计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(9):192−200 [doi: 10.15888/j.cnki.csa.008679]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于聚类和新覆盖信息的模糊测试改进①

程 亮[1,2], 王化磊[1,2], 张 阳[1,2], 孙晓山[1,2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院 软件研究所 可信计算与信息保障实验室, 北京 100190)
通信作者: 王化磊, E-mail: wanghualei19@mails.ucas.ac.cn

摘 要: 模糊测试在挖掘软件安全漏洞、提高软件安全性方面发挥着巨大的作用, 本文针对模糊测试变异策略效率较低以及种子评分策略不合理的问题进行了讨论, 提出了基于聚类的变异优化策略和基于新覆盖信息的能量分配策略. 第 1 个改进策略通过产生新覆盖的非确定性变异提取有效的组合变异位置, 然后利用聚类算法进一步确定有效变异的位置, 在变异阶段对有效变异的位置进行细粒度确定性变异. 本文第 2 个改进策略针对种子评分策略, 种子产生的新覆盖信息与静态分析的分支转移信息作为种子评分的重要指标. 我们将改进后的模糊测试工具-AgileFuzz 与现有的模糊测试改进工具 AFL 2.52b、AFLFast 以及 EcoFuzz 进行比较, 对 binutils、libxml2 等开源程序进行了多次实验. 实验结果表明, AgileFuzz 在相同时间内发现了更多的程序分支覆盖, 并且在测试过程中发现了 fontforge、harfbuzz 等开源软件中 5 个未知的漏洞.

关键词: 模糊测试; 漏洞挖掘; 聚类算法; 静态分析

引用格式: 程亮,王化磊,张阳,孙晓山.基于聚类和新覆盖信息的模糊测试改进.计算机系统应用,2022,31(9):192–200. http://www.c-s-a.org.cn/1003-3254/8679.html

Fuzzy Test Improvement Based on Clustering and New Coverage Information

CHENG Liang[1,2], WANG Hua-Lei[1,2], ZHANG Yang[1,2], SUN Xiao-Shan[1,2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Fuzzing plays a huge role in discovering software security vulnerabilities and improving software security. This study discusses the low efficiency of the mutation strategy for fuzzing and the unreasonableness of the seed scoring strategy and proposes a mutation optimization strategy based on clustering and an energy allocation strategy based on new coverage information. The former improvement strategy extracts the positions of effective combined mutations by generating new coverage of non-deterministic mutations, uses clustering algorithms to further determine the positions of effective mutations, and implements fine-grained deterministic mutations at positions of effective mutations in the mutation stage. The latter improvement strategy in this study is for the seed scoring strategy. The new coverage information generated by the seed and the branch transfer information from the static analysis are used as important indicators of seed scoring. We compare the improved fuzzing tool AgileFuzz with existing ones such as AFL 2.52b, AFLFast, and EcoFuzz and conduct multiple experiments on open source programs such as binutils and libxml2. The experimental results show that AgileFuzz finds more program branch coverage in the same amount of time. Meanwhile, five unknown vulnerabilities in fontforge, harfbuzz, and other open source software are discovered during the testing.

Key words: fuzzing; vulnerability discovery; clustering algorithm; static analysis

① 基金项目: 国家自然科学基金 (62072448)
收稿时间: 2021-12-15; 修改时间: 2022-01-12; 采用时间: 2022-01-26; csa 在线出版时间: 2022-06-16

© 中国科学院软件研究所 http://www.c-s-a.org.cn

1 引言

随着计算机和网络技术的发展, 软件与我们的生活和工作的关系越来越密切. 然而软件复杂的功能使得软件中不可避免地存在漏洞. 在模糊测试首次被提出的论文[1] 中提到“常用系统中可能会潜伏着严重的漏洞”. 比如: 2010 年披露的震网蠕虫漏洞[2] 是第一种攻击工业控制系统的病毒; 2015 年 6 月, 三星被爆出了高危的输入法漏洞. 该漏洞影响全球超过 6 亿的三星手机用户[3]. 2015 年 7 月 Android 被爆出存在 Stagefright 高危漏洞, 约 95% 的安卓设备受到该漏洞影响[3]. 2020 年 8 月, 研究人员发现 CSP 绕过漏洞 (CVE-2020-6519[4]), 该漏洞使攻击者可以完全绕过 Chrome 73 版至 83 版的 CSP 规则, 数十亿的用户可能会受到影响.

软件安全一直是安全人员研究的重点, 当前主流的程序安全分析方法有: 污点分析、符号执行、代码审计和模糊测试等. 如何高效地发掘软件可能存在的漏洞一直是这些主流安全分析方法研究的重点.

