Research On The Generation Method Of Test Cases In Fuzzing

Hongliu Cai[1, a], Xi Yu[2,b] and Liuyuqin Deng[3,c]

[1,2,3]Department of Information Engineering, Academy of Armored Force Engineering, Beijing, China 100072

[a]caihongliubj@163.com, [b]yuxijxx@163.com, [c]email

Keywords: Fuzzing, test case, code coverage, genetic algorithm

Abstract. Code coverage has always been an important factor affecting the efficiency of Fuzzing, and it is largely affected by Fuzzing test cases, so it is very important to construct efficient test cases. 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 effect of Fuzzing test.

Introduction

Nowadays, Fuzzing test has been widely applied to various kinds of software security testing, thanks to its high automation feature. However the Fuzzing test itself has blindness. Because the traditional Fuzzing test extracts values casually within input space, most of test cases cannot access the interior of software, which leads to a low testing efficiency. Genetic algorithm utilizes the information from evolutionary history to conduct search and calculation in next step. Therein the fitness function determines the direction of the whole algorithm. This paper introduces the genetic algorithm into the process of generating Fuzzing test cases, using information from evolutionary history to conduct search and calculation, controlling evolutionary direction through fitness function, and determining the test cases of next generation. On the basis of guaranteeing its high automation feature, the Fuzzing test could have directivity, which enhances efficiency of the test.

The summary of Fuzzing test technology

The concept of Fuzzing test is firstly proposed in 1989 by professor Barton Miller from Wisconsin-Madison College[1], used for robustness test of the UNIX system. It's a kind of automatic or semi-automatic software testing technology. It compels program exception by providing malicious input, and then analyses the position and reason of the exception, thus making a judgment on the fault within program. The work flow of Fuzzing test is shown in the Fig. 1.

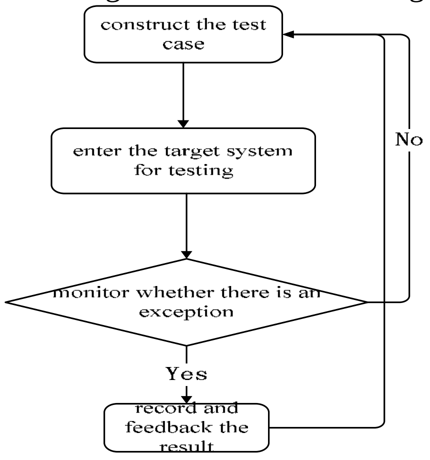
construct the test case
enter the target system for testing
monitor whether there is an exception
No
Yes
record and feedback the result

Fig. 1. the work flow of Fuzzing test

The whole process of Fuzzing test have five phases: identify the target and input, generate Fuzzing test data, execute Fuzzing test data, monitor exception, determine the utilization. The most important phase is generating Fuzzing test data. The traditional method is extracting values casually within input data space, which leads to low efficiency and unsatisfying effect. The Fuzzing test based on genetic algorithms could be regarded as a very promising automatic testing technology, which can generate testing samples with high quality for various testing targets successfully.

The brief introduction of Genetic Algorithms

The Genetic Algorithms(GA) was proposed and founded by John Holland and his colleagues from America Michigan college in the late 1960s[2]. It originates from the Darwinism, the Species Selection Theory of Weizmann and the Population Genetics Theory of Mendel. The essence of Genetic Algorithms is an adaptive machine learning method, based on Darwinism and genetics. The core idea is to express problem solving as endlessly evolution of generations of chromosome complex, including the operations of selection, crossover, variation and so on. Finally, it converge on a unity with highest adaptability to environment in order to get the optimal solution or satisfactory solution for the problem[3]. It has advantages of simpleness, versatility, strong robustness and being appropriate for parallel processing. In recent years, it has been gradually applied to the test data generation in software testing.

The generation of test cases based on Genetic Algorithms

Focusing on the deficiencies of low test cases generation efficiency, low code coverage ratio and being difficult in positioning exception in existing Fuzzing test, this paper adopts an evolutionary method of Fuzzing test. The idea is to apply the genetic algorithm to the test data generation, and monitor the target application through the debugger, then record reaction of the target application towards test cases and give response information feedback to the genetic algorithm for guiding test cases generation of higher quality. The principle is shown in the Fig. 2.

