基于回溯与引导的关键代码区域覆盖的二进制程序测试技术研究

崔宝江[①②]　梁晓兵*[①]　王　禹[②]　王建新[③]
[①](北京邮电大学计算机学院　北京　100876)
[②](中国信息安全测评中心　北京　100085)
[③](北京林业大学信息学院　北京　100083)

摘　要：基于路径覆盖的测试方法是软件测试中比较重要的一种测试方法，但程序的路径数量往往呈指数增长，对程序的每一条路径都进行测试覆盖基本上是不可能的。从软件安全测试的观点看，更关心程序中的关键代码区域(调用危险函数的语句、圈复杂度高的函数、循环写内存的代码片断)的执行情况。该文提出了覆盖关键代码区域的测试数据自动生成方法，该方法基于二进制程序，不依赖于源码。通过回溯路径获取所有可达关键代码区域的程序路径，并通过路径引导自动为获得的路径生成相应的测试数据。路径引导策略基于程序的符号执行与实际执行，逐步调整输入，使用约束求解器生成相应的测试用例。理论分析与实验结果显示该文给出的方法可以降低生成测试数据所需要的运行次数，与传统的覆盖路径测试数据生成方法相比，所需要的运行次数显著降低，提高了生成测试数据的效率。

关键词：测试数据自动生成；关键代码区域；符号执行与实际执行；路径回溯与引导
中图分类号：TP309　　**文献标识码**：A　　**文章编号**：1009-5896(2012)01-0108-07
DOI: 10.3724/SP.J.1146.2011.00532

The Study of Binary Program Test Techniques Based on Backtracking and Leading for Covering Key Code Area

Cui Bao-jiang[①②]　Liang Xiao-bing[①]　Wang Yu[②]　Wang Jian-xin[③]
[①](*School of Computer*, *Beijing University of Posts & Telecommunications*, *Beijing* 100876, *China*)
[②](*China Information Technology Security Evaluation Center*, *Beijing* 100085, *China*)
[③](*School of Information Science & Technology*, *Beijing Forestry University*, *Beijing* 100083, *China*)

Abstract: Path traverse is a kind of important software testing method of software test. However, as the number of paths of software is usually exponential, to test every path is unpractical. From the point view of software security test, the execution of critical code fragments in the binary program is more interested. The critical code fragments are the statements which call the danger function, the functions with high cyclomatic complexity and the code fragments with loop-writing memory. In this paper, a data auto-generation method is presented, which covers the critical code area, this approach is based upon binary program and does not need the source code of the test program. These paths which can reach the critical code areas are automatically obtained by a method called path backtracking, and are automatically generated test data for these paths by a method called path leading. It is based on the symbolic execution and concrete execution, regulates the test input step by step and uses the constraint solver to generate the test cases. Theory analysis and experiment results indicate that the method of path leading can reduce the execution number of test data generation contrast with existing methods of generating test data for a given path, the method of path leading improves the efficiency of test data generation.

Key words: Test data auto-generation; Critical code area; Symbolic execution and concrete execution; Path backtracking and path leading

1 引言

测试数据生成是软件测试中的关键技术之一，它的实现对于软件测试过程的自动化具有重要意义[1]。路径覆盖测试是软件测试中比较重要的一种测试方法。路径覆盖要求程序的每条可能路径都至少执行一次，如果程序中有循环，则要求每个循环至少覆盖一次。但事实上，程序中的路径数量往往呈指数增长。对程序的每一条路径都进行测试是不现

2011-06-02 收到，2011-08-29 改回
国家自然科学基金(61170268, 61070207)资助课题
*通信作者：梁晓兵　liangxbg@tom.com

实的[2]。从软件安全测试的观点看，更关心程序中关键代码区域的执行情况，关键代码区域主要有 3 类：(1)调用危险函数的语句；(2)圈复杂度[3]高的函数；(3)循环中有写内存操作的代码片段。

对于有源码的程序，覆盖路径的测试用例的生成方法有随机法、静态法和动态法[4]。随机法在程序的输入空间中不断地随机选取程序输入向量来运行被测程序，试图找到满足需求的测试数据。该方法容易实现且对简单程序有较好的效果，但对于实际的应用程序很难以合理的成本生成有效的测试数据[5]。