污点分析技术[5] 可以分为动态污点分析和静态污点分析, 静态污点在无法获取源代码时, 静态污点分析的精确性将会大大降低. 动态污点分析在运行时具有很大的开销, 难以实际分析规模较大的软件. 符号执行系统地探索许多可能的执行路径, 而不需要具体的输入, 通过抽象地将输入表示为符号, 利用约束求解器来构造可能满足条件的输入. 符号执行的缺点是处理像循环这样的语言结构时可能会成倍地增加执行状态的数量, 从而出现路径爆炸的问题[6–8]. 代码审计是一种对源代码的全面分析技术, 但是代码审计非常依赖安全人员自身的经验. 综上分析, 符号执行等方法虽然通过提取丰富的代码细节达到程序分析的功能, 但是由于其效率等方面的原因, 在程序安全性分析方面存在较多的局限. 模糊测试[9] 通过随机变异种子或者基于规则生成种子对程序进行测试. 模糊测试通常分为白盒模糊测试、灰盒模糊测试和黑盒模糊测试[10–13]. 白盒模糊测试提取程序详细的运行时信息, 但是开销很大, 总体效率很低. 黑盒模糊测试完全忽略了程序的内在信息, 虽然具有较快的运行速度, 但是准确性很低, 总体漏洞发现能力较弱. 而灰盒模糊测试兼顾了两者的优势, 通过静态代码插桩的方式获取程序运行时覆盖情况, 并用获得的信息指导种子变异方式, 以尽快扩大测试的覆盖率, 相较于其他方法能够更快地发现程序潜在的漏洞. 因此, 灰盒模糊测试已成为目前主流的模糊测试方法, 其中典型工具—AFL (American fuzzy loop)[14] 已成为学术界和工业界进行模糊测试的事实标准工具, 我们的工作也基于 AFL 实现.

虽然灰盒模糊测试在发现漏洞方面具有一定的优势. 但是其采用的随机性算法具有极大的盲目性, 所以仍然存在进一步改进的空间.

现有的针对基于 AFL 灰盒模糊测试的改进工具将全部或过多的能量集中非确定性变异阶段, 加大了变异的盲目性, 使其难以求解复杂的条件约束;并且在种子能量分配阶段忽视了种子探索新分支的能力, 使得过多的能量消耗在没有价值的种子.

针对上述问题, 我们在模糊测试工具 AFL 2.52b 进行了改进, 通过对关键变异位置持续性变异和能量分配策略的优化, 提高模糊测试发现程序覆盖和程序潜在漏洞的能力, 我们的设计有以下几个特点:

1) 提取有效变异位置. 在种子变异阶段, 首先对种子执行非确定性变异. 对于产生新覆盖的非确定性变异种子, 我们提取变异的组合位置并认为该组合位置中存在有效变异位置 (我们定义变异后能够产生有效新覆盖的单一位置是有效变异位置), 再利用聚类算法确定组合变异中有效变异位置, 最后针对有效变异位置以及其后续位置进行持续的细粒度变异.

2) 利用新覆盖信息评估种子. 对于每个发现新覆盖的种子, 保留新发现覆盖的信息. 在种子评分阶段, 根据种子的新覆盖信息对种子进行评分.

3) 提取程序静态信息. 模糊测试改进往往是在缺少程序信息的前提下进行的策略上的优化, 这种优化存在局限性; 而使用污点分析或者符号执行等方式提取程序信息的方法[15–18], 往往因为分析效率较低而影响模糊测试工具总体的效率. 因此我们提取了轻量级的程序静态信息, 在模糊测试阶段结合静态信息对种子的能量分配策略进行更加准确的计算.

我们在主流的模糊测试工具 AFL 的设计思路上完成了改进, 并实现了改进后的模糊测试工具 AgileFuzz 经过实验验证, AgileFuzz 能够在相同的时间内发现更多的程序分支覆盖, 并且能够挖掘到新的程序漏洞.

2 背景

本节主要以 AFL 为例介绍典型灰盒模糊测试的工作流程以及路径统计方式.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

2.1 AFL 工作流程

AFL 主函数是一个条件为 true 的 while 循环, 只有用户主动停止模糊测试工作, 测试才会结束. AFL 的工作流程如图 1 所示, 核心步骤如下: 1) 待测目标程序和程序的输入文件作为 AFL 启动时的原始输入, 初始输入经过运行后会加入种子库中参与后续的模糊测试. 2) AFL 每次执行测试, 都会从种子队列中选择 1 个种子, 根据该种子的执行时间、种子大小、路径覆盖数量、执行次数等进行种子评分, 评分越高的种子在非确定性变异阶段会进行更多次的变异. 并且, AFL 根据路径不同保存了很多种子, 但是 AFL 会将单一分支执行最快的种子标记为 favored, 在种子被选择后, 不是 favored 标记的种子会以较大的概率跳过执行. 3) 种子经过变异后, 得到的新种子作为待测程序的输入, AFL 监视每个种子执行过程中程序是否发生崩溃, 如果待测程序发生崩溃, 则种子被保存在 crash 种子队列. 4) AFL 还记录每个种子在程序执行过程中的路径覆盖情况, 如果产生新的覆盖则将种子加入正常样本队列, 然后参与后续的模糊测试工作.

