第47卷 第3期 计 算 机 工 程 2021年3月
Vol.47 No.3 Computer Engineering March 2021

·网络空间安全· 文章编号:1000-3428(2021)03-0147-08 文献标志码:A 中图分类号:TP309

距离与权重相结合的导向式灰盒模糊测试方法

李明磊,陆余良,黄 晖,朱凯龙
(国防科技大学 电子对抗学院,合肥 230037)

摘 要:导向式灰盒模糊测试是一种能够快速对程序指定位置进行测试的技术。通过对当前导向式灰盒模糊测试技术导向不够精确的问题进行分析,提出一种新的导向式灰盒模糊测试方法,并引入基本块权重与函数路径长度的概念。通过对被测程序的静态分析,构建被测程序的函数调用图和控制流程图,计算更准确的基本块距离并插桩到被测程序中。在模糊测试时通过插桩追踪并计算每个测试用例到指定目标的距离,模糊测试器依据该距离计算种子能量以实现对目标区域的导向,并基于该方法实现原型系统Afl-guide。实验结果表明,与现有的导向式模糊测试方法相比,该方法对目标区域导向更精确、路径覆盖更广,能够更快地生成覆盖程序指定位置的测试用例。
关键词:灰盒模糊测试;距离向量;基本块;种子能量分配;漏洞检测

开放科学(资源服务)标志码(OSID):

中文引用格式:李明磊,陆余良,黄晖,等.距离与权重相结合的导向式灰盒模糊测试方法[J].计算机工程,2021,47(3):147-154.
英文引用格式:LI Minglei,LU Yuliang,HUANG Hui,et al.Guided grey-box fuzzing test method combining distance and weight[J].Computer Engineering,2021,47(3):147-154.

Guided Grey-Box Fuzzing Test Method Combining Distance and Weight

LI Minglei,LU Yuliang,HUANG Hui,ZHU Kailong
(College of Electronic Engineering,National University of Defense Technology,Hefei 230037,China)

【Abstract】 Guided grey-box fuzzing test is a technique that can quickly test a specified location of a program. By analyzing the problem that the existing guided grey-box fuzzing test techniques are not accurate enough in guidance, this paper proposes a guided grey-box fuzzing test method. The method introduces the concepts of basic block weight and function path length. Through the static analysis of the program under test, the function call graph and control flow chart of the program under test are constructed, and the more accurate basic block distance is calculated and inserted into the program. By instrumentation, the distance from each test case to the specified target is tracked and calculated in the fuzzing test. The fuzzing tester calculates the seed energy based on this distance to achieve the guidance of the target area. Based on this method, the prototype system Afl-guide is implemented. The experimental results show that compared with the existing guided fuzzing test methods, the proposed method is more accurate in the guidance of the target area, provides wider path coverage, and can generate test cases covering the specified position of the program faster.
【Key words】 grey-box fuzzing test;distance vector;basic block;seed energy allocation;vulnerability detection
DOI:10. 19678/j. issn. 1000-3428. 0057510

0 概述

随着社会信息化程度的不断提高,网络空间安全成为国家安全的重中之重。在威胁网络空间安全的众多因素中,漏洞是最根本的原因。模糊测试技术是目前最有效的漏洞检测技术之一,其为被测程序提供多种输入并监测被测程序的多种异常行为,如堆栈或缓冲区溢出、内存泄漏等[1-2]。安全人员通过对程序异常行为进行分析,从而实现对软件漏洞位置定位。

1988年,MILLER等人提出模糊测试的概念[3],经过三十多年的发展,已经形成包括白盒模糊测试、灰盒模糊测试、黑盒模糊测试在内的三类模糊测试技术。白盒模糊测试技术基于程序源代码进行分

基金项目:国家重点研发计划“网络空间安全”重点专项(2017YFB0802900)。
作者简介:李明磊(1996—),男,硕士研究生,主研方向为网络安全;陆余良,教授、博士生导师;黄 晖,博士;朱凯龙,博士研究生。
收稿日期:2020-02-26 修回日期:2020-04-09 E-mail:921519263@qq.com

计算机工程 www.ecice06.com

析，能够根据程序源码提供更具有针对性的测试用例，但对程序的分析会引入大量的额外开销[4]。与白盒测试相反，黑盒测试不对被测程序进行分析，而是仅提供持续不断的输入并对程序运行结果进行收集。这使得黑盒测试具有较好的时间效率，能够在短时间内覆盖程序的大量路径，但不具有针对性的测试用例也使得黑盒测试难以覆盖程序深处的路径。灰盒模糊测试是一种结合黑盒与白盒测试特点的测试方法，模糊测试器在提供给程序大量测试用例的同时通过轻量级程序分析工具对程序运行状态进行分析，并根据分析结果对测试用例进行修改。灰盒模糊测试能够在保持黑盒模糊测试高效、扩展性好的基础上获得对程序关键结构信息分析的能力，使模糊测试技术更加智能[5]。

