模糊测试中测试用例生成方法①

李 彤, 黄 轩, 黄 睿

(装甲兵工程学院 信息工程系, 北京 100072)

摘 要: 代码覆盖率一直是影响模糊测(Fuzzing)测试效率的重要因素, 而模糊测试用例则很大程度上影响代码覆盖率, 所以如何构造高效的测试用例就显得非常重要. 将遗传算法应用到测试用例的生成上, 可以实现降低测试用例的冗余度, 还能提高代码的覆盖率. 从而使被测程序在尽量短的时间内得到充分的测试, 提高模糊测试的效率和效果.

关键词: 模糊测试; 测试用例; 代码覆盖; 遗传算法

Test Case Generation Method in Fuzzing

LI Tong, HUANG Xuan, HUANG Rui

(Department of Information Engineering, Academy of Armored Force Engineering, Beijing 100072, China)

Abstract: Code coverage has been an important factor affecting the efficiency of Fuzzing, but it is largely affected by Fuzzing test cases, so it is very important to construct efficient tests. Applying genetic algorithm into the generation of test cases, it can not only reduce the redundancy of test cases, but also improve code coverage. So that we can fully test the target in less time, and improve the efficiency and effectiveness of Fuzzing test.

Key words: Fuzzing; test case; code coverage; genetic algorithm

目前, 模糊测试已经被广泛的应用于各种软件的安全测试中, 这得益于其高度自动化的特性. 但模糊测试本身具有盲目性, 传统的模糊测试是在输入空间内随意取值, 这样产生的测试用例很大一部分在还没有进入到软件内部就被拒绝了, 从而导致了整个测试效率低. 遗传算法是利用进化历史中获得的信息指导下一步的搜索或计算, 其中适应度函数确定了整个算法的方向. 文中将遗传算法引入到模糊测试的测试例生成过程中, 通过利用测试过程中产生的历史信息, 由适应度函数来控制进化方向并确定下一代的测试用例. 在保证其高度自动化的基础上使得模糊测试的测试具有了方向性, 从而提高测试的效率.

1 模糊测试技术概述

模糊测试最早在 1989 年由 Wisconsin-Madison 大学的 Barton Miller 教授提出[1], 用于测试 UNIX 系统的健壮性. 模糊测试是一种自动化或半自动化的软件测试技术, 通过给程序提供恶意的输入迫使程序异常, 然后分析异常发生的位置和原因, 从而对程序内部的缺陷做出判断. 模糊测试的工作流程如图 1 所示.

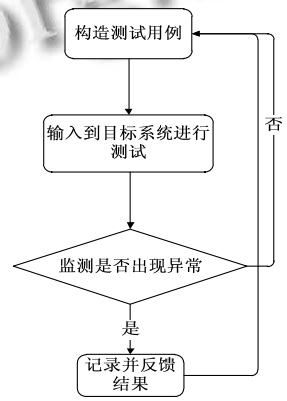

图 1 模糊测试的流程

完整的模糊测试过程大致要经过识别目标及输入, 生成模糊测试数据, 执行模糊测试数据, 监视异常, 确

① 收稿时间:2014-07-17;收到修改稿时间:2014-09-04

定可利用性五个阶段. 其中最为关键的是生成模糊测试数据阶段, 传统的模糊测试是在输入数据空间内任意取值, 效率不仅低下而且效果不是特别理想. 基于遗传算法的模糊测试可以看作是一种非常有前景的自动化测试技术, 它可以成功地为多种测试目标生成高质量的测试用例.

2　遗传算法简介

遗传算法(Genetic Algorithms, GA)由美国Michigan大学的John Holland与其同事于20世纪60年代末提出并创立[2]. 它来源于达尔文的进化论、魏茨曼的物种选择学说和孟德尔的群体遗传学说. 遗传算法本质上是自适应的机器学习方法, 是由进化论和遗传学机理而产生的一种优化方法, 其核心思想是将问题的求解表示为染色体群的一代代不断进化, 包括选择、交叉和变异等操作, 最终收敛于最能适应环境的个体, 从而求得问题的最优解或满意解[3]. 其优点是简单、通用、鲁棒性强和适于并行处理, 近年来已被逐渐应用到软件测试的测试数据生成中来.