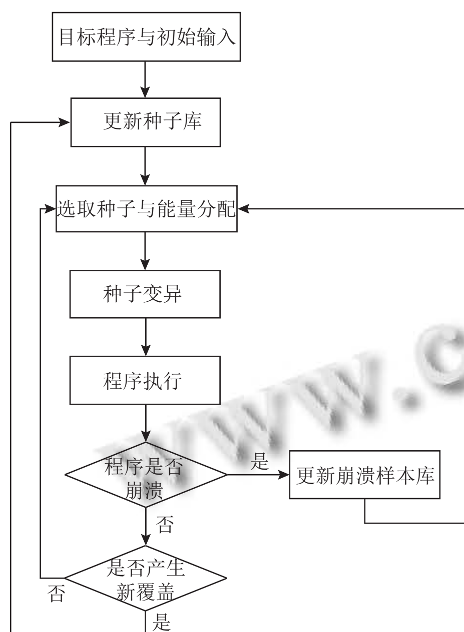

图 1 AFL 工作流程图

2.2 AFL 路径统计方式

AFL 通过插桩方式获取程序运行时的执行路径信息. 对于有源码的程序, AFL 在编译程序时将插桩代码插入程序的每个基本块中, 具体流程如下: AFL 在每个基本块入口处插入 0–65535 范围的随机数以及特定功能的函数 afl_maybe_log;如果程序执行了该基本块, afl_maybe_log 函数会将插入基本块的随机数与前一个执行的基本块的随机数进行异或操作, 运算的结果作为覆盖信息写入共享内存. 共享内存是一块 64 KB 的内存空间, 可以统计 65 536 条分支转移信息, AFL 不仅记录分支转移是否发生, 还会记录分支转移执行的次数, 因为同一个分支转移, 可能因为不同的执行次数出现不用的程序状态. 对于没有源码的程序, AFL 使用运行时插桩技术统计程序执行路径, 插桩的逻辑与静态插桩方式相同.

3 方案设计

如图 2 所示, 其中实线方框为 AFL 的标准功能模块, 使用虚线方框的部分是我们增加或改进的模块, 主要包括 3 个部分: 第 1 部分是对模糊测试变异策略的改进, 第 2 部分是利用聚类算法确定关键变异位置, 第 3 部分是结合静态分析和新覆盖信息的种子评分策略调整.

3.1 变异策略改进

AFL 在种子变异阶段分为确定性变异和非确定性变异. 每个被选中的种子都会执行一次确定性变异, 在确定性变异阶段会对种子进行比特位粒度的变异, 所以在确定性变异阶段十分耗时. 事实上, 种子的很多位置都是无效的变异位置, 即无论进行何种变异都无法产生新的覆盖. MOPT 和 EcoFuzz 考虑到确定性变异耗时, 所以选择直接跳过了确定性变异, 但是只通过非确定性变异, 将会使得模糊测试变异更加盲目和随机.

(1) 确定种子关键变异位置. 在调整后的变异策略中, 种子变异阶段只对种子的部分关键位置进行确定性变异. 核心步骤是确定种子的关键位置. 具体来说, 当选中一个种子后, 首先进行非确定性变异, 如果非确定性变异生成的种子访问了新的分支覆盖, 并不会在保存种子后直接进行下一次变异操作, 而是分析产生该种子的变异过程. 通过分析, 可以确定组合变异位置中有效的变异位置, 而有效的变异位置将会记作新种子的关键位置.

(2) 选择性确定性变异. 在确定关键位置 w 后, 变异前的种子和变异后的种子会针对关键位置进行不同的变异操作. 对变异前的种子的 w 位置进行与 AFL 相同的确定性变异, 对于记录关键位置的新种子, 如果关

© 中国科学院软件研究所 http://www.c-s-a.org.cn

键位置的字段长度小于 m 个字节, 那么当该种子被选中进行变异时, 首先对该关键位置的后续 n 个位置进行更加细粒度的确定性变异, 与 AFL 原有的确定性变异不同, 细粒度的确定性变异会将选中的位置 (字节) 变异成所有可能的结果. 经过多次实验测试, m 和 n 的值取 2 时, 总体效果较好.

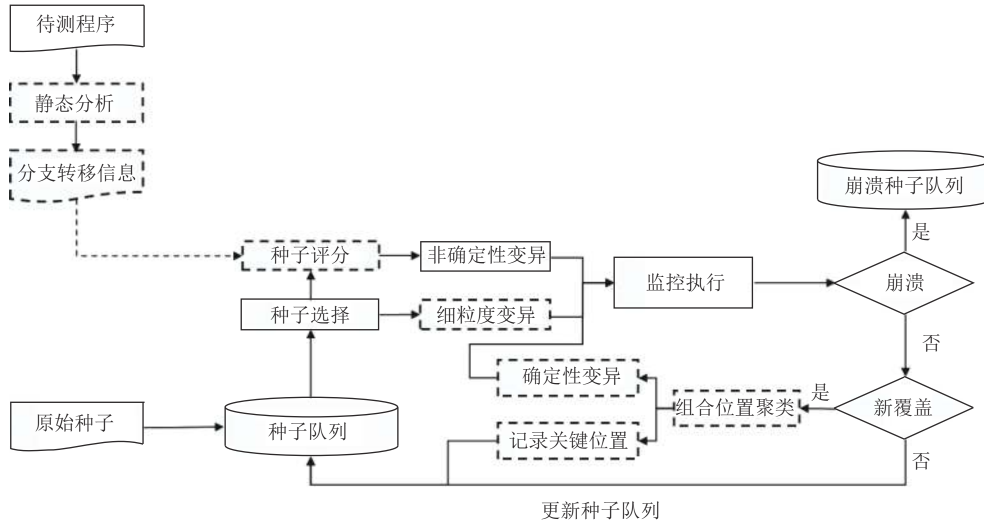

图 2 改进后的模糊测试工作流程

3.2 聚类确定关键变异位置

关键位置的长度直接影响变异的效率. 如果将产生新覆盖的种子变异过程中的所有组合变异位置都看作关键位置, 然后对这些进行确定性变异, 仍然存在较多的无效变异.