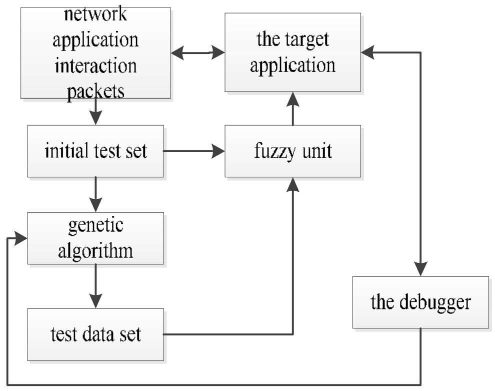

Fig. 2. the principle for evolution of Fuzzing test

In the schematic diagram above, the key is the application of genetic algorithm. Applying genetic algorithm into test cases generation should solve the problems of parameter coding, fitness function design and genetic operation, etc. In the whole iteration process of genetic algorithm, we capture some network data package as initial set firstly, then generate test cases through the genetic operations of selection, crossover, variation and so on. In this process, selection and variation operation are adaptive. When it finally reaches our termination conditions, the genetic algorithm is over. As shown in the Fig. 3.

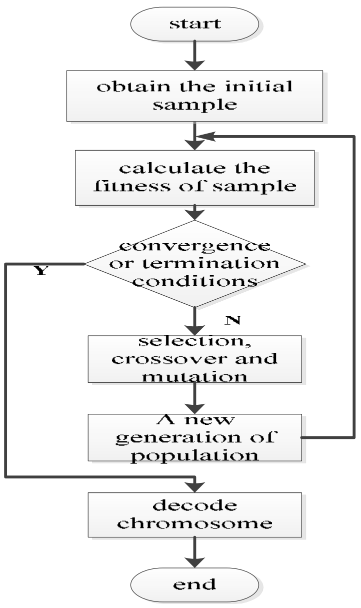
start
obtain the initial sample
calculate the fitness of sample
convergence or termination conditions
Y
N
selection, crossover and mutation
A new generation of population
decode chromosome
end

Fig. 3. the basic procedure of genetic algorithms

coding scheme

Coding is a crucial problem. The whole evolutionary process of genetic algorithm is based on the coding format. According to different problems, we ought to adopt different coding schemes. An important principle of coding is that we should insure the gene coding format of individuals won't be destroyed in evolutionary process, or we can't recognize the individuals. In the process of generating test cases based on genetic algorithm, if the fundamental genetic unit of individual has been destroyed, it won't be recognized and accepted. In this case, the target couldn't enter into the internal program to trigger exception so that we cannot find the program bug.

Since this paper tends to describe the Fuzzing tests of network applicant program, the test cases generation is actually the construction of network data packets. According to different protocols, we should strictly observe the protocol standards to create packets or the applicant will dischard the data packets that don’t conform to the protocol standards. Take the protocol of FTP as an example, when interacting with FTP servers, firstly we input ‘USER username’, the 'USER' is a command word, the 'username' is user-string. The command word is immutable, or servers will return unknown command error to refuse access. So the main factor in constructing packets is coding for data field of protocol standard. For the network data packet, we code control field and data field separately. Specifically, control field uses string coding and data field uses binary coding. In this way, it can not only avoid genetic operation destroying the immutable word, but also code data field sufficiently. As shown in Fig. 4.

USER	10011001011001101101110
PASS	01011011010101011100011 1

Fig. 4. packet encoding schematic

Fitness function

Fitness function is the core of whole genetic algorithms. It decides the direction and efficiency of the algorithms. The construction of fitness function should express the problems that the genetic algorithms need to solve in order to lead the whole evolution process of to the most optimal solution space faster. The aim of Fuzzing is trigger bugs as many as possible. According to experience and knowledge, those test cases with high code fraction coverage are more likely to trigger bugs, that is, if test cases make the software running more thoroughly, there will be more chances to find bugs. Therefore the problem of genetic algorithms on Fuzzing has changed from discovering bugs to finding test case with the highest code fraction coverage and diversities in testing space. In this paper, we firstly use debugger software to relate to target program and set the break point at corresponding entrance function, then send the generated testing sets to the target program, counting code fraction coverage of test cases by recording the number of break points. So the fitness function *F* is defined as follow:

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

C indicates the number of breakpoints in object application triggered by test cases, C_{sum} indicates the total number of breakpoints set in object application. The test cases with high fitness are chosen according to the fitness function, and then are made the operations of selection, crossover and variation to get the next-generation testing set. Repeat the steps mentioned above until achieve the termination condition[5].