3　基于遗传算法的测试用例生成

针对现有模糊测试中, 存在着测试用例生成效率低, 代码覆盖率不高, 难以定位异常位置等不足, 本文采用一种演化模糊测试方法, 其思想是将遗传算法应用到测试用例的生成中, 并通过调试器监测目标应用, 记录目标应用对于测试用例的反应, 并将反应信息反馈给遗传算法, 用于指导更高质量的测试用例生成. 其原理图如图2所示.

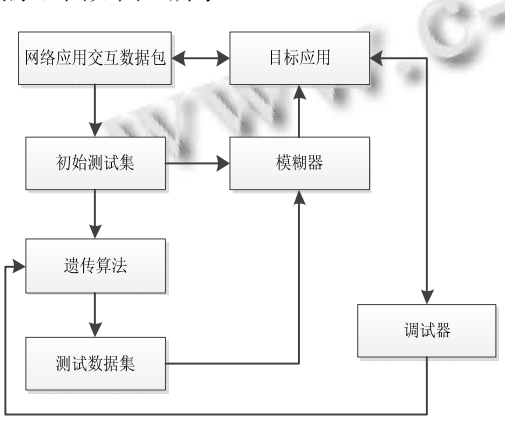

图2　演化模糊测试原理图

而在上面的原理图中, 关键的就是遗传算法的应用问题. 将遗传算法应用到测试用例生成中, 需要解决参数的编码, 适应度函数的设计, 遗传操作等问题. 在整个遗传算法的迭代过程中, 我们首先截取一些网络数据包作为初始集, 然后通过选择、交叉、变异等遗传操作来对测试用例进行生成, 在这个过程中, 变异和交叉操作都是自适应的, 最终到达我们的终止条件遗传算法就结束了. 如图3所示.

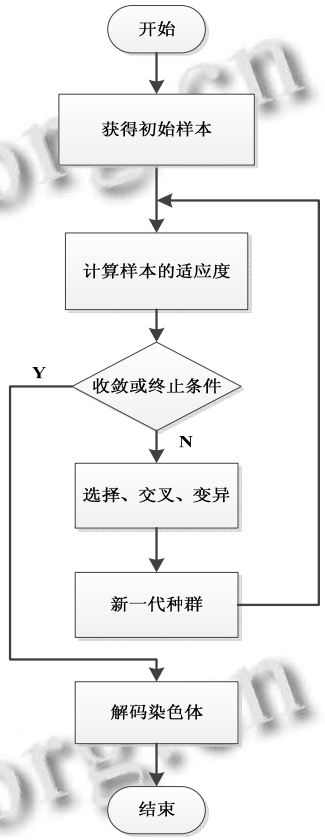

图3　遗传算法的基本流程

3.1 编码方式

编码是一个非常关键的问题, 遗传算法整个进化过程是建立在编码机制基础上的. 针对不同的问题, 我们应当采用不同的编码. 编码的一个重要原则是, 要保证在进化的过程中, 个体的基因编码格式不能被破坏, 否则将无法识别个体. 在基于遗传算法的测试用例生成的过程中, 如果个体最基本的遗传单元被破坏, 则该个体将会无法被目标应用识别和接受, 从而不会进一步的深入到程序内部, 也不会触发程序内部的异常, 这样就无法发现程序的漏洞了.

由于本文是对网络应用程序的模糊测试, 在生成测试用例的过程中实际上就是网络数据包的构造. 我

们在构造数据包时要根据不同的协议, 严格按照协议标准来构造, 否则应用程序将会丢弃不符合协议规约的数据包. 以 FTP 协议为例, 在与 FTP 服务器交互中, 我们首先得输入: USER username. 其中“USER”是命令字, “username”是用户名字符串. 命令字是不能变的, 否则服务器会返回未知命令错误, 拒绝用户访问. 因此我们对构造数据包, 主要是对协议规则中的数据字段进行编码. 对于网络数据包, 采用的是控制字段和数据字段分开编码, 控制字段用字符串编码, 数据字段用二进制编码, 这样不仅可以防止不变字段被遗传操作破坏, 还能对数据字段进行充分的编码, 示意图如图 4 所示.

USER	10011001011001101101111O
PASS	01011011010101011100O111

图 4 数据包编码示意图

3.2 适应度函数

适应度函数是整个遗传算法的核心, 它决定算法往哪个方向进行, 直接影响到整个算法的效率. 适应度函数的构造要充分表达遗传算法要解决的问题, 从而引导整个进化过程更快的向最优解空间收敛. Fuzzing 的目的就是尽量多的触发漏洞, 基于经验和常识, 那些拥有高代码覆盖率的测试用例触发漏洞的可能性更大, 也就是说测试用例使软件运行的越彻底, 发现漏洞的几率就越大. 因此, 遗传算法运用于 Fuzzing 中要解决的问题就由发现漏洞转化为从测试例空间中搜寻代码覆盖和多样性最大的测试用例. 在本文中, 我们首先使用调试器软件关联到目标程序, 在相应的入口函数设置断点, 然后将生成的测试集发送给目标程序, 调试器跟踪代码的执行, 通过记录触发的断点数来计算测试用例的代码覆盖率. 因此适应度函数 F 定义如下:

$$F = \frac{C}{C_{sum}} \tag{1}$$

其中 C 表示测试用例触发的目标应用的断点数. C_{sum} 表示目标应用中被设置的断点总数. 根据适应度函数来选择适应度高的测试用例, 然后进行选择, 交叉, 变异操作得到下一代的测试集, 重复上述步骤直到达到终止条件[5].

3.3 选择操作

选择运算又称为繁殖、再生, 或复制运算, 用于模拟生物界去劣存优的自然选择现象. 它从上一代种群中选择出适应性强的某些染色体, 为染色体交叉和变异运算作好准备. 选择的方法有轮盘赌转法、随机遍历抽样、锦标赛选择等. 其中常用的是轮盘赌转法, 它把种群中所有染色体适应度的总和看作一个轮子的圆周, 而每个染色体按其适应度在总和由所占的比例占据轮子的一个扇区. 每次染色体的选择可看作轮子的一次随机转动, 它转到那个扇区停下来, 那个扇区对应的染色体就被选中. 尽管这种选择方法是随机的, 但它与各染色体适应度成比例[6]. 这是因为适应度大的染色体占据轮子扇区面积大, 被选中的概率就高, 适应度小的染色体占的扇区小, 被选中的概率就低. 令 $\sum f_i$ 表示当前群体的适应度的总和, f_i 为群体中第 i 染色体的适应度值, 定义累计函数:

$$D_i = \sum_{i=1}^{k} \left(f_i / \sum f_i \right) \tag{1}$$

选择新个体时, 个体 a_k 对应的选择区间为[D_k, D_{k+1}). 每次选择时就产生一个位于 0 到 $\sum f_i$ 之间的随机数 r, 那么 r 所在的选择区间对应的个体就被选中作为下一代的成员[7]. 这里我们还用到了精英保留策略, 即当前群体中适应度最高的个体不参与交叉运算和变异运算, 而是用它来替换掉本代群体中经过交叉、变异等遗传操作后所产生的适应度最低的个体. 采用这种方法的优点是进化过程中某一代的最优解可以不被交叉和变异操作破坏, 能够加速算法收敛到全局最优解.

3.4 自适应交叉和变异操作

交叉和变异操作的目的是为了保持个体的多样性, 防止种群出现未成熟收敛现象. 本文提出使用自适应交叉算子 P_c 和变异算子 P_m, P_c 和 P_m 能够随适应度自动改变. 当种群中各个体适应度趋于一致或趋于局部最优时, 使 P_c 和 P_m 增加, 而当群体适应度比较分散时, 使 P_c 和 P_m 减少. 同时, 对适应度值高于群体平均适应度值的个体, 对应比较小的 P_c 和 P_m; 而对适应度值高于群体平均适应值的个体, 对应较高的 P_c 和 P_m[8]. 本文中的 P_c 和 P_m 算子按照如下的公式进行自适应调整:

$$P_c = \begin{cases} \sqrt{\left(f_{max} - f_c\left(f_{max} + f_c\right)\right) / \left(f_{max} - f_{avg}\right)}, & f_c \geq f_{avg} \\ 1, & f_c < f_{avg} \end{cases} \tag{2}$$

$$P_m = \begin{cases} \sqrt{\left(\left(f_{max} - f_m\right)\left(f_{max} + f_m\right)\right) / \left(2\left(f_{max} - f_{avg}\right)\right)}, & f_m \geq f_{avg} \\ \frac{1}{2}, & f_m < f_{avg} \end{cases} \tag{3}$$

其中 f_{max} 为群体中最大的适应度值; f_{avg} 为每代群体的平均适应度值; f_c 为交叉操作的染色体适应度值, f_m 为变异操作的染色体适应度值. 根据编码和具体问题的不同, 相应的由多种交叉策略, 如离散重组、中间重组、线性重组、单点交叉、多点交叉和均匀交叉. 这里我们采用单点交叉策略(图 5), 因为我们是构造网络数据包, 如果采用其他的交叉方法, 可能得到被破坏的包, 这样就降低了生成效率, 从而影响整体的测试效率. 单点交叉即选择一个交叉点, 子代在交叉点之前的基因从一个父代基因中得到, 后面的部分从另一个父代基因中得到.

常用的变异操作有: 基本位变异, 均匀变异, 二元变异, 高斯变异. 本文中采用的是基本位变异, 指对个体编码串以变异概率 P_m 随机指定某一位或某几位基因进行变异操作.

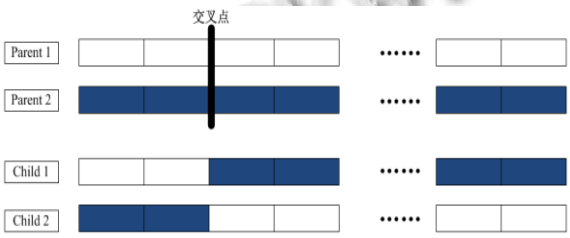

图 5　交叉算子

4　实验分析

本文的测试目标是当前局域网内广泛使用的通信软件—飞秋, 通过对飞秋消息的分析, 发现飞秋消息头部通用格式为: {0}_lbt4_{1}#{2}#{3}# {4}#{5}#{6}:{7}:{8}:{9}:{10}:消息内容. 其中各个变量的含义如图 6 所示:

编号	内容
{0}	版本号，为1，版本号1与IPMsg兼容；
{1}	发送者头像编码；
{2}	发送者等级标志编码号；
{3}	发送者MAC地址；
{4}	自定义头像的图片文件的大小；
{5}	自我形象的图片文件大小；
{6}	消息内容长度；
{7}	消息序号，一般取毫秒数；
{8}	发送者用户名；
{9}	发送者主机名；
{10}	命令字。

图 6　飞秋消息各变量定义

针对以上飞秋的消息头部通用格式, 我们采用遗传算法来构造测试数据, 参数设置为: 种群大小为 30, 染色体长度为 22, 交叉概率为 0.80, 变异概率为 0.02, 进化代数 100. 实验结果如图 4 所示.

进化代数	总交叉操作次数	总变异操作次数	最小适应度	最大适应度	平均适用度	时间
10	120	35	0.099852	0.158502	0.110043	307ms
20	233	73	0.105596	0.210087	0.156945	650ms
30	353	104	0.143654	0.274596	0.193645	1044ms
40	477	131	0.188945	0.293365	0.224887	1394ms
50	596	170	0.213655	0.336587	0.253669	1704ms
60	717	196	0.246698	0.358798	0.276648	2036ms
70	839	235	0.265562	0.371966	0.302165	2297ms
80	955	274	0.295477	0.374511	0.326652	2507ms
90	1076	311	0.352546	0.376984	0.362147	2865ms
100	1192	338	0.372365	0.376994	0.374568	3012ms

图 7　实验结果图

从图中可以看出, 进过 100 代后, 测试用例的代码覆盖率平均值有 37%, 之后我们将这些测试用例发送给飞秋, 并将飞秋附加到调试器软件 Ollydbg, 以监视飞秋在处理测试用例时的异常. 结果飞秋对其中一个测试用例响应时出现了异常(图 7 所示). 从调试器反馈的信息来看, 指令寄存器(EIP)被字符'A'所填充, 其值变为 41414141, 为缓冲溢出漏洞.

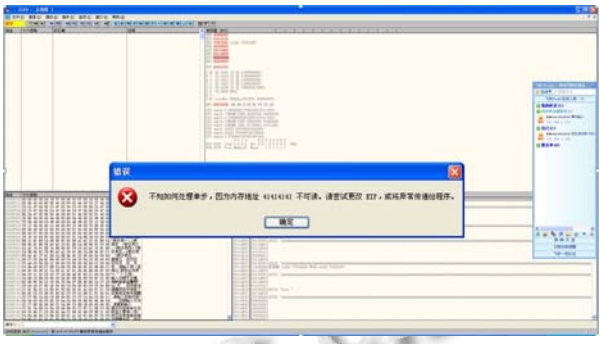

图 8　Ollydbg 监测到飞秋异常

5　结语

为了有效的进行测试用例的自动生成, 本文提出了基于遗传算法的 Fuzzing 测试用例生成的方法, 并对遗传算法的编码、选择算子、交叉算子、变异算子以及适应度函数进行定义和设计, 最后以飞秋的数据报文进行了生成实验, 从实验结果来看, 通过遗传算法生成的测试用例测试效率比较高, 触发了异常, 确定存在缓冲区溢出漏洞. 但是该方法还存在一些问题, 比如无法解决遗传算法早熟收敛的问题, 生成的测试用例无法抵达程序的内部. 下一步, 我将在遗传算法的适应度函数和遗传算子等方面进行改进.

参考文献

1 Miller BP, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities. Commun. ACM, 1990, 33(12): 32–44.

2 葛继科,邱玉辉,吴春明,等.遗传算法研究综述.计算机应用研究,2008(10): 2911–2916.

3 Yang C, Chunhua H, Luming L. An approach to generate software test data for a specific path automatically with genetic algorithm. Chengdu: 2009.

4 苗亮亮.基于遗传算法软件测试用例自动生成分析与研究[学位论文].兰州:兰州交通大学,2013.

5 王丹,高丰,朱鲁华.基于遗传算法的 Fuzzing 测试用例生成模型.微电子学与计算机,2011,(5):130–134.

6 林惠娟.基于遗传算法的测试用例自动生成技术研究[学位论文].成都:四川大学,2006.

7 王欢,王曙燕,孙家泽.基于遗传算法的 DM-GA 组合测试数据生成方法.计算机应用与软件,2012,(8):62–65.

8 吴云,胡小娟,邱宁佳,等.基于遗传算法的测试用例生成技术研究.长春理工大学学报(自然科学版),2010, (3): 137–139.