为了缩小关键位置的长度, 使用聚类算法对离散的组合变异位置进行聚类. 在文件结构中, 字段是连续的一段内容, 所以在一个文件中, 距离越近的位置越可能属于同一字段. 如图 3 所示, 文件 A 的组合变异 M 个位置 (其中包括蓝色标记和红色位置标记的位置, 而且红色位置为关键变异位置) 得到的种子 B 访问了新的覆盖, 对种子 B 变异的位置进行聚类操作-将离散的变异位置聚类为 N 个字段, 根据 N 个字段生成 N 个新的种子, 每个新种子只有对应字段的内容与原文件不同. 如果种子 Ni 访问了新的覆盖, 那么 Ni 种子对应的变异字段就是关键的变异字段.

为了完成变异位置聚类, 我们使用 Meanshift 算法.使用该算法的原因有两点: 1) 该算法是基于密度的聚类算法, 2) 聚类之前不需要指定类的数量. Meanshift 算法的核心思想是通过不断移动样本点, 使其向密度更大的区域移动, 直到满足某一条件, 此时该点就是样本点的收敛点, 收敛到同一点则被认为属于同一族类. 对于变异位置 p, 其偏移向量计算方式如式 (1) 所示, 其中, p_i 表示所有变异位置点集中到 p 点距离小于 l 的所有点.

$$M_l = \frac{1}{k}\sum\nolimits_{p_i \in p_k}(p_i - p) \tag{1}$$

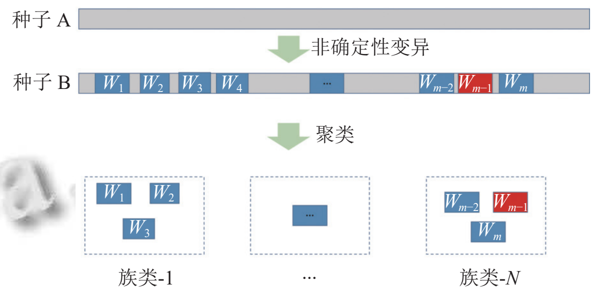

图 3 对组合变异位置进行聚类

聚类算法具体步骤如下:

1) 随机选择未被分类的变异位置, 作为中心点 P.

2) 找出离中心点 P 距离 L 范围的所有变异点, 这些点记作集合 M.

3) 计算中心点 P 与集合 M 中所有点的向量, 将这些向量相加得到偏移向量 M.

4) 将中心点 P 加上向量 M 得到新的中心点.

5) 重复步骤 2)–4), 直到偏移向量满足设定的阈值, 保存此时的中心点.

6) 重复步骤 1)–5) 直到所有点都完成分类.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

7) 根据每个类, 对每个点的访问频率, 取访问频率最大的那个类, 作为当前点集的所属类.

算法 1. 基于聚类的确定性变异优化方法

输入: 非确定性变异后产生新覆盖的种子 seed_son 和变异前的种子 seed
输出: 确定性变异产生的新覆盖的种子

```
1 diff_points=get_diff_point(seed, seed_son)
2 cluster_points=cluster(diff_ponts)
3 for cluster in cluster_points:
4     seed_single_cluster=generate_seed_by_cluster(cluster):
5     if(is_new_path(seed_single)):
6         seeds_deter=deter_variate(cluster, seed):
7         while seed_deter in seeds_deter:
8             fuzzy_run(seed_deter)
9         end while
10    end if
11 end for
```

3.3 新覆盖与静态分析进行种子评分

在 AFL 的种子评分中, 会根据种子的总分支覆盖数量进行能量分配, 这种方式导致能量分配不合理. 我们调整种子评分策略, 将种子新发现的分支数量作为评分依据. 并且我们对待测二进制程序进行静态分析, 提取基本块转移信息. 这样做的目的是因为并不是每个分支转移都存在潜在未发现的分支转移.

如图 4 所示, 对于每个基本块转移, 如: branch 1, 我们记录通过该分支转移到达基本块后, 可能存在的后续基本块转移, 即对于 branch 1, 我们记录 branch 1-1 和 branch 1-2 的信息, 我们把 branch 1-1 和 branch 1-2 记作分支 branch 1 的子分支转移. 注意的是, 虽然基本块是连续指令的集合, 但是 branch 1-1 或 branch 1-2 可能并不同时存在或都不存在. 这种情况发生在某基本块子节点的父节点超过一个时, 其子节点即使只有一个, 但是仍然是一个独立的基本块. 通过静态分析, 我们可以得到每个基本块转移的子分支转移, 如果某个分支的子分支转移数量为 2, 我们认为该分支转移对发现新覆盖是有价值的, 将会对种子评分产生增益效果.