Selection operation

The selection operation is also called reproduction, regeneration or copy operation to imitate the natural selection phenomenon that only the superior could survive. It selects some well-adapted chromosomes from the previous generation, preparing for the operation of chromosome crossover and variation. The methods of selection are roulette turning, random sampling, tournament selection and so on. The roulette turning method is the most common one. It regards the total fitness of all chromosomes in the population as the circumference of a wheel, and each chromosome represents a section in the wheel according to its fitness proportion in the sum. Each time the chromosome selection is like the wheel rolling once randomly. When the wheel stops in one section, the corresponding chromosome in the section is selected. Though the method is random, it is proportionate to the chromosome fitness[6]. Because the high-fitness chromosomes occupy larger section in the wheel and are more likely to be selected while the low-fitness chromosomes are on the contrary .

Σf_i indicates the total fitness of current population, f_i is the fitness of the No.*i* chromosome. Define the cumulative function as follow:

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

When selecting the new individual, the corresponding selective area of the individual ak is [Dk，Dk+1). Each time selection will produce a random number *r* from 0 to Σf_i , then the individuals in the corresponding selective area of *r* are selected as members of the next generation[7]. Here we also use the elite-preserving strategy, that is, the individual with highest fitness in current population will not participate in the operation of crossover and variation, but is used to replace the individual with lowest fitness. The advantage of this method is the optimum solution of a certain generation in evolution process could not be destroyed by the crossover and variation operation, which could accelerate the convergence to the overall optimum solution.

Adaptive Crossover and Variation Operation

The purpose of crossover and variation operation is to keep the diversity of individuals, and to prevent the immature convergence in a population. In this paper, we propose to use adaptive crossover operator ***P****c* and variation operator ***P****m*, ***P****c* and ***P****m* can automatically change with fitness. When the individuals fitness in population tends to be the same or local optimum, ***P****c* and ***P****m* increase; while group fitness disperses, ***P****c* and ***P****m* decrease. Meanwhile the individual which fitness value is lower than average in population should corresponding to the smaller ***P****c* and ***P****m* , the individual which fitness value is higher than average in population corresponding to the larger ***P****c* and ***P****m*. In this paper, ***P****c* and ***P****m* operator make adaptive adjustments according to the formula as follow:

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

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

f_{max} is the largest fitness value in population, f_{avg} is the average fitness value in each generation, f_c is the fitness value of chromosome in crossover operation, and f_m is the fitness value of chromosome in variation operation. According to different encoding and concrete problems, there are various corresponding crossover strategies, such as discrete reorganization, restructuring, linear restructuring, single-point crossover, multi-points crossover and uniform crossover. Here we adopt single-point crossover strategy(Fig. 5) , because we construct network data packets, if we take other methods, we may just get a destroyed packet, which reduces generation efficiency and impacts whole testing efficiency. Single-point crossover is to choose a intersection that the offspring gene before intersection obtains from one parent and the gene after intersection obtains from another parent.

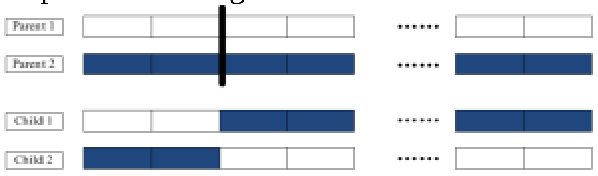

Fig. 5. crossover operator

The common variation operations are basic variation, uniform variation, dual variation, Gaussian variation. In this paper, we adopt basic variation, which randomly assigns one or several bit gene by variation probability P_m for individual coding string to carry out variation operation.