近年来，研究人员将污点分析[6]、符号执行[7]以及静态分析[8]等技术与灰盒模糊测试技术相结合，使得灰盒模糊测试技术在路径覆盖率、时间效率以及路径覆盖深度等方面取得突破。文献[9]开发的模糊测试器TaintScope使用了符号执行与污点分析技术，自动标记输入中的校验和字段并求解出可以通过程序校验和检测的测试用例。文献[10]开发的模糊测试器Dowser使用污点分析技术计算种子文件各比特位之间的突变比例，提高种子文件变异效率。文献[11]开发的模糊测试器BORG使用静态分析技术得到程序的控制流程图，并利用该程序流程图使用符号执行技术生成可以到达程序高危漏洞的测试用例。文献[12]开发的模糊测试器AflFast使用马尔科夫链模型对程序路径状态转换概率进行建模，根据概率模型选择覆盖低概率路径的种子进行变异，以提高新路径的发现速度。文献[13]开发的模糊测试器Skyfire使用概率上下文相关语法从大量现有样本中生成高质量样本。文献[14]开发的模糊测试器VUzzer使用了静态分析与动态污点分析技术，计算每个种子的适应度来提高模糊测试器路径覆盖的深度。

随着程序规模的不断增长，为提高模糊测试的效率，在进行模糊测试前，研究人员通常会对目标程序进行脆弱性分析等工作，选定程序中存在潜在问题的区域作为目标区域进行测试，如程序的补丁区域。但上述的模糊测试器是面向整个程序进行测试，被用于补丁测试、高危区域测试时缺乏导向性，会将大量时间和系统资源浪费在程序的无关区域。因此，文献[15]提出导向式灰盒模糊测试技术，导向式灰盒模糊测试技术能够快速生成到达程序目标区域的测试用例并发现漏洞[16]，其不需要重量级的符号执行、程序分析和约束求解技术，而是先通过静态分析计算出程序各基本块到目标区域的距离，并通过插桩的方式将距离信息插入到目标程序中，在测试阶段根据插桩信息计算每个种子距离目标区域的距离，使用启发式算法提升距离目标区域近的种子生成测试用例的数量，保证了测试的高效性与导向性[17]。

本文对现有的导向式灰盒模糊测试方法进行分析，提出一种距离与权重相结合的导向式灰盒模糊测试方法，通过改进距离计算方法提高导向式灰盒模糊测试的准确性。

1 导向式灰盒模糊测试分析

本文以文献[15]开发的导向式灰盒模糊测试器Aflgo为例，对导向式灰盒模糊测试的基本工作流程进行介绍。导向式灰盒模糊测试器由静态分析与灰盒测试两部分组成，静态分析部分仅在模糊测试开始前运行一次，整体架构如图1所示。

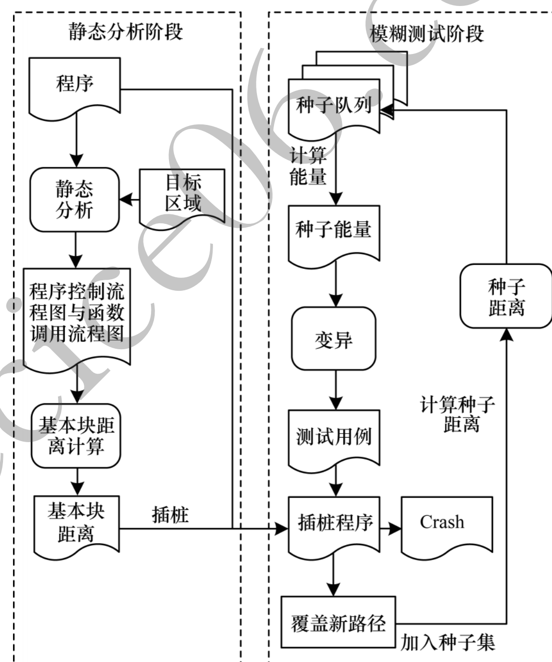

图1 导向式模糊测试技术框架

Fig.1 Framework of guided fuzzing test technology

导向式灰盒模糊测试流程如下：

1）在静态分析阶段，对目标程序进行静态分析得到目标程序的函数调用流程图和程序控制流程图。

2）计算出程序每一个基本块到目标区域的距离并通过插桩的方式将距离信息插入到程序中。

3）模糊测试阶段，从种子集中选择一个种子进行测试，收集该种子执行轨迹并计算种子与目标区域的距离。

计算机工程 www.ecice06.com

4)根据种子到目标区域的距离动态调控种子的能量,即种子变异产生的测试用例的数量。

5)如果变异生成的测试用例覆盖到程序新的路径,就将此测试用例加入种子集。

重复操作流程3)~流程5),当种子集中种子都被选择过一遍时为一轮,一轮测试后从种子集第一个种子开始新一轮测试直到时间结束。

Aflgo通过与Afl及导向式符号执行工具KATCH[16]进行对比实验,证明了其导向的有效性,但依然存在以下3个问题:

1)距离计算粒度粗糙,没有仔细区分函数与基本块对种子距离的影响,而是简单地认为函数距离是基本块距离的常数倍。

2)没有对程序不同位置的基本块进行区分,认为程序中基本块的权重是相同的,降低了距离计算的精度。

3)种子能量调控策略以程序运行时间作为调控指标不够准确。程序受运行环境影响较大,同一个程序在不同运行环境下运行相同的时间运行状态有很大的差别。

为解决上述问题,本文提出一种更加精确有效的距离计算与能量调控方法,对图1中的基本块距离计算与种子能量计算进行改进,以提高导向式模糊测试器的导向性。

2 本文方法

实现导向式灰盒模糊测试导向性的关键在于计算种子到目标区域的距离。通过上文的分析可以看出,目前限制距离计算准确性的主要原因有两点:1)如何计算不同函数、不同基本块对种子到目标区域距离的影响;2)如何用统一的量纲表示程序中函数和基本块到目标区域的距离。本文通过对导向式灰盒模糊测试中的距离与能量计算方法进行改进,将不同函数与基本块的差异考虑到距离计算当中,统一函数与基本块的距离计算标准,设计合理的种子能量调控策略。

为提高距离计算的准确性与合理性,本文将距离计算与能量调控分为函数路径长度(P_f)计算、基本块距离计算、种子距离与能量计算3个部分。3个部分为递进关系,计算过程如图2所示。函数路径长度表示不同函数参与距离计算时贡献的距离,基本块距离指程序中基本块到目标区域的距离,种子距离指进行模糊测试过程中种子执行轨迹到目标区域的距离,本文中目标区域由若干同属一个函数的基本块构成。

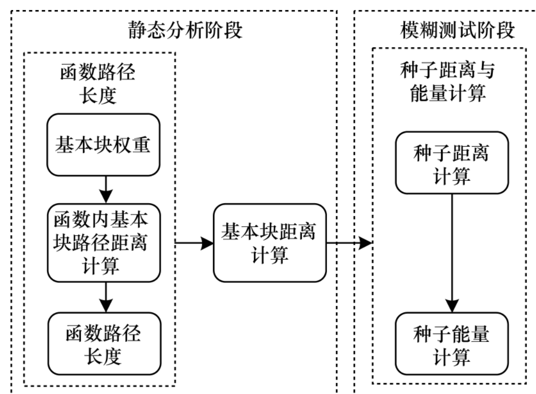

图2 距离计算示意图

Fig.2 Schematic diagram of distance calculation

在静态分析阶段完成程序函数路径长度计算与基本块距离计算并将基本块距离插桩到程序中。在模糊测试阶段,模糊测试器记录种子覆盖到的基本块信息,并根据基本块信息计算种子距离与能量。

在计算函数路径长度(P_f)前引入基本块权重(W_b)概念以提高距离计算的合理性。基本块权重表示基本块在程序路径中的重要程度,以区分不同基本块对距离计算的影响。根据W_b得到函数内部基本块间路径距离计算公式$L(i,j)$,路径距离指两点间最短的一条路径的长度。根据本文的方法利用$L(i,j)$计算程序中每一个函数的函数路径长度,$L(i,j)$由基本块间的距离而来,因此函数路径长度的量纲与基本块距离的量纲是统一的。

得到程序函数路径长度后,根据公式$L(i,j)$与函数调用图可以得到程序任意基本块到目标区域的距离。程序基本块距离计算较为复杂,因此将基本块分为四类进行计算,将计算得到的基本块距离插桩到目标程序中。得到包含插桩信息的程序后,根据模糊测试时种子覆盖到的基本块中包含的基本块距离信息计算种子距离并归一化处理。为了更加合理地对种子能量进行调度,根据种子执行轮次对种子能量进行修正。

3 技术实现

3.1 函数路径长度计算

导向式灰盒模糊测试是为了在有限的时间里尽可能多地发现覆盖目标区域的路径。传统的距离计算方法没有对处于程序路径不同位置的基本块进行区分。以图3所示的程序路径为例,传统的计算方法[15]认为基本块D和基本块B到达目标区域G的距离相同。但从路径图可以看出,经过基本块B到目标区域G有两条路径,而经过D到目标区域G的路径仅有一条,经过基本块B比经过基本块D更容易到达目标区域。传统的计算方式不能体现出不同位置基本块的区别,降低了导向的精度和速度。为提高距离计算的准确性,本文对传统的计算方法进行