所以在种子评分阶段, 具体步骤如下:

1) 对于待测程序, 通过静态分析获取每个基本块转移的子分支转移信息, 保存在文件中, 在启动模糊测试测试时, 该文件作为分支附加信息提供.

2) 对于每个新加入队列的种子, 比较其分支覆盖与总分支覆盖的差异, 得到新发现的分支覆盖数量和具体分支编号.

3) 在种子评分阶段, 计算种子分支覆盖中是新覆盖且对应分支有两个子分支的数量, 记作 *value_branch_num*.

4) 种子 A 的评分使用式 (2) 进行计算, 使用 log 函数是为了避免分支覆盖数量对种子能量进行成倍的增益, 并且经过多次实验验证, ln 函数作为计算种子能量总体实验效果较好. 根据公式可知, 当种子的新覆盖数量为 0 时, 最低能量值为 1.

$$socre_A = \ln(value_branch_num + 1) + 1 \tag{2}$$

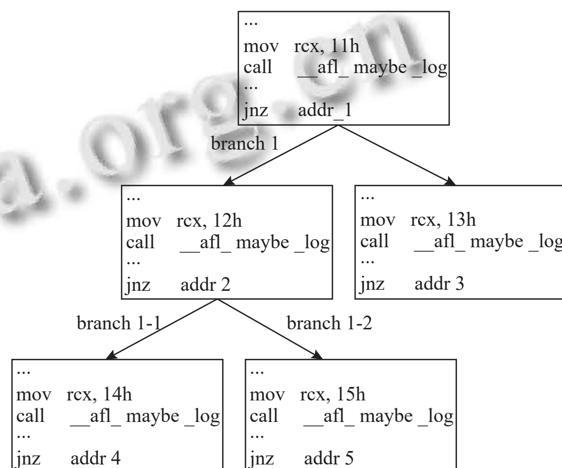

图 4　基本块转移示意图

4　实验评估

我们在 AFL 2.52b 的基础上实现了改进策略, 得到了改进后的模糊测试工具 AgileFuzz. 我们的评估主要回答下面 4 个问题:

1) 产生新覆盖的非确定性变异的组合位置中是否存在关键位置? 如果存在, 存在的比例是多少? 对关键位置进行确定性变异, 是否能够发现新的覆盖?

2) 与非确定性变异相比, 对聚类得到关键位置进行细粒度持续变异如何能够更快地发现新覆盖?

3) AgileFuzz 在实际程序中发现程序覆盖的能力与现有的模糊测试改进工具对比如何?

4) AgileFuzz 是否能够发现真实程序程序中的漏洞?

4.1 “关键位置”分析与统计

非确定性变异阶段对种子的随机组合位置进行变异, 变异后的种子产生新覆盖可能有两个原因: 1) 组合变异产生新覆盖, 单一位置变异并不能产生; 2) 组合变异中, 变异了“关键位置”, 而且即使只变异该位置, 也可以产生新覆盖. 其中, 我们称这种“关键位置”为单一

© 中国科学院软件研究所　http://www.c-s-a.org.cn

有效变异位置. 我们统计在模糊测试过程中, 产生新覆盖的非确定性变异包括多少“关键位置”以及针对“关键位置”变异仍然能够产生新覆盖的数量. 实验测试选择 libxml2 (xmllint)、binutils (readelf) 和 harfbuzz (test) 这 3 组程序, 这 3 组程序常用作模糊测试工具的测评, 比较具有代表性. 在相同的实验环境下, 对 3 个程序进行 3 组重复性实验, 每组实验进行 72 h, 最终取平均结果. 统计结果如表 1 所示, 从结果中可以看出, 产生新覆盖的非确定性变异次数中, 具有“关键位置”的非确定性变异比例较高 (分别为 59%、45% 和 36%), 并且 90% 以上的“关键位置”经过确定性变异后仍然能够产生新覆盖.

表 1 单一有效位置统计

程序	产生新覆盖的非确定性变异次数	具有“关键位置”的非确定性变异次数	仍然产生新覆盖的次数
xmllint	976	581	570
test	1 054	478	465
readelf	2 483	901	897

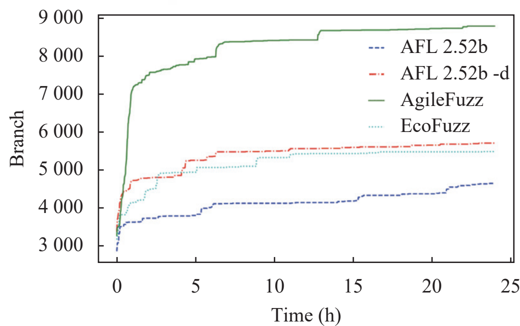

图 5 xmllint 程序覆盖率对比

4.2 新覆盖发现效率的分析

在本小节, 我们解释为什么针对“关键位置”进行细粒度变异相比非确定性变异能够更快地发现新覆盖. 我们以 libxml2 的 xmllint 程序为例, 将我们的模糊测试工具 AgileFuzz 与 AFL 2.52b、关闭确定性变异的 AFL 2.52b -d 以及关闭确定性变异的模糊测试改进工具 EcoFuzz 进行 24 h 实验比较, 覆盖率对比结果如图 5 所示, 图中可以看出 AgileFuzz 在相同的时间内发现了更多的路径. 我们将对这一实验结果进行详细的分析, 并展示我们的模糊测试工具的优势.