Experiment Analysis

In this paper, the testing target is a communication software called Feiqiu which is widely used in current local area network(LAN). Analyzing the Feiqiu messages, we found the general format of Feiqiu message head is:{0}_lbt4_{1}#{2}#{3}#{4}#{5}#{6}:{7}:{8}:{9}:{10}:content. The meaning of these variables are shown in Table 1:

Number	Content
{0}	Version number, 1, version number 1 is compatible with IPMsg
{1}	Head image coding of the sender
{2}	Registration marks of the sender coding number
{3}	The MAC address of the sender
{4}	The size of the head image files
{5}	The size of the self image files
{6}	The length of the message
{7}	Message number, the miximum unit usually is millisecond
{8}	The user name of the sender
{9}	The host name of the sender
{10}	Command word

Table 1. Variable definition of Feiqiu message

According to the general format of Feiqiu message head, we adopt genetic algorithm to construct the test data, and the parameter of population is set at 30, the length of chromosome is 22, crossover probability is 0.80, variation probability is 0.02, evolution algebra is 100. The experiment results are shown in Table 2.

Evolution algebra	The total number of crossover operation	The total number of variation operation	The minimum fitness	The maximu m Fitness	The average Fitness	times
10	120	35	0.099852	0.158502	0.110043	307ms
20	233	73	0.105596	0.210087	0.110043	650ms
30	353	104	0.143654	0.274596	0.156945	1044ms
40	477	131	0.188945	0.293365	0.193645	1394ms
50	596	170	0.213655	0.336587	0.224887	1704ms
60	716	196	0.246698	0.358798	0.253669	2036ms
70	839	235	0.265562	0.371966	0.276648	2297ms
80	955	274	0.295477	0.374511	0.326652	2597ms
90	1076	311	0.352546	0.376984	0.362147	2865ms
100	1192	338	0.372365	0.376994	0.374568	3012ms

Table 2. The experiment results

As the figure shown, the average value of the code coverage of test cases is 37% after 100 generation. And then we send these test cases to Feiqiu which is added to the debugger software Ollydbg to monitor the exception in processing the test cases. We found the exception when Feiqiu responses to one of the test cases(Fig. 6 as shown). As the information feedback from debugger shown, execute interface program(EIP) was filled with character ‘A’, and the value became 41414141, which is a buffer-overflow bug.

Conclusion

In order to automatically generate test cases efficiently, this paper proposes the method to generate Fuzzing test cases based on genetic algorithm. And then we define and design the coding genetic algorithm, selection operator, crossover operator, variation operator and fitness function. Finally we make an experiment with data packets of Feiqiu, and the results shows that the genetic algorithm has higher testing efficiency in generating test cases. Also it triggers the exception and makes sure there is a buffer-overflow bug. However this method has some problems, for example, the problem of immature convergence of genetic algorithm can’t be solved, the generated test cases cannot reach the internal program. Next we will improve the fitness function of genetic algorithm and genetic operators, etc.

References

[1] Miller B P, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities [J]. Commun. ACM, 1990, 33 (12) : 32-44.

[2] Jike Ge, Yuhui Qiu, Chunming Wu, etc. Research review on the genetic algorithm [J]. Computer application research. 2008 (10) : 2911-2916.

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

[4] Lingling Miao. Analysis and research on software automatically generate test cases based on the genetic algorithm [D]. Lanzhou jiaotong university, 2013.

[5] Dan Wang, Feng Gao, Luhua Zhu. The generation model of test cases in Fuzzing based on genetic algorithm [J]. Journal of microelectronics and computer. 2011 (5) : 130-134.

[6] Huijuan Lin. Research on the generation technique of automatic test cases based on genetic algorithm [D]. Sichuan university, 2006.

[7] Huan Wang, Shuyan Wang, Jiaze Sun. The generation method of DM-GA combination test data based on genetic algorithm [J]. Journal of computer applications and software. 2012 (8) : 62-65.

[8] Yun Wu, Xiaojuan Hu, Ningjia Qiu, etc. Research on the generation technology of test cases based on the genetic algorithm [J]. Journal of changchun university of science and technology (natural science edition), 2010 (03) : 137-139.