF 和 senmail 中，边覆盖率、有价值的种子数和独特崩溃数三个指标都提升了 10% 以上，提升最为明显. 因为随着程序规模的增大，HASH 碰撞问题会更加严重，效率提升空间就更大，这和之前的问题分析也是相符合的. 从总体实验结果来看，本文提出的基于分支敏感的方法通过解决 HASH 碰撞问题来提高程序覆盖率，进而提高灰盒模糊测试效率是有效的.



	libxml2 xpath.c	libxml2 testapi.c	ntpq-4.2.2	libelfmaster	libexif	LIEF	sendmail	binutils
AFL	6340	19 967	20 378	2694	3910	35 426	17 248	7753
NHFuzz	6108	19 511	19 660	2581	3706	33 478	16 643	7387
对比	-3.65%	-2.30%	-3.52%	-4.19%	-5.20%	-5.50%	-3.50%	-4.70%

图 9　插桩数对比

Fig. 9　Comparison of instrumention number

表 3　实验结果对比

Tab. 3　Comparison of experimental results

测试对象	工具	覆盖程序边数	对比	有价值的种子数	对比	独特崩溃数	对比
Libelfmaster elfparse. c	AFL	1993	+3. 2%	771	+8. 1%	216	+2. 3%
	NHFuzz	2058		834		221	
Libxml2 paser. c	AFL	5027	+5. 2%	3173	+4. 5%	25	+16%
	NHFuzz	5291		3316		29	
Libxml2 xmllint. c	AFL	6916	+6. 1%	4107	+5. 1%	198	+9. 1%
	NHFuzz	7335		4318		216	
Binutils 2. 30 nm-new	AFL	7553	+8. 3%	1543	+9. 6%	7	+14. 3%
	NHFuzz	8182		1692		8	
Ntpq 4. 2. 2	AFL	10 485	+9. 7%	379	+8. 9%	46	+15. 2%
	NHFuzz	11 509		413		53	
Libexif 0. 6. 22	AFL	17 039	+16. 3%	5846	+14. 5%	95	+18. 9%
	NHFuzz	19 817		6697		113	
LIEF 0. 12. 3	AFL	18 539	+14. 8%	3367	+15. 1%	127	+14. 1%
	NHFuzz	21 284		3874		145	
Sendmail 8. 8. 0	AFL	16 884	+13. 0%	137	+13. 8%	13	+23%
	NHFuzz	19 078		156		16	

5　结　论

针对 AFL 为代表的传统灰盒模糊测试中存在的 HASH 碰撞问题，本文对问题成因及影响进行了细致的分析，并提出了基于分支敏感的插桩及新的统计方法，并实现了模糊测试工具 NHFuzzer. 实验证明，NHFuzzer 在提高代码覆盖率，提升漏洞挖掘效率上是行之有效的，且实验表明该方法在大规模程序的测试中效果更好. 下一步考虑研究在必经点的分析提取上做相关工作进一步提升模糊测试的效率.

参考文献：

[1]　Li J, Zhao B, Zhang C. Fuzzing: a survey [J]. Cy-

bersecurity，2018，1：1.

[2] Chen C，Cui B，Ma J，*et al*. A systematic review of fuzzing techniques [J]. Comput Secur，2018，75：118.

[3] Cadar C，Dunbar D，Engler D R. Klee：unassisted and automatic generation of high-coverage tests for complex systems programs [C]//Usenix Conference on Operating Systems Design and Implementation. San Diego：USENIX Association，2008：209.

[4] Ahmed A，Mishra P. QUEBS：qualifying event based search in concolic testing for validation of RTL models [C]//International Conference on Computer Design (ICCD). Boston：IEEE，2017：185.

[5] Lyu Y，Ahmed A，Mishra P. Automated activation of multiple targets in rtl models using concolic testing [C]//2019 Design，Automation & Test in Europe Conference & Exhibition (DATE). Florence：IEEE，2019：354.

[6] Ahmed A，Farahmandi F，Mishra P. Directed test generation using concolic testing on rtl models [C]//2018 Design，Automation & Test in Europe Conference & Exhibition (DATE). Dresden：IEEE，2018：1538.

[7] Ren Z，Zheng H，Zhang J，*et al*. A review of fuzzing techniques [J]. Comput Resnd Dev，2021，58：944.

[8] Serebryany K. Continuous fuzzing with libfuzzer and addresssanitizer [C]//Cybersecurity Development. Boston：IEEE，2016：157.

[9] Swiecki R. Honggfuzz：a general-purpose，easy-to-use fuzzer with interesting analysis options [EB/OL]. [2022-10-17]. https://github.com/google/honggfuzz.

[10] Zalewski M. American fuzzy lop [EB/OL]. [2022-10-17]. http://lcamtuf.coredump.cx/afl/.

[11] Fioraldi A，Maier D，Eißfeldt H，*et al*. {AFL++}：combining incremental steps of fuzzing research [C]//14th USENIX Workshop on Offensive Technologies (WOOT 20). [S.l.：s.n.]，2020：10.

[12] Serebryany K. Oss-fuzz-google's continuousfuzzing service for open source software [EB/OL]. [2022-10-17]. https://github.com/google/oss-fuzz/.

[13] Böhme M，Pham V T，Nguyen M D，*et al*. Directed greybox fuzzing [C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. New York：ACM，2017.

[14] Böhme M，Pham V T，Roychoudhury A. Coverage-based greybox fuzzing as markov chain [C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York：ACM，2016.

[15] TN_root. AFL 技术实现分析 [EB/OL]. [2018-06-06]. https://blog.csdn.net/qq_32464719/article/details/80592902.

[16] Ahmed A，Hiser J D，Nguyen-Tuong A，*et al*. BigMap：future-proofing fuzzers with efficient large maps [C]//Proceedings of the 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). Taiwan：IEEE，2021.

[17] Hsu C C，Wu C Y，Hsiao H C，*et al*. Instrim：lightweight instrumentation for coverage-guided fuzzing [C]//Symposium on Network and Distributed System Security (NDSS). San Diego：ISOC，2018.

[18] Gan S，Zhang C，Qin X，*et al*. Collafl：path sensitive fuzzing [C]//Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP). San Francisco：IEEE，2018：679.

引用本文格式：

中　文：王松，方勇，贾鹏. 无碰撞灰盒模糊测试方法研究[J]. 四川大学学报：自然科学版，2023，60：033004.

英　文：Wang S，Fang Y，Jia P. Research on Collision-Free grey box fuzzing method [J]. J Sichuan Univ：Nat Sci Ed，2023，60：033004.