为了分析覆盖率差异, 我们使用 afl-cov[19] 分析 4 个模糊测试工具所发现的程序覆盖的差异. 分析结果显示, AgileFuzz 不仅发现了 EcoFuzz 等工具所发现的全部程序代码, 还发现了更多的难以发现的程序代码. 其中 hash.c 文件的代码, AgileFuzz 覆盖率为 56.3%, 而 EcoFuzz 等工具为 0%, 这是导致程序覆盖率出现较大差异的主要原因. 我们进一步分析产生现象的原因, 通过程序分析发现如图 6 所示的调用序列以及条件判断. 这里简单介绍一下 CMP9 函数, 这是 xmllint 实现的定长字符串匹配宏定义, 与 C 语言自带的 strcmp 函数不同的是, “<”和“<!”虽然都没有完全满足匹配条件, 但是调用 cmp9 触发的程序覆盖是不同的, 而调用 strcmp 触发的程序覆盖是相同的.

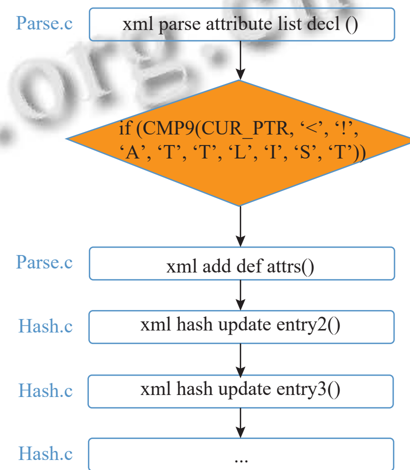

图 6 xmllint 程序 hash.c 调用依赖

EcoFuzz 通过 24 h 的变异无法产生满足条件约束的字符串-“<!ATTLIST”, 这是非确定性变异的随机性和组合性所造成的. 当 EcoFuzz 得到包括“<”的种子后, 保存该种子. 当从种子队列选择该种子后, 随机选择变异位置, 选择“<”后继位置的可能性很低, 并且组合变异可能会导致“<”或者其他已经满足约束条件的字段被修改为错误的内容, 从而导致无法产生有效变异. 未关闭确定性变异的 AFL 2.52b, 虽然会对种子所有位置进行变异, 但是由于其变异效率较低, 同样没有产生满足约束条件的种子.

AgileFuzz 在通过非确定性变异产生了包含字符串“<**”种子时, 触发了 cmp9 函数新的覆盖, 此时 AgileFuzz 利用聚类算法确定单一有效变异位置, 确定新覆盖是由于变异了“<”字符所在的位置. 保存的新种子记录了关键位置, 能够对关键位置的后继位置进行持续性的细粒度变异, 从而保证了产生“<!”字符串, 迭代下去则保证能够产生“<!ATTLIST”字符串, 从而满足条件约束. 图 7 为 AgileFuzz 通过确定性变异产生满足 cmp9 函数条件约束的种子的产生序列, 红色标记的

© 中国科学院软件研究所 http://www.c-s-a.org.cn

字段是种子记录的关键位置, 对于 id 为 001889 种子, AgileFuzz 会对“<”的后继字符进行细粒度确定性变异, 从而能够迅速产生“<!**”字符串.

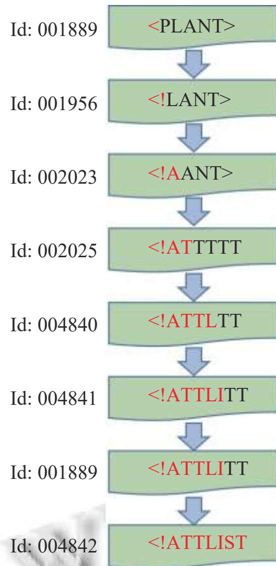

图 7　种子变异过程

4.3　程序覆盖率对比

为了验证我们改进的模糊测试工具 AgileFuzz 在通用程序上的测试效果, 我们与近年来效果比较好的 EcoFuzz、AFLFast 以及原始的 AFL 2.52b 进行了实验, 实验以覆盖率作为对比指标, 选择了 libxml2、binutils 等常用的程序作为测试对象. 在相同的硬件和软件环境中进行了 3 次实验, 每次实验的时间为 72 h, 3 次实验的覆盖率的平均值作为实验结果, 然后计算覆盖率对比图, 如图 8 所示. 通过实验对比可以看出, AgileFuzz 能够更快地发现程序的覆盖, 并且在不同的程序中效果较为稳定.