改进，给不同位置的基本块赋予不同的权重，使得经过 B 到目标区域 G 的距离小于经过 D 到 G 的距离。

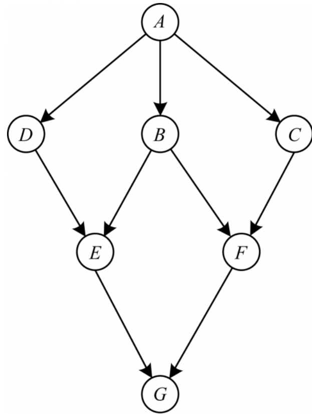

图 3 程序路径示意图

Fig.3 Schematic diagram of program path

通过对程序路径的分析，一个基本块的后继基本块越多，则经过该基本块的种子变异后覆盖不同路径的概率越高。因此，在距离计算中用基本块出度（一个基本块的后继基本块个数）作为基本块的权重，用符号 W_b 表示。

在引入基本块权重后基本块间的距离如图4所示。分别计算图4中 A 到 G 的路径1{A,D,E,G}和路径2{A、B、E、G}的长度，得到路径1长度为7/3，路径2长度为11/6。在未将权重引入前，距离相同的两条路径在引入基本块权重后距离产生差值，经过权重高的基本块的路径距离更短，在变异时可以得到更多的变异机会。

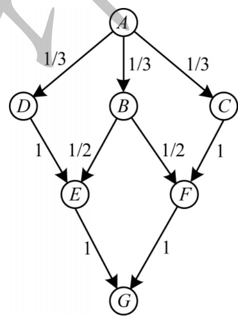

图 4 加权路径距离计算

Fig.4 Distance calculation of weighted path

式(1)表示同属一个函数的基本块 i 与基本块 j 之间的距离，如果从 i 到 j 有多条路径则选择其中最短的一条路径计算，集合 B 表示该路径覆盖到的基本块，$i\in B, j\notin B$。

$$L(i,j)=\sum_{k\in B}\frac{1}{W_k} \quad (1)$$

以图4为例，A 到 G 的路径有4条。选择其中最短的一条利用式(1)计算，可得 A 到 G 的路径距离 $L(A,G)$=11/6。

在之前的距离导向启发式计算规则中[15]，不同函数在基本块距离计算中贡献的距离为相同的常数值，没有考虑到不同函数对基本块距离不同的影响，计算基本块距离时会引起较大的误差。为提高距离计算的精度，本文引入函数路径长度的概念，用函数包含的基本块表示该函数在基本块距离计算中贡献的距离。函数路径长度指一条路径穿过该函数实际经过的距离，用函数入口基本块到函数出口基本块全部路径的距离的平均值表示该长度。

以图5为例，基本块 A 为函数入口基本块，基本块 G 为函数出口基本块。由 A 出发到 G 的路径有4条，分别为{A,D,E,G}、{A,B,E,G}、{A,B,G}和{A,C,G}，长度分别为7/3、11/6、5/6和4/3，因此该函数路径长度为19/12，当该函数参与到距离计算时贡献的距离为19/12。

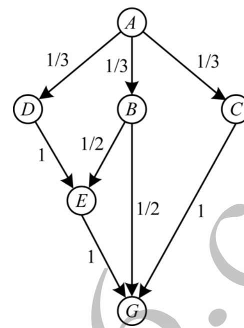

图 5 函数路径长度计算

Fig.5 Length calculation of function path

3.2 基本块距离计算

依据3.1节中得到的同属一个函数的2个基本块间距离计算公式 $L(i,j)$ 与函数路径长度(P_f)计算方法，计算程序基本块到目标区域距离。依据程序中基本块与目标区域的位置分为4种情况讨论，如图6所示。其中，方框表示函数，圆圈表示基本块，三角形表示目标基本块。

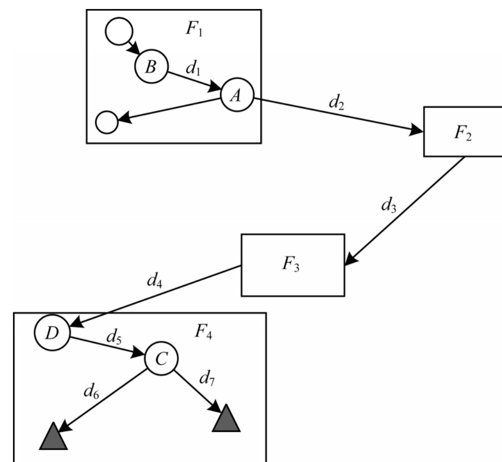

图 6 基本块分类

Fig.6 Classification of basic blocks

为便于计算与讨论，首先引入符号 H，H 表示程序中包含目标区域的函数从入口基本块到目标区域

计算机工程 www.ecice06.com

的距离。如图6中基本块D到目标区域的距离,利用式(2)可以求得该距离。集合B_t由该函数包含的目标基本块组成,b_{1st}为入口基本块。

$$R(b_{1st}, B_t) = \sum_{b \in B_t} [L(b_{1st}, b)^{-1}]^{-1} \quad (2)$$

4种情况讨论如下:

情况1 当基本块为目标基本块时,该基本块到目标区域的距离为0。

情况2 当基本块与目标区域属于同一函数但不是目标基本块时,如图6中的基本块C。利用式(1)分别计算基本块C到每个目标基本块的距离后,求调和平均值作为基本块C到目标区域的距离。

情况3 当基本块与目标区域属于不同函数但可以直接通过函数调用到达目标区域所在函数时,如图6中基本块A。用基本块A调用的系列函数的函数路径长度的倒数和加上H作为基本块A到目标区域的距离,即图6中函数F_2与F_3的函数路径长度的倒数和与函数F_4的H之和,如果有多条调用路径则取平均值作为该基本块的基本块距离。

情况4 当基本块与目标区域属于不同函数且可以通过同属一个函数内的其他基本块到达目标区域所在函数时,如图6中基本块B通过基本块A可以到达包含目标区域的函数F_4。计算出基本块A到目标区域的距离,在用该距离加上基本块A到基本块B的距离,作为基本块B到目标区域的距离。

式(3)为基本块距离计算公式,集合T_b由目标基本块组成,集合T_a由与目标基本块同属一个函数的基本块组成,集合T_f由可以直接到达包含目标区域的函数的基本块组成,集合T_j由可以通过T_f中的元素到达包含目标区域的函数的基本块组成。集合F_i由基本块i到包含目标区域的函数所需要调用的函数组成,$\mathrm{Num}(F_i)$表示集合F_i中元素的数量,P_f表示函数f的函数路径长度。

$$P(i, T_b) = \begin{cases} 0,\ i \in T_b \\ \sum_{b \in T_b} [L(i, b)^{-1}]^{-1},\ i \in T_a \\ H + \dfrac{\sum_{f \in F_i} p_f}{\mathrm{Num}(F_i)},\ i \in T_f \\ L(i, d) + P(d, T_b), i \in T_j,\ d \in T_f \end{cases} \quad (3)$$

在静态分析阶段利用式(3)完成对基本块距离的计算,计算过程如算法1所示。

算法1 基本块距离计算算法

输入 程序控制流图(CFG),函数调用流图(CG),目标基本块集合(T)

输出 基本块距离

```
1.While function in CG://对程序每一个函数进行操作
2.While BB in function://对函数中每一个基本块进行//操作
3.Calculate_weights(BB);//计算基本块权重
4.if BB in T://BB为目标基本块
5.Fun=function;//记录这个函数
6.Calculate_Pf(function);//计算函数路径长度
7.Assignment();//将权重赋值到CFG与CG图中
8.H=Calculate_H(Fun);//计算Fun入口基本块到T的//距离
9.While function in CG://对程序每一个函数进行操作
10.While BB in function://对函数中每一个基本块进行//操作
11.Calculate(BB);//根据式(3)计算基本块距离
```

对算法1的代价进行分析,假设程序有m个函数和n个基本块,符号K_i表示函数i包含的基本块数量。对算法中第一个双层循环进行分析(算法第1行~第6行),由函数路径长度计算方法可知,Calculate_P_f(function)相当于对函数function包含的基本块进行一次遍历,代价为$O(k_{\mathrm{function}})$,而第二层循环(算法第2行~第5行)代价也为$O(k_{\mathrm{function}})$,所以算法1中第一个双层循环总执行次数相当于程序中每个函数包含的基本块数量之和的两倍,故代价为$O(n)$。对算法1第二个双层循环(第9行~第11行)进行分析,由式(3)可知Calculate(BB)的代价为$O(m)$,故第二个双层循环的代价为$O(m \times n)$。取两个双层循环代价之和作为算法1的总代价,总代价为$O((m+1) \times n)$。多数应用程序中基本块数量远远大于函数数量,$m+1$可以看为常数项,所以距离计算花费的代价为$O(n)$。

3.3 种子距离与能量计算

在3.2节中得到了程序中任意基本块到目标区域的距离计算公式。在对程序静态分析时利用公式计算出每一个基本块到目标区域的距离并插桩到程序中。在模糊测试时追踪每个种子执行的轨迹,根据种子执行到的基本块计算该种子到目标区域的距离,如式(4)所示:

$$d(s, T_b) = \frac{\sum_{i \in Q} P(i, T_b)}{\mathrm{Num}(Q)} \quad (4)$$

其中,集合Q由种子s覆盖的基本块构成,$\mathrm{Num}(Q)$表示集合Q内元素的数量。

对种子距离进行归一化得到式(5),集合S由模糊测试器使用的种子构成:

$$\tilde{d}(s, T_b) = \frac{d(s, T_b) - \min D}{\max D - \min D} \quad (5)$$

$\max D$定义如式(6)所示:

$$\max D = \max_{s' \in S} [d(s', T_b)] \quad (6)$$

$\min D$定义如式(7)所示:

$$\min D = \min_{s' \in S} [d(s', T_b)] \quad (7)$$

在模糊测试中种子能量是用来调控一个种子变异次数的重要变量,如果一个种子与模糊测试器的测试策略契合度越高,那么该种子得到的能量也越高,在种子进行变异时模糊测试器会根据种子的能