静态法不需要运行被测程序，而是对程序进行静态分析和转换得到相应的约束系统，求解约束系统即可得到测试数据。主要包括符号执行法[6]、区间算术法[7]等。静态法要进行复杂的代数运算并且难以处理数组、不定长循环、指针别名和指针引用等问题，在实际应用中存在困难[4]。

动态法基于程序的实际运行，主要利用程序运行过程中收集的信息来判断当前的输入向量在多大程度上能够满足特定的测试需求，并逐步调整当前的输入向量试图找到满足测试需求的测试数据。具体包括遗传算法[1,5,8]，迭代松弛法[9]，直线式程序法[10]，以及 Korel 方法[11]等。由于在程序运行时各输入变量值已经确定，故动态法可以克服静态法面临的一些困难。但是，当问题有解时，动态法不保证总能找到。

本文提出了覆盖被测程序关键代码区域的基于符号执行与实际执行的路径回溯与引导的测试用例生成方法。该方法首先对二进制代码进行静态分析，识别出二进制代码中的关键代码区域，生成二进制代码的控制流图和函数调用图，然后采用路径回溯的方法回溯出从关键代码区域到程序输入点的所有路径。最后基于反馈机制，将符号执行与实际执行相结合使用路径引导的方式为每条指定的路径生成相应的测试用例。本文给出的方法基于二进制程序，不需要源码，并且生成测试数据的整个过程完全自动化，不需要额外的人工干预。

2　覆盖关键代码区域的测试用例生成方法

本节阐述覆盖被测程序中关键代码区域的测试用例生成方法。该方法可以为被测程序中通过关键代码区域的所有路径生成相应的测试用例。该方法主要分为 3 个模块：关键代码区域与输入点的识别模块、路径回溯模块与基于符号执行与实际执行的路径引导模块。

2.1 关键代码区域与输入点识别

从安全测试的观点看，程序中的关键代码区域主要有 3 类：(1)调用危险函数的汇编指令；(2)圈复杂度高的函数；(3)循环中有写内存操作的代码片段。被测程序中关键代码区域的识别算法流程如图 1 所示：首先对被测程序进行静态分析，获得被测程序中危险函数的列表。对其中的每一个函数，分别计算其圈复杂度，可以识别出圈复杂高的函数。在对被测程序进行静态分析的基础上，利用循环检测算法对被测程序进行静态分析，从中提取循环写内存的代码片段的相关信息。

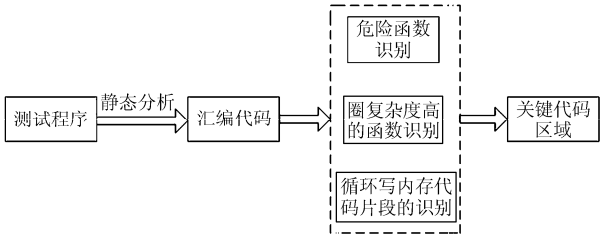

图 1 关键代码区域识别

程序的执行路径受外部输入的影响，输入点是程序中读取外部输入数据的程序语句，即程序中调用输入函数读取外部输入的程序语句，它是路径回溯的终点。使用调试器加载被测试程序，通过导入表监控程序的输入函数，获取输入函数的位置。

2.2 路径回溯

路径回溯是一种自动获取被测程序关键路径的方法，它采用静态分析的方法首先获取二进制程序的控制流图、函数调用图及关键代码区域的位置，采用路径回溯的技术，从关键代码区域开始，自动地获取从二进制程序关键代码区域到程序输入点的所有程序路径。

2.2.1 静态分析　静态分析是通过遍历 IDAPro[12]产生的反汇编代码，采用静态分析的方式，将代码中的所有指令按基本块进行划分，根据汇编指令的跳转方式获取基本块的块头、块尾以及跳转信息，如图 2 所示。同时，静态分析还可以获取函数调用信息，函数调用信息形式为：Address1 call Address2，即在地址 Address1 处调用了地址 Address2, Address2 为被调用函数的起始地址。

