一种基于遗传算法的 Fuzzing 测试用例生成新方法

刘 渊,杨永辉,张春瑞,王 伟
(中国工程物理研究院计算机应用研究所,四川绵阳 621900)

摘 要: 本文根据传统漏洞挖掘 Fuzzing 技术的特点,针对其存在的不能求解非线性解和只能有单个的输入的问题,提出了一种基于遗传算法的漏洞挖掘测试用例生成的新方法. 该方法能够利用遗传算法的优势,同时可以应对多输入测试用例问题和非线性求解问题. 从自测程序的结果看出,相比于传统随机生成 Fuzzing 测试用例的方法,本方案在效率和覆盖率方面具有明显的提高.

关键词: 遗传算法; 非线性求解; 多维 Fuzzing 技术
中图分类号: TN311.5 **文献标识码:** A **文章编号:** 0372-2112 (2017)03-0552-05
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.03.007

A Novel Method for Fuzzing Test Cases Generating Based on Genetic Algorithm

LIU Yuan, YANG Yong-hui, ZHANG Chun-rui, WANG Wei
(*Institute of Computer Application, China Academy of Engineering Physics, Mianyang, Sichuan* 621900, *China*)

Abstract: Considering the features of the traditional Fuzzing technology, a method is proposed for Fuzzing test case generating in vulnerability exploiting, which is aimed at nonlinear solution and single input problem. This method takes advantage of the genetic algorithm and deals with those two problems mentioned above. The experiment results show that, the proposed solution has an obvious improvement compared with the early method which generates the test cases randomly.

Key words: genetic algorithm; nonlinearity solution; multidimensional fuzzing technology

1 引言

在目前各类软件的安全测试中,Fuzzing 技术以其高度的自动化特性被广泛使用. 但 Fuzzing 技术在测试的过程中具有很大的盲目性,当遇到巨大的测试用例空间时测试效率受到很大影响. 特别是在求解非线性解和只能有单个输入这两方面具有很大的不足. 本文将遗传算法的思想应用到 Fuzzing 测试用例生成的过程之中,参考历史进化信息,使用适应度函数来影响进化方向. 在不影响 Fuzzing 测试高度自动化的前提下指导测试用例的生成规则,从而提高了 Fuzzing 测试的效率. 最后通过设计实验证明了本方法(适应度函数构造方法)在提高测试效率方面的有效性.

传统的漏洞挖掘主要使用 Fuzzing 测试技术,其基本框架如图 1 所示. 目前常用的 Fuzzing 测试软件 Spike 能够克服静态挖掘的测试效率不高,漏报率高的问题,从而具有较好的测试效果[1~3]. 但是,由于传统 Fuzzing 技术[4~6]的测试用例是随机生成的,而在随机生成的测试用例中往往存在大量冗余无用的数据,占用了大量测试时间,造成系统资源的浪费,因此需要对以往 Fuzzing 测试用例的生成方法进行改进,提高生成有用测试用例的效率.

目前相关研究针对的目标程序多只有一个输入,即待测程序变量只有一个,并且程序中的大部分判断条件基本上都是简单的加减法,即一般线性判断条件(如 $a*x+b<0$). 但在实际的漏洞挖掘中,往往输入变量并不止一个,并且分支语句、循环语句的判断条件也不一定满足线性,故而本文针对这种情况提出了新的解决办法.

收稿日期:2015-01-04;修回日期:2015-09-07;责任编辑:梅志强
基金项目:中国工程物理研究院科学技术发展基金(No. 2014A0403020);中国工程物理研究院网络安全与可信软件重点实验室基金(No. J-2014-KF-01)

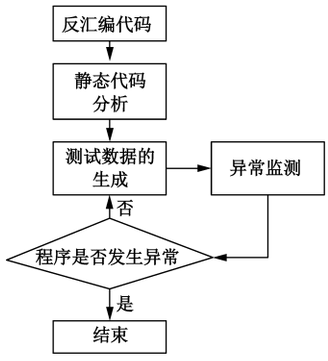

图1　传统Fuzzing测试的基本框架

2　改进的 Fuzzing 测试用例生成方案设计

2.1　总体设计方案

本文总体设计方案的框架如图 2 所示. 在改进的 Fuzzing 测试过程中,通过控制流提取模块获取待测程序的控制流图,从中得到需要检测的目标函数. 使用遗传算法通过交叉变异的方法以尽可能快的速度获取测试用例出发目标使其到达危险函数.

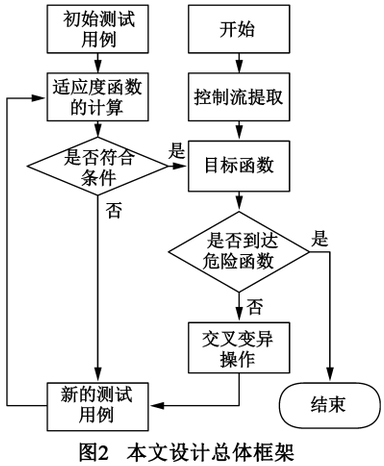

图2　本文设计总体框架

计算机程序一般由一些分支语句(如 if-else),顺序语句,循环语句(如 while/for)组成. 在控制流的提取模块中,将程序的语句按照图 3 所示的三种方式进行提取,并构造程序的控制流图. 通过控制流图,确定需要检测程序的"目标"(程序中任意需要检测的分支、语句,或函数),在此将其称之为所检测程序的危险函数. 当一组输入能够到达这个"目标",即危险函数时,称这个输入是一个"好的"输入,并将此输入作为检测程序漏洞的依据. 在本设计中,通过遗传算法来快速生成这些输入,使得系统能够在最短的时间达到希望进入程序的目标. 控制流分析模块的设计在 2.2 节中详述. 通过多次试验比较得出的适应度函数 Fitness(x)在遗传算法中使用,具体内容见 2.3 节. 遗传算法交叉变异的方法在 2.4 节中给出.

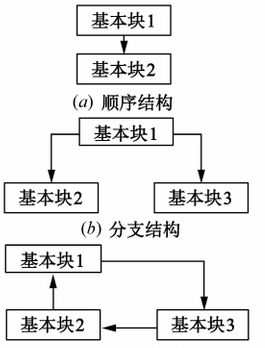

图3　程序的控制流图

2.2　控制流分析模块

软件的功能实现主要是通过一个个程序互相之间紧密协调运用来实现的,所以说分析程序是分析软件的基本[7]. 而程序的运行情况可以通过一些控制流图来表现,故而分析程序的控制流是漏洞挖掘技术的基本,程序的控制流可以用式(1)来表示

$$G=(N,E,D,s,e) \tag{1}$$

其中,G 为程序的控制流,N 为控制流的基本块,D 为控制流的危险函数基本块,E 为基本块之间的路径,s 为控制流的入口节点,e 为控制流的出口节点. 对于本文实验所分析的目标程序,具有 20 个控制流基本块,故 $N=20$,每个基本块可能出现也可能不出现在控制流中,故 $E=2^{20}$,控制流的危险函数基本快 $D=5$.

2.3　适应度函数构造模块

适应度函数是指导测试用例生成的根本,是该测试用例在软件输入好坏的体现. 它反映了各个测试用例的不同,如果适应度函数构造不好,使得不同好坏的测试用例具有相同的适应度函数值,或者使得适应度函数层次划分不够明确,这样就会影响到测试用例的结果,使得测试用例生成不明显. 但过度希望适应度函数较高,这样就会使测试数据陷入过早收敛,使得种群多样性过小[7,8]. 所以,根据具体实验设计合适的适应度函数才是关键.

本实验分析的进行测试的程序的控制流的有 20 个基本块,有 5 个危险函数. 其中只有一条路径能把程序的危险函数全部覆盖. 本实验构造的适应度函数见式(2)

$$\mathrm{Fitnes}(x)=\frac{N+D*k}{N_{\max}+D_{\max}*k} \tag{2}$$

由于程序经过危险函数要比程序经过其他基本块要更重要,故而设危险函数 D 的权值比重是程序基本块 N 的 k 倍. $N_{\max}$是程序基本块的最大数目,在本实验待测程序为 15,$D_{\max}$是程序危险函数的最大数目,本实验程序为 5. 如果程序运行完整个基本块,它的适应度函数的适应度值是最高的,为 1. 由于一些程序的

约束条件是相互矛盾的,故而程序不可能执行完整个基本块,实际上我们只关心程序运行时是否经过危险函数,故而当程序经过 5 个危险函数我们就认为它是最好的.

2.4 交叉变异模块

交叉变异是整个遗传算法中数据改变的根本,选择合适的交叉变异方法对实验的效率起着重要的作用. 现在遗传算法的运用往往只是单维度的优化,对于多维度的优化问题研究还有待提高. 在漏洞挖掘过程中,输入数据不可能单维度,故而研究多维度的遗传算法交叉变异方法才符合实际应用的需要.

在进行交叉变异方法时,先确定输入变量的个数 n,这时每一代种群个数就为 $n*2+2$. 因为每一代中必须存储上一代中一个最好的种群,还要存储一个元素完全是变异值的种群,而且还要存储上一代中最好的种群分别和上一代中按随机轮盘方法选出的那个种群和上一代变异的种群交叉出来的 $2n$ 个种群. 这样一代中就必须有 $n*2+2$ 个种群.

轮盘选择策略是先产生一个 0 到 1 的随机数,由于每一代中种群个数为 $n*2+2$,对每一组种群分别分配一个在 0 和 1 之间的取值区间,例如第一组种群的取值区间为 0 到 dist(1),第二代为 dist(1)到 dist(1) + dist(2). 具体算法见式(3)

$$\mathrm{dist}(i) = \frac{\mathrm{Fitness}(i)}{\sum_{1}^{n*2+2}\mathrm{Fitness}(i)} \tag{3}$$

其中,Fitness(i)为一代中第 i 个种群的适应度函数值. 为了防止一代中用轮盘选择法抽到最好的种群,这样在这一代中交叉就没有意义了,故而如果抽到最好种群时,我们把他种群位数加 1.

由于本实验输入数据有 4 个,故而一代中种群个数为 10,具体的交叉变异方法见图 4.

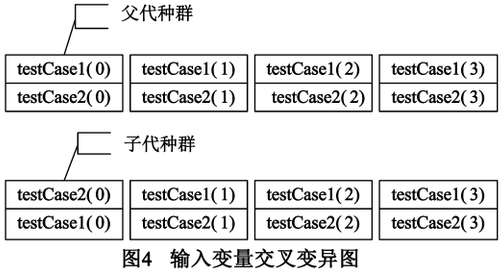

图4 输入变量交叉变异图

3 实验结果分析

本文选取目标程序 InterFlower. exe 进行实验分析,此程序代码 100 行左右. 程序包含 4 个独立不相关的输入:sig(0),sig(1),sig(2),sig(3),以此来模拟多输入程序. 含有 20 个分支语句,其中一些分支的判断条件是线性的,另一些分支判断条件(如$\frac{\mathrm{sig}(2)}{\mathrm{sig}(1)}>5$)是非线性的,以此来模拟针对非线性判断条件的目标程序.

(1)$k=5$ 时,第 1 代中程序到达的基本块和危险函数的执行次数和覆盖率见表 1.

(2)第 100 代中程序到达的基本块和危险函数的执行次数和覆盖率见表 2.

(3)第 278 代中程序到达的基本块和危险函数的执行次数和覆盖率见表 3.

(4)随机测试 5700 代中程序到达的基本块和危险函数的执行次数和覆盖率见表 4.

表 1 $k=5$ 时,第 1 代的数据

程序到达的基本块	危险函数的执行次数	覆盖率
1,2,*D3*,11,12,*D5*	2	30%
2,8,3,*D3*,11	1	25%
2,8,3,*D3*,11	1	25%
2,5	0	10%
2,8,3,*D3*,11,12,*D5*	2	35%
2,5	0	10%
1,3,*D3*,11,12,*D5*	2	30%
2,8,3,*D3*,11,12	1	30%
2,8,3,*D3*,11	1	25%
2,8,3,*D3*,11,12,*D5*	2	30%

表 2 $k=5$ 时,第 100 代的数据

程序到达的基本块	危险函数的执行次数	覆盖率
2,8,3,*D3*,*D4*,12,*D5*	3	35%
2,8,3,*D3*,*D4*,12,*D5*	3	35%
2,8,3,*D3*,*D4*,12,*D5*	3	35%
2,8,3,*D3*,*D4*,12,*D5*	3	35%
2,8,3,*D3*,*D4*,12,*D5*	3	35%
2,8,3,*D3*,*D4*,12,*D5*	3	35%
2,8,3,4,6,11	0	30%
2,8,3,*D3*,*D4*,12,*D5*	3	35%
2,8,3,*D3*,11,12,*D5*	2	35%
2,8,3,*D3*,11,12,*D5*	2	35%

表 3 $k=5$ 时,第 278 代的数据

程序到达的基本块	危险函数的执行次数	覆盖率
1,*D1*,5,*D3*,*D4*,12,*D5*	4	35%
1,*D1*,5,*D3*,*D4*,12,*D5*	4	35%
1,*D1*,5,*D3*,*D4*,12,*D5*	4	35%
1,*D1*,5,*D3*,*D4*,12,*D5*	4	35%
1,*D1*,5,*D3*,*D4*,12,*D5*	4	35%
2,5	0	10%
1,*D1*,4,*D2*,5,*D3*,*D4*,12,*D5*	5	45%
1,*D1*,5,*D3*,*D4*,12,*D5*	4	35%
1,*D1*,5,*D3*,11,12,*D5*	3	35%
1,*D1*,5,*D3*,*D4*,12,*D5*	4	35%

表 4　随机测试第 5700 代的数据

程序到达的基本块	危险函数的执行次数	覆盖率
2,8,3,*D*3,11	1	25%
2,8,3,*D*3,11,12,*D*5	2	35%
2,8,3,*D*3,11	1	25%
2,8,3,*D*3,11,12,*D*5	2	35%
2,8,3,*D*3,11	1	25%
2,8,3,*D*3,11,12,*D*5	2	35%
2,8,3,*D*3,11,12	1	30%
1,*D*1,4,*D*2,5,*D*3,*D*4,12,*D*5	5	45%
2,8,3,*D*3,11,12,*D*5	2	35%
2,8,3,*D*3,11	1	25%

当然,这只是经过一次测试的结果,但由以上表格数据可以看出,用遗传算法生成测试用例每一代中在程序到达的基本块,危险函数的执行次数,覆盖率等方面都要比上一代要好,而且用遗传算法生成测试用例经过 278 代后就可以全部覆盖危险函数,而随机生成测试用例在 5700 代后才全部覆盖危险函数.

在图 5、图 6 中,我们给出遗传算法生成测试用例和随机生成测试用例经过每隔一定代数后的平均危险函数的执行次数和基本块的平均覆盖率比较图.

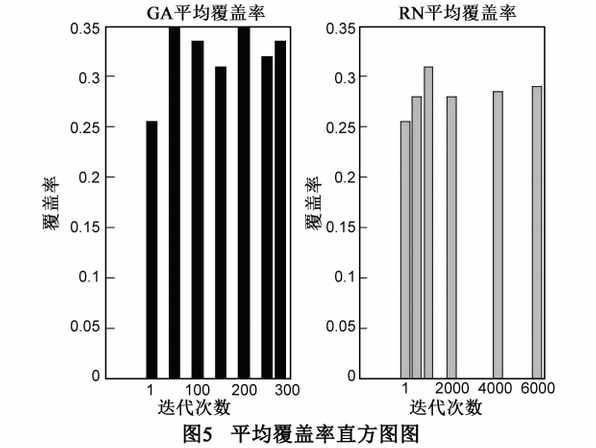

图5　平均覆盖率直方图图

从图中数据可以看出,遗传算法比随机生成操作对程序控制流的覆盖率要好,但是并不是十分明显. 遗传算法相对与随机生成的方法来说,当进行遗传算法操作时,危险函数的执行每一代基本是上升的,而随机生成测试用例的危险函数的执行次数是随机有升有降的. 所以说,我们所用的算法是可行的.

从表中结果可以看出,在危险函数被执行一次的情况下,用遗传算法生成测试用例比随机生成测试用例大约提高 15 倍. 这也说明说明了用遗传算法生成测试用例是可行的.

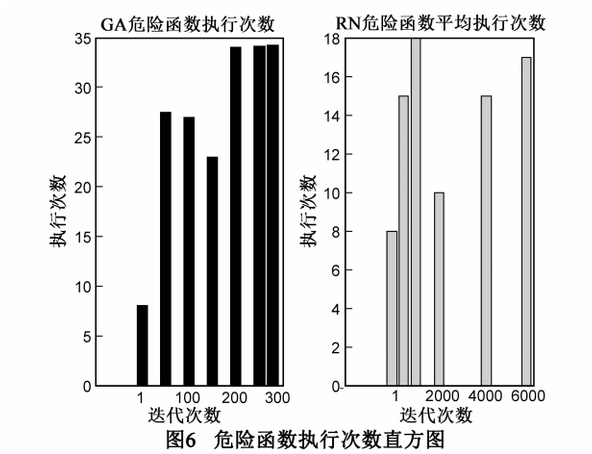

图6　危险函数执行次数直方图

表 5　随机算法和遗传算法搜索效率的比较

随机生成	遗传算法生成
15643	6318
53256	12422
74354	9804
47124	404
17392	2128
21481	4764
119743	10044
16206	7486
16507	833
60146	4213

4　结论

从上述两个程序中用遗传算法进行测试和随机测试的进行对比我们可以看出,用我们遗传算法生成测试用例的方法是有效的. 不论是非线性的,还是线性的,遗传算法的测试结果都要比随机生成的效率高很多. 相对于其他方法来说要好.

参考文献

[1] Chen J M,Shu H,Xiong X B. Fuzzing test approach based on symbolic execution[J]. Computer Engineering,2009,35(21):33 - 35.

[2] 万勇兵,徐中伟,梅萌. 一种符号化执行的实时系统一致性测试生成方法[J]. 电子学报,2013,41(11):2276 - 2284.
WAN Yong-bing,XU Zhong-wei,MEI Meng. A symbolic execution method for conformance test generation of real-time system[J]. Acta Electronica Sinica,2013,41(11):2276 - 2284. (in Chinese)

[3] Biyani,Aabha,Sharma,Gantavya,Aghav,Jagannath,et al.

Extension of SPIKE for encrypted protocol fuzzing[A]. Multimedia Information Networking and Security (MINES), 2011 Third International Conference on IEEE [C]. Shanghai1: IEEE, 2011. 343 – 347.

[4] Bhansali S, Chen W K, Jong S D, et al. Framework for instruction-level tracing and analysis of program executions [A]. Proceedings of International Conference on Virtual Execution Environments[C]. New York: ACM, 2006. 154 – 163.

[5] Song D, Brumley D, Yin H, et al. BitBlaze: A new approach to computer security via binary analysis[A]. Proceedings of the 4th International Conference on Information Systems Security2008 [C]. Berlin, Heidelberg: Springer-Verlag, 2008. 1 – 25.

[6] Wang T, Wei T, Gu G, et al. TaintScope: a checksum-aware directed fuzzing tool for automatic software vulnerability detection[A]. Security and Privacy (SP), 2010 IEEE Symposium on IEEE[C]. Washington, DC, USA: IEEE Computer Society, 2010. 497 – 512.

[7] Cui B, Liang X, Wang J. The study on integer overflow vulnerability detection in binary executables based upon genetic algorithm[J]. Advances in Intelligent & Soft Computing, 2011, 122: 259 – 266.

[8] Memon A M, Pollack M E, Soffa M L. Hierarchical GUI test case generation using automated planning[J]. IEEE Transactions on Software Engineering, 2001, 27(2): 144 – 155.

作者简介

刘 渊 男,1974 年 2 月出生于四川自贡,高级工程师,研究方向为网络与信息安全、软件安全分析与漏洞检测.
E-mail: lyisme@caep.cn

杨永辉(通信作者) 男,1973 年 4 月出生于江西丰城,研究员,博士生导师,研究方向为信息对抗、软件安全测评.
E-mail: younphy@163.com

张春瑞 男,1980 年 2 月出生于山西祁县,高级工程师,研究方向为网络与信息安全、大数据分析.
E-mail: zhangcr@caep.cn

王 伟 男,1986 年 3 月出生于山东滕州,工程师,研究方向为软件安全分析与漏洞检测.
E-mail: wwmuou@caep.cn