文章编号:1009-3087(2016)02-0132-07 DOI:10.15961/j.jsuese.2016.02.019

基于改进轮盘赌策略的反馈式模糊测试方法

蔡 军[1],邹 鹏[1],沈弼龙[2],何 骏[1]
(1.装备学院 复杂电子系统仿真实验室,北京 101416;2.清华大学 计算机科学与技术系,北京 100084)

摘 要:为了解决传统模糊测试方法测试盲目效率低下的问题,提出一种反馈式模糊测试方法。该方法通过对传统模糊测试方法的流程进行优化来提高测试效果。通过网络爬虫收集原始样本文件,确保原始样本的多样性。通过代码覆盖率分析来从大量样本文件中筛选用于变异的最佳种子文件,在保证足够的测试空间的同时避免冗余测试。通过对种子文件进行变异来生成测试用例,并根据测试结果基于改进轮盘赌策略不断调优对种子文件的变异参数,以期发现尽可能多的软件故障。实现了一个原型系统OSSRWSFuzzer,在实验中发现了WPS Office for Linux软件的56个故障,其测试效果明显优于现有模糊测试工具。

关键词:反馈式模糊测试;种子优选;轮盘赌选择
中图分类号:TP309 **文献标志码**:A

Feedback Fuzzing Based on Improved Roulette Wheel Selection Strategy

CAI Jun[1], *ZOU Peng*[1], *SHENG Bilong*[2], *HE Jun*[1]
(1. Sci. and Technol. on Complex Electronic System Simulation Lab., Academy of Equipment, Beijing 101416, China;
2. Dept. of Computer Sci. and Technol., Tsinghua Univ., Beijing 100084, China)

Abstract: In order to solve the problem of blindness and poor efficiency of traditional fuzzing, a feedback fuzzing method was proposed. This method can improve the test effect by optimizing the process of traditional fuzzing. Firstly, original sample files were collected via a web crawler to ensure the diversity of them. Then, the best seed file was selected from a large number of sample files, to avoid redundant testing while ensuring adequate testing space. Finally, test cases were generated by mutating the seed file, and the mutation parameters was continuously adjusted according to the testing result based on improved roulette wheel selection strategy, in order to find as many software failures as possible. A prototype system named OSSRWSFuzzer was implemented, which had found 56 failures of WPS Office for Linux in experiments, and its test effect is obviously better than that of the existing fuzzers.

Key words: feedback fuzzing; optimizing seed selection; roulette wheel selection

随着软件的使用越来越广泛,其安全性也变得越来越重要。事实上,大部分网络攻击都是基于软件漏洞来实施的,一旦那些高危的软件漏洞被利用,将会带来不可估量的损失。模糊测试是一种出现较早的软件漏洞检测技术[1],其基本思想为:向待测程序提供大量特殊构造的或是随机的数据作为输入,监视程序运行过程中的异常并记录导致异常的输入,辅助以人工分析,基于导致异常的输入进一步定位软件中漏洞的位置[2]。

与目前新兴的漏洞检测技术如符号执行[3-5]、污点析[6-8]相比,模糊测试在很大程度上是一种强制性的技术,简单且有效,缺点是测试存在较大的盲目性。然而,一个好的模糊测试工具在现代软件的安全漏洞测试中仍然具有较高的使用价值。符号执行和污点分析是近几年新兴的漏洞检测技术,虽然在思想上更先进,但是到目前为止,由于它们自身的一些缺陷,尚不能很好地适用于大型应用软件,至今仍不能完全取代模糊测试技术。

收稿日期:2015-10-15
基金项目:国家高技术研究发展计划资助项目(2012AA012902);"核高基"国家科技重大专项基金资助项目(2013ZX01045-004)
作者简介:蔡 军(1982—),男,博士生.研究方向:信息安全;网络安全;软件安全.E-mail:cjgfkd@163.com
网络出版时间:2016-3-15 11:17:35 **网络出版地址**:http://www.cnki.net/kcms/detail/51.1596.T.20160315.1117.001.html