计算机工程 www.ecice06.com

量对种子进行变异操作，种子能量越高变异生成的测试用例越多。

为避免种子在模糊测试一开始就收敛到某一条路径上，将模糊测试执行的轮次t作为调节种子能量的因子。使得模糊测试在初始的几轮测试中能够探索足够多的路径，避免模糊测试路径过度集中。将Afl能量计算方法[18]与式(7)结合可以得到种子能量计算公式：

$$E(s,T_b)=A(s)\times\left(\left(1-\frac{1}{t}\right)\times\widetilde{d}(s,T_b)+\frac{1}{t}\right) \tag{8}$$

其中，$A(s)$表示在Afl能量计算中种子s的能量。

当t趋近正无穷时，$E(s,T_b)=A(s)\times\widetilde{d}(s,T_b)$；当$t=1$时，$E(s,T_b)=A(s)$。当模糊测试刚开始时种子的距离对能量计算不会产生影响，而随着迭代次数的增加，种子距离对能量的影响逐渐增强。图7为不同迭代次数对种子能量变化的影响，可以看到当种子距离一定时，随着迭代次数的增加，种子能量在初始的几轮迭代中急剧变化后迅速趋于稳定，这证明了迭代次数t仅在测试开始阶段影响种子能量计算，随着迭代次数的不断增加，t对种子能量影响逐渐降低，种子距离在种子能量计算中起主导作用。

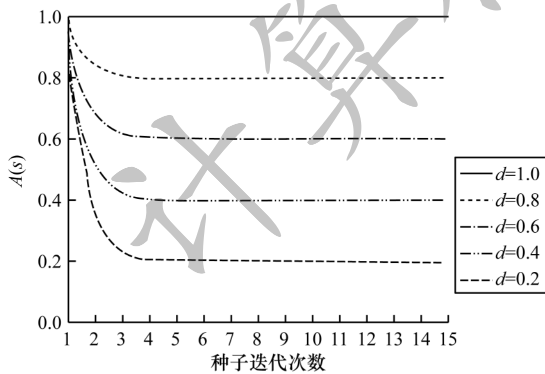

图7 迭代次数对种子能量的影响

Fig.7 Effect of iteration times on seed energy

4 实验与评估

为验证本文方法的有效性，基于Afl[18]设计实现了原型系统Afl-guide，并与Aflgo、Afl进行对比实验。Afl-guide使用LLVM[19]对目标程序进行分析生成函数调用图和函数控制流程图，使用Python脚本利用Networkx包实现对函数调用图和函数控制流程图的解析并计算基本块距离。扩展Afl的插桩模块，将基本块距离插桩到程序中。

本文结合Aflgo，选取libming与GNU Binutils作为测试程序。libming是一款使用C语言编写的Flash（SWF）输出库，可在C ++、PHP、Python、Ruby和Perl中使用，拥有10万行的代码，是被广泛使用的开源项目。GNU Binutils是GNU/Linux平台中使用的二进制分析工具集，有着近100万行的代码，被广泛用于对模糊测试工具的测试中[20-21]。

实验选取libming和GNU Binutils中近年公开的CVE作为导向目标，分别用3个工具对目标程序进行实验，每组实验重复5次，持续24 h。实验结果如表1和表2所示，其中，Arrive表示到达目标点的次数，TTE表示第一次覆盖到目标站点花费的时间，Improve为改善因子，值为Afl-guide的TTE与Aflgo和Afl的TTE的商，表示相较于Afl与Aflgo，Afl-guide的效率提升了多少。实验服务器设置为：AMD ryzen7 2700处理器，64 GB内存，2 TB固态硬盘，运行Ubuntu 16.04（64 bit）操作系统。

表1 libming目标站点覆盖

Table 1 libming target site coverage

CVE-ID	工具	Arrive	TTE/s	Improve
2018-7867	Afl-guide	5	17 726	—
	Aflgo	5	30 603	1.72
	Afl	3	63 055	3.56
2018-8807	Afl-guide	5	29 105	—
	Aflgo	3	43 436	1.49
	Afl	2	67 714	2.33
2018-8962	Afl-guide	5	14 484	—
	Aflgo	5	17 252	1.19
	Afl	4	45 022	3.11
2019-12982	Afl-guide	5	434	—
	Aflgo	5	1 563	3.60
	Afl	5	3 674	8.47
2020-6689	Afl-guide	5	322	—
	Aflgo	5	436	1.35
	Afl	5	1 684	5.23