4.4　漏洞挖掘能力

如表 2 所示, 为体现工具的普适性, 我们使用 AgileFuzz 对字体解析、html 解析等多种类型的程序进行漏洞挖掘工作, 如: fontforge、harfbuzz、htmlcss、ffjpeg 等软件, 在挖掘工作中发现了大量的未知漏洞, 经过分析确定了漏洞类型, 并将漏洞崩溃样本以及漏洞分析结果反馈给作者. 通过在实际程序中发现的未知漏洞, 证明了 AgileFuzz 具有发现实际程序漏洞的能力.

www.c-s-a.org.cn

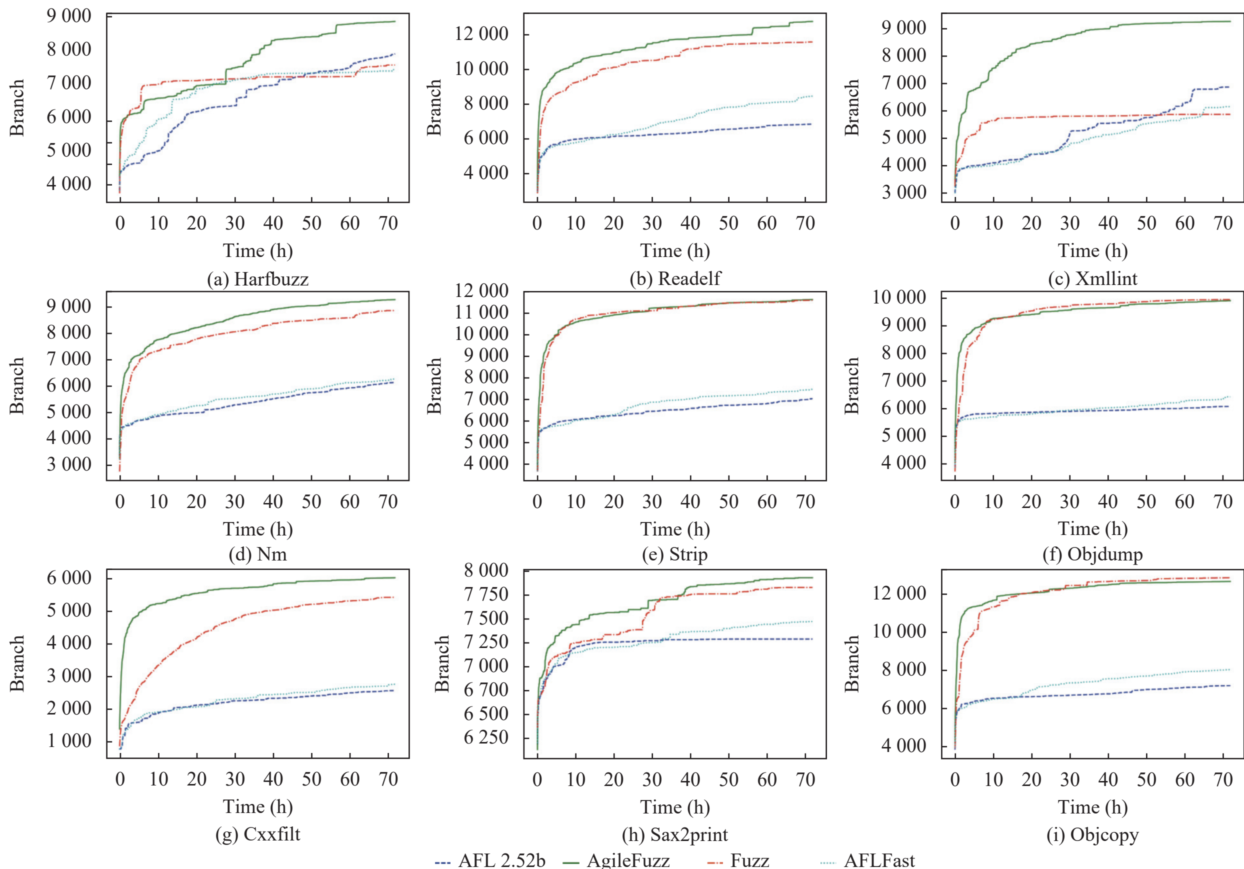

图 8　多个程序 72 h 的覆盖率比较

© 中国科学院软件研究所　http://www.c-s-a.org.cn

表 2 程序漏洞挖掘结果统计

软件	版本	漏洞类型	状态
fontforge	2021.5.9	内存泄漏	等待CVE审核
harfbuzz	2.8.0	内存泄漏	等待CVE审核
htmlcss	2021.5.9	内存泄漏	等待CVE审核
packJPG	2021.4.6	堆溢出	等待CVE审核
ffjpeg	2021.4.4	使用未初始化的值	等待CVE审核

5 相关工作

国际很多研究工作都针对 AFL 进行了改进. 这些针对 AFL 的改进主要在种子变异、能量分配方面进行了改进[20–23], 在确定性变异改进方面, FairFuzz[21] 根据是否包括稀有分支跳过部分确定性变异, MOPT[22] 详细分析了确定性变异效率低对变异的影响, 并只保留了极少数的确定性变异, EcoFuzz[19] 直接跳过了所有的确定性变异. 在种子评分改进方面, AFLFast[20] 使用马尔科夫链对种子能量分配进行建模, 被选次数更多的和被执行次数较少的种子分配的能量较高. EcoFuzz[19] 使用多臂老虎机对模糊测试进行更准确的建模以完成能量分配. 但是这些模糊测试改进存在以下两个问题:

1) 变异更加盲目和随机. 这些模糊测试工具认为确定性变异效率较低, 采取的策略是直接跳过全部或大部分确定性变异, 但是只保留非确定性变异的模糊测试在对种子进行变异时更加盲目和随机, 无法对产生新覆盖的变异位置进行持续的变异.