近年来,针对模糊测试方法的研究大多是通过使用其他漏洞检测方法来辅助模糊测试,综合多种方法的优势来提高模糊测试的漏洞检测效果。文献[9]提出了一种基于污点分析的智能模糊测试方法,该方法首先通过污点分析来识别输入文件中影响目标程序安全敏感操作的字节,然后通过集中对敏感字节进行变异来生成测试用例,在一定程度上克服了模糊测试的盲目性;不足之处在于对于不同的文件,敏感字节的位置会发生变化,需要多次进行污点分析。文献[10]提出了一种混合模糊测试方法,该方法利用符号执行遍历不同的路径,再利用模糊测试快速测试每一条路径,提高了测试的代码覆盖率;不足之处在于该方法尚处于实验阶段,还没有应用到真实应用程序。文献[11]提出了一种校验和感知的模糊测试方法,该方法综合运用符号执行和污点分析方法来定位目标程序中的校验和检测点,并自动修复畸形样本中的校验和域来绕过程序中的校验和检查机制,取得了较好的测试效果;不足之处在于未考虑代码覆盖率,且对未知格式规范的文件测试效果有限。本文试图从流程优化的角度来提高模糊测试的效率,设计了一套全新的模糊测试流程,并将遗传算法中的轮盘赌选择算子运用到了测试过程中,提出了一种基于改进轮盘赌策略的反馈式模糊测试方法,并实现了一个原型系统 OSSRWSFuzzer。

1　背　景

1.1　典型模糊测试工具 zzuf

zzuf[12]是一个标准的基于变异的模糊测试工具,采用的是随机化的变异策略,它通过改变种子文件的部分比特来生成测试用例。zzuf 的行为通过一系列命令行参数来控制,最常用的参数有2个:一个是随机种子(random seed),用 s 表示,随机种子是 zzuf 的随机数发生器的初始值,默认为0,s 不同产生的随机数也不同;另一个是模糊率(fuzzing ratio),用 r 表示,模糊率是 zzuf 改变的比特的比例,默认的模糊率是0.004,指模糊0.4% 的比特。s 和 r 的值也可以是一个范围,如 $s = 1 \sim 1\,000$,$r = 0.01 \sim 0.1$,指 s 的值从1逐步递增到1 000(每次增加1),这样一共要测试1 000次,每次从0.01 ~ 0.1之间随机的选一个数作为 r。

zzuf 有3个重要特征:1)可重现性。给定相同的参数,zzuf 生成的测试用例也相同,参数不变,再次执行得到的结果也不变;2)一致性。数据对于不同的应用程序采用相同的模糊方式;3)可以不调用应用程序直接模糊文件,且创建的文件也可以重现。

1.2　轮盘赌选择算子

轮盘赌选择算子[13-14]是遗传算法的一种常用的选择算子,其基本原理是个体适应度按比例转换为选择概率,按选择概率所占的比值在一圆盘上进行比例划分,每次转动圆盘后待圆盘停止时指针停靠扇区对应的个体即为选中的个体。由于适应值大的个体对应的扇区面积也大,因此它被选中的概率也大。设群体大小为 n,标准的轮盘赌选择过程如下[15-17]:

Step 1:计算每个个体的适应度 F_i。

Step 2:计算每个个体被选中遗传到下一代群体的概率 P_i:

$$P_i = \frac{F_i}{\sum_{i=1}^{n} F_i},\ i = 1,2,\cdots,n \tag{1}$$

根据 P_i 把一个圆盘分成 n 个扇形区间,扇形区间的中心角与 P_i 成正比。

Step 3:使用模拟赌盘操作,产生一个[0,1]内的随机数 a,如果 $\sum_{j=1}^{i-1} P_j \le a \le \sum_{j=1}^{i} P_j$,选择个体 i 进入子代种群;

Step 4:重复 Step 3 步骤 n 次,得到的个体构成新一代种群。

2　反馈式模糊测试方法

反馈式模糊测试是指利用前面模糊测试的效果反馈来指导改进后面的模糊测试。OSSRWSFuzzer 系统实现了本文提出的基于改进轮盘赌策略的反馈式模糊测试方法,它以二进制插桩平台 Pin[18]和模糊测试工具 zzuf[12]为基础开发,使用了 Python 和 C 语言,其中,Python 代码用于实现流程控制,C 代码用于实现覆盖率分析和异常分析。OSSRWSFuzzer 系统工作流程如图1所示,主要分为4个步骤:1)通过网络爬虫从互联网上收集测试目标程序的原始样本文件;2)基于动态二进制插桩测量每个样本文件的代码覆盖率,选择覆盖率最高且文件大小最小的样本文件作为种子文件;3)使用 zzuf 对种子文件实施变异来生成测试用例,交给目标程序执行,同时根据测试效果基于改进的轮盘赌策略实时调整变异参数,并监视和收集异常;4)基于动态污点分析方法进一步分析导致异常的原因,定位漏洞位置。其中,第2)步种子文件选择和第3)步反馈式模糊测试是

本文讨论的重点,第4)步异常分析和漏洞定位将在其他文章讨论。

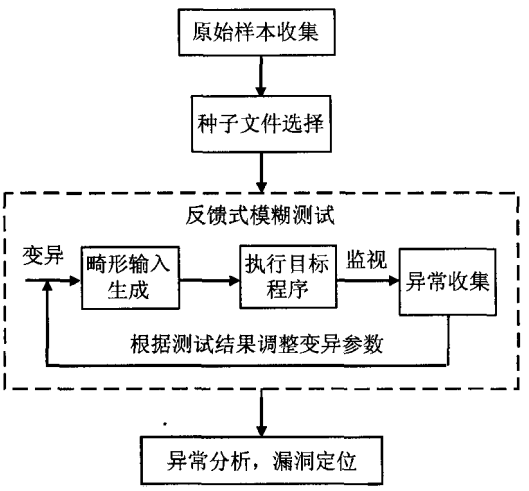

图1 OSSRWSFuzzer工作流程

Fig. 1 Workflow of OSSRWFFuzzer

2.1 基于代码覆盖率分析的种子优选

样本文件是基于变异的模糊测试的基础。给定一个目标程序,在进行模糊测试前,首先需要识别目标程序支持的文件格式类型,并准备相应格式的样本文件(能被目标程序正常处理的文件)。OSSRWSFuzzer借助商业软件Website Ripper Copier(WRC)[19]来收集样本文件,WRC是一款功能强大的网络爬虫软件,只要给它一个网站地址,并配置要抓取的文件的大小和扩展名范围,它就能自动从网站上抓取所需文件。由于互联网上资源极其丰富,通过从互联网收集大量样本文件,确保了样本文件的多样性,扩大了测试空间。

显然,不可能将所有样本文件都作为种子文件来进行变异,而且2个不同的文件在模糊测试时也可能出现相同的行为,没必要进行重复测试。因此,为提高效率,需要从众多样本文件中选出最具代表性的文件来作为种子文件。

软件是由代码编译而来的,软件从每一次打开到关闭,都会执行一定的代码,这些执行到的代码只占软件代码的一部分。例如,对于文件处理类程序,在打开不同的文件时,其执行到的代码量也不尽相同。将软件在一次测试中执行到的代码量占软件源代码总量的百分比称为这次测试的代码覆盖率。理论上,如果能让软件在一次测试中执行到它的所有代码,就能触发该软件的所有漏洞。当然,这只是理想情况,在现实中基本不可能达到100%的代码覆盖率,只能尽可能地提高测试的代码覆盖率。

OSSRWSFuzzer通过测试目标程序在打开样本文件时的代码覆盖率来选择种子文件,遵循2条原则:一是,代码覆盖率高的文件优先选择;二是,多个文件代码覆盖率相同时,大小最小的文件优先选择,因为小的文件需要的测试开销更小。OSSRWSFuzzer将样本文件按照扩展名进行分类,测试并记录目标程序在打开某类文件时的代码覆盖率,同时记录文件的大小,按照代码覆盖率的高低对样本文件进行排序,选择代码覆盖率最高且文件大小最小(有重复时取最小)的文件作为该类文件的种子文件。

OSSRWSFuzzer的覆盖率分析模块基于二进制插桩平台Pin来实现,其基本原理是在程序执行过程插入分析代码来记录目标程序处理样本文件时执行到的基本块(basic block,简称BBL)的数量,以该数量的大小来反应代码覆盖率的高低。Pin是Intel公司研发的一个动态二进制插桩平台,与其他二进制插桩平台如Valgrind[20]、DynamoRio[21]相比,Pin提供了更丰富的用户API接口和详细的说明文档,用户利用这些API可以方便地编写自己的插桩工具。OSSRWSFuzzer主要通过实施2种插桩来实现覆盖率分析逻辑。一是,利用"TRACE_AddInstrumentFunction"插桩程序轨迹(Trace),利用"TRACE_BblHead"、"BBL_Valid"和"BBL_Next"来遍历轨迹中的基本块,记录基本块的地址、指令数量等信息,并统计基本块的数量;二是,利用"IMG_AddInstrumentationFunction"插桩模块(Image),记录模块的名字、最低地址、最高地址等信息。对于Linux操作系统,如果模块的名字中包含".so"和".so.",则该模块为共享库,如果某个基本块的地址在这个模块的最低地址和最高地址之间,则该基本块属于共享库的基本块,不进行计数统计。

2.2 基于改进轮盘赌策略的反馈式模糊测试

使用模糊测试挖掘漏洞的首要任务就是要找到导致程序异常的畸形输入,通常一个异常就可能对应一个漏洞。发现的不同异常的数量越多,说明模糊测试的效果越好。但是,随着软件出厂前安全测试的增强,要发现一个导致现代软件异常的输入并不容易。本文所提反馈式模糊测试是一个循环往复的过程,目的是通过不断的测试、调整、再测试、再调整,找到最佳的变异参数,发现最多的异常。

OSSRWSFuzzer使用模糊测试工具zzuf来对种子文件实施变异,变异策略的调整等价于zzuf输入参数的调整。本文重点研究模糊率r的调整,调整r也就是调整种子文件的变异范围。

2.2.1　问题建模

通过测试不同的模糊测试工具,观察到:在对样本文件进行变异时,如果文件中的过多的比特被改变了,则改变后的文件格式将不再被目标程序认为是合法的。目标程序可能在处理导致它异常的数据之前抛弃非法文件,则这样的变异将毫无意义;同时,如果变异的比特数过少,会导致搜索的可能的测试用例空间不够。因此,为提高效率,需要找到最佳的 r 的取值范围。

在实验中还观察到变异 0.5% 和变异 1% 有明显的差别,而变异90% 和90.5% 的差别很小。因此,将 r 的取值范围0.0 ~ 1.0 按等比数列划分区间,取公比为2,从变异1 ~ 2个比特开始,接着2 ~ 4、4 ~ 8,以此类推。假定种子文件的大小为 x 个比特,则将 0.0 ~ 1.0 划分为 $N = [\mathrm{lb}\ x]$ 个区间,相应的 r 的取值范围为 $1/x \sim 2/x$,$2/x \sim 4/x$,以此类推。将这 N 个区间依次编号为第 1 ~ N 号区间。

刚开始并不知道这 N 个区间中哪一个是最佳区间(r 的取值为这个区间时,最容易发现异常),需要通过某种方法来找到这个区间。下面定义几个概念:

单次测试:每生成一个测试用例(对种子文件变异一次)称为一次测试。

单次测试任务:每调用一次 zzuf 称为一次测试任务,一次测试任务包含多次单次测试,例如某次调用 zzuf 时设定参数 s 的值为 1 ~ 1 000,则该次测试任务将测试 1 000 次;

一轮测试:完成 N 次测试任务称为一轮测试。

异常密度:对于某个区间,导致软件异常的测试次数与在该区间上进行的总的测试次数之比。

选择概率:在一轮测试中,选中某个区间作为参数执行单次测试任务的概率。

本文的目的就是要根据异常密度为各个区间分配最佳的选择概率,一个好的分配策略应遵循以下3条原则:

1)某个区间的选择概率应与该区间的异常密度成正比,异常密度大的选择概率也大,异常密度相同的选择概率也相同。

2)如果在某个区间上没有发现异常,该区间不能被完全抛弃,因为继续在该区间上测试仍有可能出现异常。

3)如果在 2 个不同区间上观测到的异常次数均为 0,则测试次数少的区间应该比测试次数多的区间优先选择。

2.2.2　改进的轮盘赌策略

OSSRWSFuzzer 使用改进的轮盘赌策略来作为上述问题的解决方案。将 N 个区间作为遗传算法的初始群体,将异常密度作为每个个体的适应度。虽然标准的轮盘赌策略能够让适应度大的个体具有更多的选择机会,也能让适应度小的个体获得一定的选择机会,但是由于使用随机数来做选择,操作的随机性比较大,有时会使种群出现“退化”现象,即适应度比较高的个体失去选择的机会,最终使得种群的进化产生剧烈的震荡。因此,根据 2.2.1 节提出的3条原则,对标准轮盘赌策略做了适当的改进。改进的轮盘赌策略的具体过程如下:

Step 1:根据种子文件的大小 x 为 r 的取值范围 0.0 ~ 1.0 划分区间,区间个数为 $N = [\mathrm{lb}\ x]$,第1个区间为 $1/x \sim 2/x$,第2个区间为 $2/x \sim 4/x$,以此类推,最后一个区间为 $2^N/x \sim 1$。

Step 2:在每个区间执行一次测试任务,测试次数均为 t_0,观测它们的异常次数 n_{i0},则每个区间的初始适应度 F_{i0} 为:

$$F_{i0} = \frac{n_{i0}}{t_0}, i = 1,2,\cdots,n \tag{2}$$

若 n_{i0} 为0,则假定下一次测试异常就会发生,相应的取 $F_{i0} = 1/(t_0 + 1)$。

Step 3:计算每个区间的初始选择概率 P_{i0}:

$$p_{i0} = \frac{F_{i0}}{\sum_{i=1}^{n} F_{i0}},\ i = 1,2,\cdots,n \tag{3}$$

根据 P_{i0} 构造初始轮盘。

Step 4:产生 N 个随机数,执行一轮测试,记录本轮测试中每个区间测试次数 t_i 和出现的异常次数 n_i,同时记录每个区间的累积测试次数 t_{is} 和累积异常次数 n_{is}。需要注意的是 n_{is} 的数值远小于 t_{is}。

Step 5:更新每个区间的适应度 F_i:

$$F_i = \begin{cases} \dfrac{n_i}{t_i}, n_i > 0, t_i > 0; \\ \dfrac{n_{is}}{t_{is}}, n_i = 0, n_{is} > 0; \\ \dfrac{1}{t_{is} + 1}, n_{is} = 0 \end{cases} \tag{4}$$

其中,$i = 1,2,\cdots,n$。

分为3种情况:1)本轮测试中有异常;2)本轮测试中没有异常但历史上有异常,包括本轮测试没有被选择到的情况;3)历史上都没有异常。

Step 6:按照式(1)更新每个区间的选择概率,

构造新的轮盘。

Step 7:重复上述步骤,从步骤 Step 4 开始。

改进的轮盘赌策略既保证了优良的个体能够以较高的概率进入下一代种群,也让适应度低的个体具有一定的选择机会,不会丢失个体,保证了种群的完整性和多样性,满足了之前提出的3条原则。

在运用了改进的轮盘赌策略后,OSSRWSFuzzer能够根据测试结果自动调整变异参数,使容易出现异常的区间多测试,也让不容易出现异常的区间获得一定的测试机会,提高了发现异常的效率。

3 实验分析

实验分为2部分:一是,测试 OSSRWSFuzzer 的种子文件优选能力;二是,测试 OSSRWSFuzzer 的漏洞检测能力。测试对象为 Linux 系统上运行的文件处理类软件,测试平台为 Ubuntu12.04 32 位系统,Intel i5-4200 M 处理器,4 GB 内存。

3.1 种子文件优选实验

进行种子文件优选的目的是为了从大量样本文件中挑选出最有代表性的文件作为种子文件,避免重复测试,提高效率。样本文件的好坏是相对于具体软件而言的,假设有2款软件都能打开这类文件,一个样本文件被选作其中一款软件的种子文件时,不一定还能被选作另一款软件的种子文件。使用 Website Ripper Copier 在百度、腾讯、新浪等网站上爬行了10 h,一共爬取了7类共9 133个文件,使用这些文件对7款软件进行了测试,实验结果如表1所示。

表1 种子文件优选实验结果

Tab.1 Experimental results of seed file selection

被测软件名称版本	样本文件格式	样本文件数量	最大基本块覆盖数	最小基本块覆盖数	种子文件大小/B
eog 3.4.2	gif	380	5 371	4 359	7 236
et 8.1.0.3724	xls	1 163	11 758	7 068	6 656
feh 2.12	jpg	3 632	1 107	815	37 462
flashplayer 11.2.202.442	swf	171	72 599	20 016	354 661
shotwell 0.12.3	png	615	18 391	9 976	1 411
wpp 8.1.0.3724	ppt	247	10 018	9 953	51 200
wps 8.1.0.3724	doc	2 925	11 012	7 819	690 688

从实验结果可以看出,并不是文件越大,其代码覆盖率就越高,且不同的文件代码覆盖率相同的情况也很多。以看图软件 shotwell 为例,样本文件为615个 png 文件,经测量得到的最大基本块覆盖数为18 391个,最小基本块覆盖数为9 976个,且基本块覆盖数为18 391的样本文件有5个:"log_weixin. png"、"log_renren. png"、"log_qq. png"、"log_weibo. png"、"pw-logo. png",这5个文件的大小分别为1 514、1 696、1 411、1 620、16 168 B,其中,最大的文件为最小的文件的11倍多。由于文件"log_qq. png"的覆盖率最高且大小最小,故选它作为种子文件。

3.2 漏洞挖掘实验

评判一个模糊测试工具能力的最重要的标准就是看它能否检测到漏洞。到目前为止,OSSRWS-Fuzzer 已经发现了 WPS Office for Linux 软件(版本号8.1.0.3724)中的56个故障,其中,et(WPS 表格处理软件)10个,wps(WPS 文字处理软件)19个,wpp(WPS 幻灯片播放软件)27个,部分故障已确认为漏洞。

以 et 为例进行分析,其测试结果如表2所示。

表2 et 测试结果

Tab.2 Test results of et

被测软件名称版本	发现异常的测试参数		发现的异常个数
	r	s	
et 8.1.0.3724	0.004 808 ~ 0.009 615	50	1
	0.000 601 ~ 0.001 202	268	1
	0.001 202 ~ 0.002 404	73,510,521,688	3
	0.000 300 ~ 0.000 601	113,314,433,889	3
	0.000 150 ~ 0.000 300	29,764,586	2

种子文件大小为6 656 B,也就是53 248比特,根据改进的轮盘赌策略,需要划分的区间数为N = [lb 53 248] = 15,相应区间为1/53 248 ~ 2/53 248、2/53 248 ~ 4/53 248、4/53 248 ~ 8/53 248,以此类推。从表2中可以看出,r 的取值在5个区间内发现了异常,且当r的取值为0.001202 ~ 0.002404(即64/53 248 ~ 128/53 248)和0.000 300 ~ 0.000 601(即16/53 248 ~ 32/53 248)时,发现的异常最多,均为3个。这说明,当改变种子文件的16 ~ 32比特和64 ~

128 比特时,测试效果最好。

使用自研的污点分析工具 SwordDTA 对这些异常进行了分析,已经确定 et 的 10 个异常中有 3 个为整数溢出漏洞,其中,一个的污点分析结果如图 2 所示。

```
phoenix@Lenovo-E4430:~/workspace/SwordDTA/tools$ sudo ~/pin/pin -follow_execv -t
 sworddta.so -f 1 -tf /home/phoenix/0day/et/10.xls -- /home/phoenix/wps-office/o
ffice6/et /home/phoenix/0day/et/10.xls
phoenix@Lenovo-E4430:~/workspace/SwordDTA/tools$ cat 0.out
fdset insert: /home/phoenix/0day/et/10.xls, 0xd
Taint introduce: 0x8840b000, 0xa00
Taint introduce: 0x8840b000, 0x1000
Taint introduce: 0x8840b000, 0xa00
Taint introduce: 0x8840b000, 0x1000
Taint introduce: 0x8840b000, 0xa00
Taint introduce: 0x8840b000, 0x1000
Taint introduce: 0x8840b000, 0xa00
Malloc: n= 0xa2dd9104, 0xbf969ae0
phoenix@Lenovo-E4430:~/workspace/SwordDTA/tools$
```

图 2　一个 et 异常的污点分析结果

Fig. 2　Taint analysis result of an et failure

图 2 中,文件“10. xls”是 OSSRWSFuzzer 发现的导致 et 出现异常的输入,“0. out”是保存污点分析结果的文件。当用 et 打开文件“10. xls”时,软件会发生崩溃自动退出。从图 2 可以看到,该异常是由内存分配函数 malloc 的参数 n = 0xa2dd9104 过大引起的,符合整数溢出漏洞的特征。

为进一步说明本文方法的有效性,又使用 zzuf 和另一款著名的模糊测试工具 Peach[22] 对 WPS Office for Linux 8.1.0.3724 进行了测试,测试时间与 OSSRWSFuzzer 相同,均为 10 h,并采用与 OSSRWSFuzzer 相同的种子文件。在测试时,zzuf 保持其默认的模糊率 0.4% 不变,Peach 采用随机变异策略(Peach 有 3 种变异策略:随机、顺序和确定性随机)。实验结果如表 3 所示,可以看到 OSSRWSFuzzer 的测试效果明显好于 zzuf 和 Peach。

表 3　3 种模糊测试工具测试效果对比

Tab. 3　Comparison of the test results of three fuzzing tools

被测软件	发现的故障数		
	OSSRWSFuzzer	zzuf	Peach
wps	19	0	3
wpp	27	0	2
et	10	0	2

4　结　论

从流程优化的思想出发,提出了一种基于改进轮盘赌策略的反馈式模糊测试方法,并设计实现了一个原型系统 OSSRWSFuzzer。该方法基于覆盖率分析来从大量样本文件中筛选最具代表性的文件作为种子文件,基于改进的轮盘赌策略根据测试效果反馈自动实时调整优化种子文件的变异范围。实验表明,该方法具备较强的漏洞检测能力,目前已发现 WPS Office for Linux 软件中的多个漏洞。在进一步的工作中,有 2 点需要改进:一是,区间的划分方法还有待完善;二是,要增加对变异位置的调整。

参考文献:

[1] Sutton M, Greene A, Amini P. Fuzzing: Brute force vulnerability discovery[M]. Upper Saddle River: Addison-Wesley Professional, 2007.

[2] 吴世忠,郭涛,董国伟,等. 软件漏洞分析技术[M]. 北京:科学出版社,2014.

[3] Cadar C, Godefroid P, Khurshid S, et al. Symbolic execution for software testing in practice: Preliminary assessment[C]//Proceedings of the 33rd International Conference on Software Engineering. New York: ACM, 2011: 1066 - 1071.

[4] Cadar C, Sen K. Symbolic execution for software testing: Three decades later[J]. Communications of the ACM, 2013, 56(2): 82 - 90.

[5] Anand S. Techniques to facilitate symbolic execution of real-world programs [D]. Atlanta: Georgia Institute of Technology, 2012.

[6] Newsome J, Song D. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software[C/OL]//Proceedings of the 12th Annual Network and Distributed System Security Symposium. San Diego: ISOC, 2005. http://repository.cmu.edu/cgi/viewcontent.cgi?article=1042&context=ece.

[7]Bosman E,Slowinska A,Bos H. Minemu:The world's fastest taint tracker[C]//Recent Advances in Intrusion Detection. Berlin:Springer,2011:1-20.

[8]Kemerlis V P,Portokalidis G,Jee K,et al. libdft:Practical dynamic data flow tracking for commodity systems[C]//ACM SIGPLAN Notices. New York:ACM,2012,47(7):121-132.

[9]Bekrar S,Bekrar C,Groz R,et al. A taint based approach for smart fuzzing[C]//Proceedings of the 2012 IEEE Fifth International Conference on Software Testing,Verification and Validation. Montreal:IEEE,2012:818-825.

[10]Pak B S. Hybrid Fuzz testing:Discovering software bugs via fuzzing and symbolic execution[D]. Pittsburgh:Carnegie Mellon University,2012.

[11]Wang T,Wei T,Gu G,et al. Checksum-aware fuzzing combined with dynamic taint analysis and symbolic execution[J]. ACM Transactions on Information and System Security,2011,14(2):15.

[12]Caca Labs. ZZUF:multi-purpose fuzzer[EB/OL](2015-06-09)[2015-05-20]. http://caca.zoy.org/wiki/zzuf.

[13]Liang Yuhong,Zhang Xin. An improved method for roulette wheel selection of genetic algorithm[J]. Information Technology,2009,33(12):127-129.[梁宇宏,张欣. 对遗传算法的轮盘赌选择方式的改进[J]. 信息技术,2009,33(12):127-129.]

[14]Xia Guimei,Zeng Jianchao. A stochastic particle swarm optimization algorithm based on the genetic algorithm of roulette wheel selection[J]. Computer Engineering & Science,2007,29(6):51-54.[夏桂梅,曾建潮. 一种基于轮盘赌选择遗传算法的随机微粒群算法[J]. 计算机工程与科学,2007,29(6):51-54.]

[15]Li Chen,Ning Hongyun. Improved selection operator of genetic algorithm[J]. Jounal of Tianjin University of Technology. 2008,24(6):1-4.[李晨,宁红云. 改进的遗传算法选择算子[J]. 天津理工大学学报,2008,24(6):1-4.]

[16]Yang Jiuhong,Wang Xiaozeng. Improved hybrid particle swarm optimization algorithm[J]. Microelectronics & Computer,2012,29(5):170-173.[杨久红,王小增. 改进的混合粒子群算法[J]. 微电子学与计算机,2012,29(5):170-173.]

[17]Wei Bo,Yu Fei,Xu Xing,Xie Chengwang. Interactive evolutionary algorithm based on improved roulette wheel selection strategy[J]. Computer & Digital Engeering,2014,42(10):1763-1767.[魏波,喻飞,徐星,等. 基于改进轮盘赌策略的交互式演化算法磁[J]. 计算机与数字工程,2014,42(10):1763-1767.]

[18]Intel. Pin-A dynamic binary instrumentation tool[EB/OL](2012-06-13)[2015-05-20]. https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-ool.

[19]Tensons Corporation. Website ripper copier[EB/OL][2015-05-20]. http://www.tensons.com/products/websiterippercopier/.

[20]Armour-Brown C,Borntraeger C,Fitzhardinge J,et al. Valgrind[EB/OL](2015-09-22)[2015-05-20]. http://valgrind.org/.

[21]The Runtime Introspection and Optimization Group. DynamoRIO[EB/OL](2002-06-01)[2015-05-20]. http://www.dynamorio.org/.

[22]Deja vu Security. Peach fuzzer[EB/OL](2014-02-23)[2015-05-20]. http://community.peachfuzzer.com/.

(编辑 赵 婧)