表2 GNU Binutils目标站点覆盖

Table 2 GNU Binutils target site coverage

CVE-ID	工具	Arrive	TTE/s	Improve
2016-4487	Afl-guide	5	763	—
	Aflgo	5	1 148	1.51
	Afl	5	1 786	2.34
2016-4489	Afl-guide	5	3 815	—
	Aflgo	5	4 019	1.05
	Afl	5	7 493	1.96
2016-4490	Afl-guide	5	285	—
	Aflgo	5	264	0.93
	Afl	5	197	0.69
2016-6131	Afl-guide	5	27 407	—
	Aflgo	5	31 409	1.15
	Afl	2	40 853	1.49

在表1、表2的数据中，除CVE-2016-4490外，Afl-guide与Aflgo到达目标区域的时间明显小于

计算机工程 www.ecice06.com

Afl,证明了Afl-guide与Aflgo的导向性。在CVE-2016-4490中,目标站点在程序路径的浅层容易被测试用例覆盖,因此导向式模糊测试器在测试的过程中引入的资源消耗反而导致Afl-guide与Aflgo覆盖目标站点的速度慢于Afl。通过表1和表2中的数据还可以看出,利用本文方法实现的工具Afl-guide在导向性上优于Aflgo。在CVE-2018-7867中提升最为显著,Afl-guide到达目标点仅用了Aflgo花费时间的27.8%。实验结果表明,使用本文的方法可以快速生成覆盖程序指定位置的测试用例,能够大幅提高在已知程序脆弱区域、补丁检测等情景下的模糊测试效率。

为进一步说明在种子能量计算过程中引入种子迭代次数的对路径收敛速度的影响。Aflgo与Afl-guide路径覆盖率比对情况如表3、表4所示。其中,Cov表示第一次覆盖到目标区域时模糊测试器的路径覆盖率。AveC表示Cov与TTE的比值,Better值为Aflgo的AveC与Afl-guide的AveC的商,表示相较于Aflgo的覆盖率提升了多少。

表3 libming路径覆盖率

Table 3 libming path coverage

CVE-ID	工具	Cov/%	AveC	Better
2018-7867	Afl-guide	6.14	3.46×10^{-6}	1.63
	Aflgo	6.50	2.12×10^{-6}	—
2018-8807	Afl-guide	6.48	2.23×10^{-6}	1.43
	Aflgo	6.76	1.56×10^{-6}	—
2018-8962	Afl-guide	6.83	4.72×10^{-6}	1.20
	Aflgo	6.77	3.92×10^{-6}	—
2019-12982	Afl-guide	3.06	7.05×10^{-5}	2.58
	Aflgo	4.37	2.8×10^{-5}	—
2020-6689	Afl-guide	3.19	9.91×10^{-5}	1.31
	Aflgo	3.30	7.57×10^{-5}	—

表4 GNU Binutils路径覆盖率

Table 4 GNU Binutils path coverage

CVE-ID	工具	Cov/%	AveC	Better
2016-4487	Afl-guide	4.33	5.67×10^{-5}	1.47
	Aflgo	4.41	3.84×10^{-5}	—
2016-4489	Afl-guide	9.31	2.44×10^{-5}	1.04
	Aflgo	9.37	2.33×10^{-5}	—
2016-4490	Afl-guide	2.72	9.78×10^{-5}	1.08
	Aflgo	2.38	9.02×10^{-5}	—
2016-6131	Afl-guide	11.31	4.13×10^{-6}	1.11
	Aflgo	11.67	3.72×10^{-6}	—

从表3、表4可以看出,Afl-guide与Aflgo在到达目标位置时路径覆盖率基本一致,但考虑到所花费的时间,单位时间内Afl-guide的路径覆盖要高于Aflgo。证明在种子能量计算中引入种子迭代次数达到了预期的目标,在测试初期快速探索路径并随着轮次的增加快速收敛到目标区域。

5 结束语

本文对导向式灰盒模糊测试进行研究,提出一种距离与权重相结合的导向式灰盒模糊测试方法。对距离计算方法进行改进,提高了距离计算的精确性,增强模糊测试器的导向性。实验结果表明,该方法能够有效提高模糊测试导向的效率以及对目标区域覆盖的概率。由于现在的静态分析工具还无法有效识别程序中的间接调用,导致函数距离计算中存在一定的误差,同时程序中存在校验和的情况使得种子无法覆盖到目标区域。下一步将对静态分析方法进行改进,以提高函数距离的精度,并将符号执行与模糊测试相结合,使模糊测试可以对校验和程序进行快速导向。

参考文献

[1] CHEN Hongxu,XUE Yinxing,LI Yuekang,et al. Hawkeye:towards a desired directed grey-box fuzzer[C]//Proceedings of 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto,Canada:ACM Press,2018:2095-2108.

[2] CRAWFORD J B. A survey of some free fuzzing tools[EB/OL].(2018-01-17)[2020-01-20]. https://lwn.net/Articles/744269/.

[3] MILLER B P,FREDRIKSEN L,SO B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM,1990,33(12):32-44.

[4] GODEFROID P,LEVIN M Y,MOLNAR D. SAGE:Whitebox fuzzing for security testing[J]. Communications of the ACM,2012,55(3):40-44.

[5] ZOU Quanchen,ZHANG Tao,WU Runpu,et al. From automation to intelligence:software vulnerabilities mining technology progress[J]. Journal of Tsinghua University(Science and Technology),2018,58(12):1079-1094.(in Chinese)
邹权臣,张涛,吴润浦,等. 从自动化到智能化:软件漏洞挖掘技术进展[J]. 清华大学学报(自然科学版),2018,58(12):1079-1094.

[6] REN Yuzhu,ZHANG Youwei,AI Chengwei. Review of stain analysis technology research[J]. Journal of Computer Applications,2019,39(8):2302-2309.(in Chinese)
任玉柱,张有为,艾成炜. 污点分析技术研究综述[J]. 计算机应用,2019,39(8):2302-2309.

[7] CHEN Jianmin,SHU Hui,XIONG Xiaobing. Fuzzing test method based on symbolic execution[J]. Computer Engineering,2009,35(21):33-35.(in Chinese)
陈建敏,舒辉,熊小兵. 基于符号化执行的Fuzzing测试方法[J]. 计算机工程,2009,35(21):33-35.

[8] ZHANG Lin,ZENG Qingkai. Static detection technology of software security vulnerabilities[J]. Computer Engineering,2008,34(12):157-159.(in Chinese)
张林,曾庆凯. 软件安全漏洞的静态检测技术[J]. 计算机工程,2008,34(12):157-159.

www.ecice06.com 计算机工程

[9] WANG Tielei, WEI Tao, GU Guofei, et al. TaintScope: a checksum-aware directed fuzzing tool for automatic software vulnerability detection[C]//Proceedings of 2010 IEEE Symposium on Security and Privacy. Berkeley/Oakland, USA: IEEE Press, 2010: 497-512.
[10] HALLER I, SLOEINSKA A, NEUGSCHWANDTNER M, et al. Dowsing for over flows: a guided fuzzer to find buffer boundary violations[C]//Proceedings of the 22nd USENIX Conference on Security. Washington D. C., USA: USENIX Association, 2013: 49-64.
[11] NEUGSCHWANDTNER M, MILANI C P, Haller I, et al. The BORG: nanoprobing binaries for buffer overreads[C]//Proceedings of the 5th ACM Conference on Data and Application Security and Privacy. New York, USA: ACM Press, 2015: 87-97.
[12] BOHME M, PHAM V T, ROYCHOUDHURY A. Coverage-based greybox fuzzing as Markov chain[J]. IEEE Transactions on Software Engineering, 2017, 45(5): 489-506.
[13] WANG J, CHEN B, WEI L, et al. Skyfire: data-driven seed generation for fuzzing[C]//Proceedings of IEEE Symposium on Security and Privacy. San Jose, USA: IEEE Press, 2017: 579-594.
[14] RAWAT S, JAIN V, KUMAR A, et al. VUzzer: application aware evolutionary fuzzing[C]// Proceedings of NDSS'17. San Diego, USA: [s. n.], 2017: 1-14.
[15] BOHME M, PHAM V T, NGUYEN M D, et al. Directed greybox fuzzing[C]//Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, USA: ACM Press, 2017: 2329-2344.
[16] DAI Wei, LU Yuliang, ZHU Kailong. Guided grey box fuzzy testing technology combined with mixed symbol execution[J]. Computer Engineering, 2020, 46(8): 190-196. (in Chinese)
戴渭,陆余良,朱凯龙. 结合混合符号执行的导向式灰盒模糊测试技术[J]. 计算机工程,2020,46(8):190-196.
[17] MARINESCU P D, CADAR C. KATCH: high-coverage testing of software patches[C]//Proceedings of the 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg, Russian Federation: [s. n.], 2013: 235-245.
[18] ZALEWSKI M. American fuzzy lop[EB/OL]. [2020-01-20]. https://www. cnblogs. com/0xHack/p/9414444. html.
[19] LATTNER C, ADVE V. LLVM: a compilation framework for lifelong program analysis & transformation[C]// Proceedings of International Symposium on Code Generation and Optimization. San Jose, USA: [s. n.], 2004: 75-86.
[20] BOHME M, PHAM V T, ROYCHOUDHURY A. Coverage based greybox fuzzing as Markov chain[C]// Proceedings of ACM SIGSAC Conference on Computer and Communications Security. New York, USA: ACM Press, 2016: 1032-1043.
[21] LEMIEUX C, SEN K. FairFuzz: targeting rare branches to rapidly increase greybox fuzz testing coverage[EB/OL]. [2020-01-20]. https://arxiv. org/pdf/1709. 07101. pdf.

编辑 索书志

计算机工程 06.com