2) 忽视了种子产生的新分支覆盖数量对模糊测试的影响. 种子评分阶段根据种子大小、种子运行速度和种子总分支覆盖数量对种子进行评分, 但是总分支覆盖数量越多的种子新发现的分支覆盖数量并不一定越多, 所以新分支数量多的种子评分可能很低, 这导致模糊测试将大量的能量消耗在已经经过多次变异的分支覆盖, 发现新覆盖的能力受到限制, 并且作为影响种子评分的分支信息缺少程序的静态分析.

6 结束语

模糊测试是一种高效的漏洞挖掘工具, 能够发现程序真实的漏洞. 本文设计了一种基于聚类算法和新覆盖的模糊测试改进工具, 能够针对单一有效位置进行持续性细粒度变异, 并且利用待测程序的静态分析结果与新覆盖信息结合对种子评分进行调整, 使得更多的能量分配给新分支, 降低了变异的盲目性. 总体来看, 我们的改进取得了较好的效果.模糊测试仍然存在很多的工作需要进一步研究, 在将来的工作中, 我们将研究如何针对程序的长字符串和整数匹配进行拆分, 提高针对关键位置进行细粒度变异的适用性和效率.

参考文献

1 Miller BP, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities. Communications of the ACM, 1990, 33(12): 32–44. [doi: 10.1145/96267.96279]

2 陈梁. “震网”病毒敲响自动化系统安全警钟. 工业控制计算机, 2010, 10: 94.

3 柒月. 2015 年度国际网络安全重大事故攻击、漏洞无处不在, 数据安全危在旦夕. 信息安全与通信保密, 2016, 2: 38–39. [doi: 10.3969/j.issn.1009-8054.2016.02.013]

4 CVE. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-6519. (2020-01-08).

5 任玉柱, 张有为, 艾成炜. 污点分析技术研究综述. 计算机应用, 2019, 39(8): 2302–2309. [doi: 10.11772/j.issn.1001-9081.2019020238]

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

7 Zhang T, Jiang Y, Guo RS, *et al.* A survey of hybrid fuzzing based on symbolic execution. Proceedings of the 2020 International Conference on Cyberspace Innovation of Advanced Technologies. Guangzhou: ACM, 2020. 192–196.

8 Baldoni R, Coppa E, D’elia DC, *et al.* A survey of symbolic execution techniques. ACM Computing Surveys, 2019, 51(3): 50.

9 任泽众, 郑晗, 张嘉元, 等. 模糊测试技术综述. 计算机研究与发展, 2021, 58(5): 944–963. [doi: 10.7544/issn1000-1239.2021.20201018]

10 李红辉, 齐佳, 刘峰, 等. 模糊测试技术研究. 中国科学: 信息科学, 2014, 44(10): 1305–1322.

11 张婉莹. 白盒模糊测试技术的研究与改进 [硕士学位论文]. 南京: 南京邮电大学, 2019.

12 Duchene F. Fuzz in the dark: Genetic algorithm for black-box fuzzing. São Paulo: Black-Hat, 2013.

13 陈衍铃, 王正. 模糊测试研究进展. 计算机应用与软件, 2011, 28(7): 291–293, 295. [doi: 10.3969/j.issn.1000-386X.2011.07.088]

14 Technical “whitepaper” for afl-fuzz. http://lcamtuf.coredump.cx/afl/technical_details.txt.

15 Chen P, Chen H. Angora: Efficient fuzzing by principled search. Proceedings of 2018 IEEE Symposium on Security

© 中国科学院软件研究所 http://www.c-s-a.org.cn

and Privacy (SP). San Francisco: IEEE, 2018. 711–725.

16 Jain V, Rawat S, Giuffrida C, *et al*. TIFF: Using input type inference to improve fuzzing. Proceedings of the 34th Annual Computer Security Applications Conference. San Juan: ACM, 2018. 505–517.

17 Wang MZ, Liang J, Chen YL, *et al*. SAFL: Increasing and accelerating testing coverage with symbolic execution and guided fuzzing. Proceedings of the IEEE/ACM 40th International Conference on Software Engineering: Companion. Gothenburg: IEEE, 2018. 61–64.

18 Zhang L, Thing VLL. A hybrid symbolic execution assisted fuzzing method. Proceedings of 2017 IEEE Region 10 Conference. Penang: IEEE, 2017. 822–825.

19 GitHub. https://github.com/mrash/afl-cov. (2018-12-29).

20 Böhme M, Pham VT, Roychoudhury A. Coverage-based Greybox fuzzing as Markov chain. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Vienna: ACM, 2016. 1032–1043.

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

22 Lyu C, Ji SL, Zhang C, *et al*. MOPT: Optimize mutation scheduling for fuzzers. Proceedings of the 28th USENIX Security Symposium. Santa Clara: USENIX Association, 2019. 1949–1966.

23 Yue T, Wang PF, Tang Y, *et al*. EcoFuzz: Adaptive energy-saving greybox fuzzing as a variant of the adversarial multi-armed bandit. Proceedings of the 29th USENIX Security Symposium. USENIX Association, 2020. 130.

(校对责编: 牛欣悦)

www.c-s-a.org.cn

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn