Software Security Test Data Generation Based on Genetic Algorithms

Qiong Li, and Jinhua Li
College of Information Engineering, Qingdao University, Qingdao, China
Liqiong1230@163.com, lijh@qdu.edu.cn

I. INTRODUCTION

Software security is about making software behave in the presence of a malicious attack. The intention of the software is to security ascertain whether the characters of security is in line with the prospective requirements when the software was design [1]. Generation efficient test cases are the essential passport for simplifying the test work and improving the test efficiency [2]. The test work is inefficient because of the great number of the initial test cases, so some optimization algorithms are used to optimize the test cases. Genetic Algorithm (GA) is an optimization method imitated the biology heredity and evolution [3]. A new approach is presented utilizing program dependence analysis techniques and GA to generate the Boolean variables or enumerated types as described by [4]. And the effectiveness and efficiency is proved. But this approach is unable to test the security of the software.

The relatively complete coverage (RCC) principle discussed in [5] provides only a guideline for systematic security testing. The Randomized Algorithms (RA) is general applied by most of the security testing methods currently. But RA is just generated abundant Pseudo-random data for the target software, in order to get the best or the worst results. This algorithm has lots of defects such as too much testing data, low efficiency and deficiently testing, etc. A new algorithm is designed with the GA based on the RCC. The intention is to optimize test cases and to enhance the efficiency.

II. BASIC CONCEPTS

A principle for software security testing is proposed by RCC. Then the main contents are described chiefly as follows.

Let P be a part of program, and let D and R denote its input domain and its range, respectively. RC represents the run-time constraints for P. For simplicity, we assume that P behaves as a partial function with domain \( D \times 2^{RC} \) and range R. Let SR denote the security requirements for P. Test all the paths in Fp in all the inputting domains. Fp is a finite state machine capture all the behavior of P.

Def. 1 The given set \( d \in D \), the run-time constraints \( rc \in 2^{RC} \), P is said to be relatively secure for \( d \) with respect to \( rc \) if any computing path leading P(d) in Fp satisfies SR. P is complete security if and only if it is relatively security for every \( d \in D \) arbitrary and every \( rc \in 2^{RC} \).

Def. 2 The presence of a security defect or vulnerability is demonstrated by showing that P is not relatively security for some \( d \in D \) under some constraints \( rc \in 2^{RC} \), that is, it does not satisfy the security requirements.

Def. 3 A test case is an element in \( D \times 2^{RC} \). A test case T is a finite subset of test cases. P is relatively security for T if it is relatively security for all elements in T. If P is not security, it means there exists a \((d, rc)\) in T is insecure for \((d, rc)\).

Def. 4 If the input domain can be divided into classes such that the elements of a given class are expected to behave in exactly the same way with respect to a particular run-time constraint, then we could select one single test case from each class as representative and form all these test cases into a test set C, called a test criterion, it satisfied the RCC.

III. TEST DATA GENERATION BASED ON GA

A. Initialization and preprocessing

Setting \( SR \) to be security requirements need to be covered, \( RC \) to be run-time constraints sets; assuming \( rc \) is subset of \( RC \). All test data \( d \in D \) are optimized in every test criterion C. The test data \( d \) is coded in symbol encoding. It is fit for coding the data that has no sense of numerical value but only has the sense of code [6].
B. Calculation of fitness value

Initial population is generated by RA at the primary of the algorithm run-time. The fitness function is used to calculate the fitness of every chromosome and decide the select probability of chromosome in the initial population. The process is described by the following steps:

1) Implementing the current population.
2) Recording the safety requirements \( n \) meeting SR.
3) Comparing number of safety requirements \( n \) in SR, recording safety fails to meet requirement.
4) Calculating the fit of the current test data with Eq. (1),

\[
fit = \frac{f_i}{\sum_{j=1}^n f_j} \tag{1}
\]

5) The next test data is selected to calculate. When fitness is calculated, safety requirements are recorded during execution of the test data. These recorded requirements are compared with target requirements, getting same number as fitness value. If test data \( i \) contains security requirements more than test data \( j \), then gives a high fitness to \( i \).

C. Gene Selection

Gene is the element of the chromosome which is used to show the characters of the individual [7]. A chromosome is a test data \( d \). By calculating the fitness of chromosome, which has the excellent genes are selected to generate new individuals.

Calculate the current group fitness value \( f_i \) and select individuals from the current group as parent generation. The individual will have a high probability of being selected or being selected several times if it has a high degree of fitness. Otherwise the individual will have a low probability of being selected or even not being selected. The probability of selection \( P_i \) is decided according to the fitness value \( f_i \) of the test data \( d_i \) in this algorithm.

\[
P_i = \frac{f_i}{\sum_{j=1}^n f_j} \tag{2}
\]

In Eq. (2) \( n \) is the number of the test data in the group and,

\[
\sum_{j=1}^n P_i = 1.
\]

In the selection, a random number \( r(0 \leq r < 1) \) is generated in this algorithm and every test data \( d_i \) corresponds to two select numbers,

\[
m_j = \sum_{i=1}^j P_i \quad \text{and} \quad m_{j+1} = \sum_{i=1}^{j+1} P_i \quad (m_j \leq r < m_{j+1})
\]

D. Chromosome Crossover and mutation

A Chromosome is also called genotype individual [8]. The chromosomes which are chosen to use single-point crossover to generate new individuals. In order to prevent the population evolution stopped or degeneration, minimum is draw into the population. That will prevent the population constringing too fast to influence the evolution of the whole population.

Mutation has strong destructibility so that the frequency of mutation is much less than crossover. Single-point crossover and mutation is the characteristic manufacture in GA. It is easier to be achieved and speed up the process of covering goal.

The new generation is achieved by repeating crossover and mutation. When produce offspring, two parents crossover generates two children and a parent mutation generates a child. The number of parents determines and equals the number of kids.

E. Design of Algorithm

The process of the specific algorithm is given based on the above discussion.

1) Generate initial test cases:
Divide the input domain \( D \) into classes \( D_i \) \((i=1, 2, ... , n)\) such that the elements of a given class are expected to behave in exactly the same way for any given \( r \) in \( RC \) and \( D = \bigcup D_i (i=1, 2, 3, ..., n) \). Form a test criterion \( C = (d_i, r_c) \).

2) Optimization of test population:
Test criterion \( C \) conducted on the symbol encoding to form the first generation of test group \( t \), and calculation fitness of each chromosome in the group. Chromosomes are operated to generate a new group \( t+1 \) through selection, crossover and mutation.

3) Test optimization results:
Judge the results whether can cover the target SR or not, if they do, exit. Or judge whether the iterations is bigger than \( N \).

P will be tested by the test cases sets if it can cover the domain SR. If \( P \) could satisfy SR, then \( P \) is relatively security for the test sets in SR. Otherwise, \( P \) has a security defect. Fig. 1 depicts the algorithm.

IV. EXPERIMENTS

The Web Browser provides a GUI to access the Web and local data for users [9]. It should have the function which could avoid attacking from the network based on the vulnerability of the security. Let’s take Web Browser as a software application to demonstrate how to apply RCC principle to test its security. Test cases generated
based on GA could be used to test the security problems exist in Web Browser. Calculate the fitness of the data evaluate all the data of the currently population according to the SR.

\[ SR = \{sr_1: \text{DoS attack}; \quad sr_2: \text{Buffer overflow} \}; \quad sr_3: \text{long-distance command executing}; \quad sr_4: \text{round cross-domain constrain}; \quad sr_5: \text{round security area}; \quad sr_6: \text{address filed deceive}\} \]

\[ RC = \{mc, dc, nc, dl, rt, df, rf\}, \] where mc: memory constrain, dc: disk space constrain, nc: net constrain, dl: database logging on constrain, rt: registry tamper, df: disturb file, rf: replace file. If \( rc = \{mc, dc, nc\} \) can be satisfied, then select other constrain condition in RC as the constrain subset such as \( rc_j = \{dl, rt\}, rc_2 = \{dl, df, \}

The attacker is Web Server for Web Browser and all the content transferred by the server can be explained by the browser. All the content transferred by the server is made up of a test data set \( D \). Test Web Browser with a test case \( C \). The goal of generating test case is to cover 100% security requirement SR with the least test case. A large number of the experiments were done to verified that the most iterate number is less than 100, so we give the iterate number \( N = 100 \). The initial population is described as Table 1.

Table 1 shows whatever input subset can’t cover \( sr_5 \), so \( sr_5 \) is the target that the test case should be evolution based on GA. After the program executed there are some test case generated in test case set \( C_S = \{D_s, rc_j\} \) which can cover the \( sr_5 \). Then all the security requirements are covered by test cases.

Ten experiments have been done in different initial population in RA and GA, respective. Fig. 2 shows the results and the contrast of the two algorithms.

Table 1. Initial Population

<table>
<thead>
<tr>
<th>Test Case Sets</th>
<th>Subsets of Input</th>
<th>Subsets of Constraints</th>
<th>Test Cases</th>
<th>Covering SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 )</td>
<td>( D_1 )</td>
<td>( rc_1, rc_2, ..., rc_6 )</td>
<td>( C_1 = (d_1, rc_i) ... C_1 = (d_{120}, rc_6) )</td>
<td>( SR_1, SR_2 )</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>( D_2 )</td>
<td>( rc_1, rc_2, ..., rc_6 )</td>
<td>( C_2 = (d_1, rc_i) ... C_2 = (d_{120}, rc_6) )</td>
<td>( SR_2 )</td>
</tr>
<tr>
<td>( C_3 )</td>
<td>( D_3 )</td>
<td>( rc_1, rc_2, ..., rc_6 )</td>
<td>( C_3 = (d_1, rc_i) ... C_3 = (d_{120}, rc_6) )</td>
<td>( SR_3, SR_4 )</td>
</tr>
<tr>
<td>( C_4 )</td>
<td>( D_4 )</td>
<td>( rc_1, rc_2, ..., rc_6 )</td>
<td>( C_4 = (d_1, rc_i) ... C_4 = (d_{120}, rc_6) )</td>
<td>( SR_4 )</td>
</tr>
<tr>
<td>( C_5 )</td>
<td>( D_5 )</td>
<td>( rc_1, rc_2, ..., rc_6 )</td>
<td>( C_5 = (d_1, rc_i) ... C_5 = (d_{120}, rc_6) )</td>
<td>( SR_5 )</td>
</tr>
<tr>
<td>( C_6 )</td>
<td>( D_6 )</td>
<td>( rc_1, rc_2, ..., rc_6 )</td>
<td>( C_6 = (d_1, rc_i) ... C_6 = (d_{120}, rc_6) )</td>
<td>( SR_6 )</td>
</tr>
<tr>
<td>( C_7 )</td>
<td>( D_7 )</td>
<td>( rc_1, rc_2, ..., rc_6 )</td>
<td>( C_7 = (d_1, rc_i) ... C_7 = (d_{120}, rc_6) )</td>
<td>( SR_7, SR_8 )</td>
</tr>
</tbody>
</table>

The y-axis shows how many test data have been executed before \( sr_5 \) was covered. And the x-axis shows the serial number of the experiments. After analyzing, we can see the number of test data used GA is much less than RA.

Table 2 shows the comparing of the experiments data of the two algorithms.

Table 2. Data of Experiments

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Result of Experiment</th>
<th>Average Number(n)</th>
<th>Average Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA</td>
<td>1319000</td>
<td>1.442</td>
<td></td>
</tr>
<tr>
<td>GA</td>
<td>634000</td>
<td>2.104</td>
<td></td>
</tr>
</tbody>
</table>

The Average Number (m) is the number of test data which have been executed when the vulnerability was discovered based on \( sr_5 \). The Average Time (s) is the time that the test data costs. Table 2 shows that the number of the test data that RA needs is twice more than the GA’s. The time cost by RA is more than the GA. So generate security test cases based on GA is effective. It is worth doing the further research based on this method.

V. CONCLUSIONS AND FUTURE WORKS

This article is about test cases generated for software security based on GA. The method we design is to select...
the most favorite test cases in every $D_i$ in order to discover the vulnerabilities in the software. This method can reduce the test time and improve the test efficiency to a certain extent.

There are still some issues. The convergence of the algorithm is not fast enough. Especially, the lower accuracy of the test case generation is a problem for much more complex security requirements. The future work is the improvement of the encoding and the fitness of the chromosome in order to further improve the test efficiency and accuracy.

REFERENCES