2.2.2 回溯前驱块，获得回溯路径　路径回溯就是从关键代码区域所在基本块开始回溯前驱基本块，直至回溯到程序的输入点，从而获得从输入点起到关键代码区域的所有程序路径。

基本块 A 的前驱基本块 B 是指，它于基本块 A 之前执行，基本块 A 紧接着它执行。通过在基本块

基本块 1	基本块 2	基本块 3	基本块 4
Head:01009968	Head:010099f4	Head:01009a04	Head:01009a21
Tail:010099f2	Tail:01009a02	Tail:01009a1f	Tail:01009a27
Exit1:01009a45	Exit1:01009a3d	Exit1:01009a2c	Exit1:01009a2c
Exit2:010099f4	Exit2:01009a04	Exit2:01009a21	Exit2:01009a29
JxxA:010099f2	JxxA:01009a02	JxxA:01009a1f	JxxA:01009a27

图 2 mspaint.exe 部分基本块信息

信息中的跳转分支地址中搜寻此基本块头地址，就可以获得该基本块的前驱基本块。在图 2 中，基本块 2 的块头地址是 0x010099f4, 基本块 1 在执行结束后紧接着运行的地址可能为 0x010099f4 或 0x01009a45，即执行基本块 2。则基本块 1 是基本块 2 的前驱基本块。一个基本块的前驱基本块可能不止一个，也可能是多个。

从静态分析得到的基本块信息可以查找到前驱基本块，回溯获得到达关键代码区域的程序路径。然而若关键代码区域是在某一函数内，此函数的起始地址所在的基本块没有前驱基本块，此时通过回溯前驱块的方法，只能回溯到该函数的起始基本块，得出从函数起始处到达关键代码区域的所有程序路径。要获得从程序输入点到关键代码区域的完整路径，还需要在回溯到函数起始基本块后，在函数调用信息中搜索调用此函数的地址，从而得以进一步向前回溯路径，直至回溯到程序的输入点所在的基本块，从而获得从程序的输入点到关键代码区域的所有路径。

2.3 基于符号执行与实际执行的路径引导策略

路径引导的过程是为指定路径生成测试数据的过程。路径引导基于程序的符号执行，并将实际执行结合到符号执行中。它的基本思想就是在程序符号执行的过程中获取路径约束条件，从回溯到的路径集合中选取一条路径为引导路径，借助反馈机制，逐步调整输入使得程序沿着引导路径实际运行。

2.3.1 输入处理 程序的执行路径受外部输入的影响，这些外部输入的数据即为对程序进行符号执行时的符号变量。输入点即程序获取外部输入数据的输入函数调用处，也为符号执行的起始点。进行符号执行前需要先确定输入点和输入数据的大小，标记符号执行中的符号变量，并把这些符号变量定义为原始符号变量集合$\{\mathrm{var}_1, \mathrm{var}_2, \cdots, \mathrm{var}_n\}$。

2.3.2 符号执行与实际执行 确定符号执行的符号变量后，程序开始符号执行与实际执行。符号执行可以收集路径约束条件，获取输入与程序运行路径的对应关系信息。在外部输入数据读入到内存后程序进入执行状态，对于即将执行的每一句指令，首先判断是否涉及符号变量操作，如果涉及符号变量的操作，则进行符号执行，否则只需实际执行以保持和符号执行同步即可。这里的符号变量包括原始符号变量和中间符号变量，中间符号变量是指由原始符号变量表达式来表示的符号变量。

符号执行与实际执行相结合的过程可以抽象为

(1)定义原始变量集合$\{\mathrm{var}_1, \mathrm{var}_2, \cdots, \mathrm{var}_n\}$；

(2)存在一系列映射 fxxx()，其中 xxx 代表了 x86 指令集，映射$\{\mathrm{fxxx}(\mathrm{var}_i)|i=1,2,n\}$为中间变量集合；

(3)根据程序跳转点对变量的约束，为原始变量和中间变量建立起一个方程组，即程序的路径约束条件。

2.3.3 路径引导 设二进制程序从输入点到关键代码区域的执行轨迹为 $P=(P_1,P_2,\cdots,P_L)$，路径约束条件为 $PC=(PC_1,PC_2,\cdots,PC_L)$，生成路径的影响变量为 $V=(X_1,X_2,\cdots,X_L)$，程序对变量 $X_i, i\in[1,L]$ 的修改操作为 G，则有

$$G(X_i)=\begin{cases}X_i, & G\text{为恒等变换}\\ g(X_i), & G\text{不是恒等变换}\end{cases}$$

路径引导算法如表 1 所示。

表 1 路径引导算法

(1) PATH-LEADING(P)
(2) **input**: path orbits
(3) **ouput**: path constraint conditions
(4) **begin**
(5) **foreach** $X_i \in V$ **do**
(6) **if**($G(X_i) == P_i$) **then**
(7) **return** PC_i
(8) **else**
(9) X_i = REGULATE(X_i , PC)
(10) **repeat** 6
(11) REGULATE(X_i , PC)
(12) **input**: variable X_i , path constraint condition PC
(13) **output**: variable X_i
(14) **begin**
(15) **if** (SOLVECONS($\neg PC$))//SOLVECONS: 求解约束条件，获得测试输入
(16) **return** X_i =SOLVECONS($\neg PC$)
(17) **else**
(18) **return** This path is unfeasible

3　算法分析与实验验证

本节首先对随机法、爬山法、遗传算法和本文给出的算法需要被测程序实际执行次数进行理论上的分析与比较，然后针对一个具体的二进制程序，对上述算法需要被测程序实际执行的次数进行比较，从两方面的结果看，本文给出算法需要二进制程序实际运行的次数是最少的，如表 2 所示。

表 2 路径测试数据生成方法所需程序执行次数复杂度比较

算法	复杂度
随机法	$O(m^L)$
爬山法	$\geq mL$
遗传算法	$>L$
本文算法	$O(L)$

3.1 算法分析

基于程序实际执行的路径测试数据生成方法有随机法、爬山法与遗传算法。采用这些方法所需要的被测程序运行次数均与路径的深度正相关。本节首先给出了爬山法[13]和遗传算法[8]的算法描述，然后对这些算法需要程序实际执行次数的复杂度进行了比较。

基于爬山法的测试用例生成算法如表 3 所示。基于遗传算法的测试用例生成算法如表 4 所示。

表 3 基于爬山法的测试用例生成算法

```
(1) HILL-CLIMING(P)
(2) input: The input data
(3) output: The test case
(4) begin
(5)     CurrentNode = InitialValue
(6)     foreach Node N in the path do
(7)       foreach Neighbor(N) do
(8)         if(Eval(Neighbor(N))<=Eval(CurrentNode) then
(9)           Restart(P)
(10)          Repair(the input data)
(11) return the test case
```

表 4 基于遗传算法的测试用例生成算法

```
(1) GENETICALGORITHM(P)
(2) input: The test case in the initial population
(3) output: the test case
(4) begin
(5)     foreach test case n in the initial population N do
(6)       ComputeFitness(n)
(7)       if(ComputeFitness(n) == 0) then
(8)         return n
(9)       else
(10)          N = Select(m)
(11)          repeat 5
```

对于爬山法，假设被测程序只有一个输入点，从程序输入点到关键代码区域的路径深度为 L，每一层的节点数目分别为 m_i，其中 $1 \leq i \leq L, m_i \geq 1$, $m = \min\limits_{1\leq i\leq L}\{m_i\}$，则在最好情况下，爬山法需要被测程序执行的次数复杂度为

$$T_{\text{clihill}} = \sum_{i=1}^{L} m_i \geq L \cdot \min_{1\leq i\leq L}\{m_i\} \geq mL$$

对于遗传算法，假设从程序输入点到关键代码区域的路径深度为 L，路经节点的约束条件分别为 α_i ,输入 X 在每一个节点的实际的路经约束条件分别为 $g(X_i)$，则基于遗传算法的数据生成问题可以转化成求：$f(X) = \sum_{i=1}^{L} |g(X_i) - \alpha_i|$ 的最小值。

设遗传操作对输入 $X_{(i)}$ 的每一位改变的概率为 P_m，则输入 $X_{(1)}$ 经过遗传操作变成输入 $X_{(2)}$ 的概率为 $P_m^k(1-P_m)^{n-k}$，其中输入 $X_{(1)}$ 和输入 $X_{(2)}$ 有 k 比特差异。对于任意输入 $X_{(i)}$，任意小的数 $\eta > 0$，则输入 $X_{(i)}$ 经过遗传操作变成 $f(X) = \sum_{i=1}^{L} |g(X_i) - \alpha_i|$ 最优解的概率为 $P_{X(i)} = P_m^k(1-P_m)^{L-k}$，输入 $X_{(i)}$ 和最优解有 k 个比特的差别，经过 x 次遗传操作不是最优解的概率为 $1-(1-P_m^k(1-P_m)^{L-k})^x$，若该值可以任意小，即输入 $X_{(i)}$ 经过 x 次遗传操作可以无限逼近最优解。即有

$$1-(1-P_m^k(1-P_m)^{L-k})^x < \eta$$

可以解得

$$x > \log_{(1-P_m^k(1-P_m)^{L-k})}(1-\eta) \tag{1}$$

由式(1)，令 $y1 = \log_{(1-P_m^k(1-P_m)^{L-k})}(1-\eta)$，其中 k 表示初始输入与最优解有 k 个比特差异，η 是一个大于 0 的任意小的实数，y 是关于 P_m 和 L 的一个函数。函数 $y1 = \log_{(1-P_m^k(1-P_m)^{L-k})}(1-\eta)$ 与函数 $y2=L$ 的图像如图 3 所示，可以看出 $\log_{(1-P_m^k(1-P_m)^{L-k})}(1-\eta)$ 的值远大于 L，即 x 大于 L，其中 $k = 500$, $\eta = 0.001$ 。

3.2 实验验证

本节分别对随机法、爬山法、遗传算法及本文

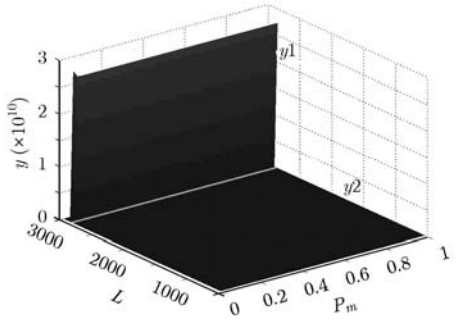

图 3 函数 $y1 = \log_{(1-P_m^k(1-P_m)^{L-k})}(1-\eta)$ 与函数 $y2 = L$ 的对比图

给出的算法需要程序实际执行的次数进行了对比实验，具体实验结果如表 5 所示。

表 5 实验对比

算法	平均运行次数(次)
随机法	—
爬山法	232
遗传算法	127614
本文算法	7

3.2.1 验证程序 本文给出一段二进制代码 CipVer.exe，该代码片段用于密码验证的代码，本文给出的方法是基于二进制代码，不需要源代码。

图 4 为该程序的部分流程图，图中节点 0x004114f8 有输入函数调用语句 call _fscanf，程序在此处获取外部输入，节点 0x004114f8 为输入点。节点 0x004114f8 有两个分支，右边的分支节点 0x00411551 是密码验证成功的分支，左边的分支节点 0x00411538 是密码验证失败的分支。设定密码验证成功的节点为该二进制程序的关键代码区域，通过回溯路径得到从输入点到目标节点的一条引导路径 0x004114f8→⋯→0x00411551。

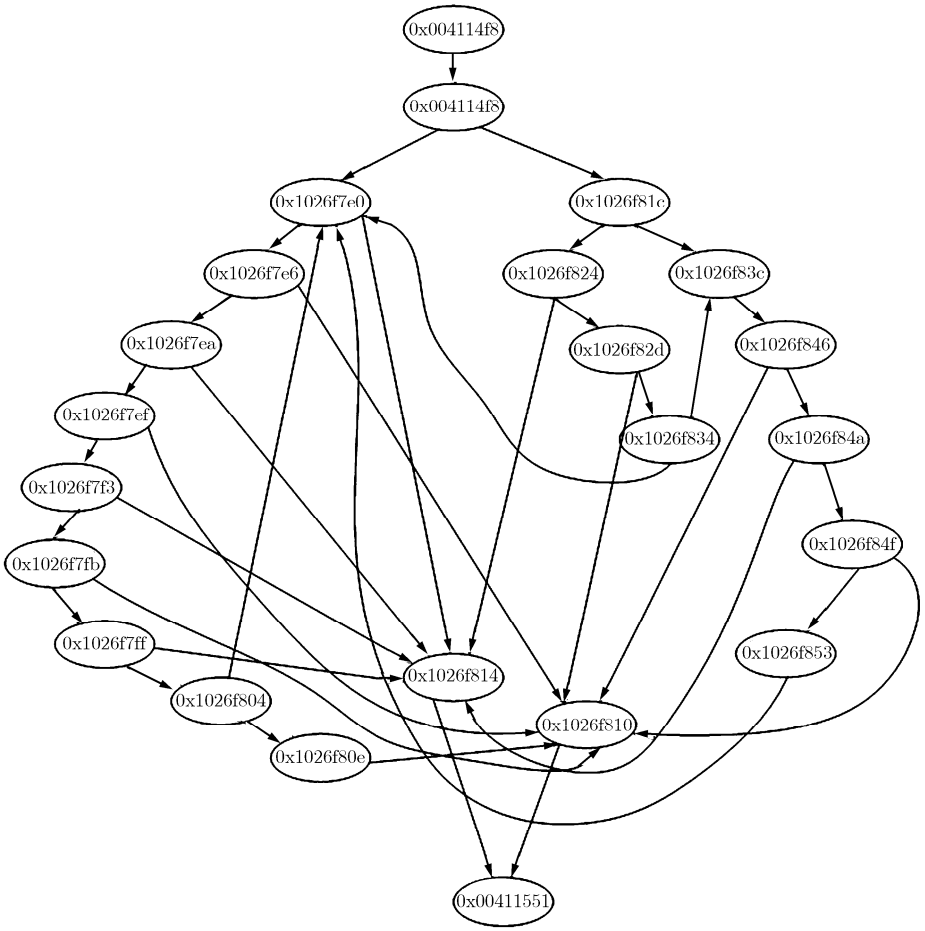

图 4 CipVer.exe 的部分控制流图

节点 0x004114f8 处还有函数调用语句 call Sub_401000, Sub_401000 函数即为源码中的函数 verify_password。函数 verify_password 把用户输入的密码与程序内置的密码“1234567”的相应字节进行对比验证。如果不相同，密码验证失败，否则密码验证成功。

符号执行与实际执行通过不断调节输入，完成路径引导最终获得关于输入数据的部分 STP[14]语言形式的路径约束条件如图 5 所示。

求解的结果即为能够通过密码验证的输入数据。图 6 中数据的显示是 16 进制的。可知，输入数据的第 1 个字节的值为 0hex31, 0hex31 的十进制形式是 49，代表了 ASCII 码的字符“1”，0hex32 代表了 ASCII 码的字符“2”，往下依次类推，可知当

```
ASSERT(( BVSUB(8, a[0hex003d2370],0hex31)=0hex00));
ASSERT(NOT((a[0hex003d2370]=0hex00)));
ASSERT(( BVSUB(8, a[0hex003d2371],0hex32)=0hex00));
ASSERT(NOT((a[0hex003d2371]=0hex00)));
ASSERT(( BVSUB(8, a[0hex003d2372],0hex33)=0hex00));
ASSERT(NOT((a[0hex003d2372]=0hex00)));
ASSERT(( BVSUB(8, a[0hex003d2373],0hex34)=0hex00));
ASSERT(NOT((a[0hex003d2373]=0hex00)));
ASSERT(( BVSUB(8, a[0hex003d2374],0hex35)=0hex00));
ASSERT(NOT((a[0hex003d2374]=0hex00)));
ASSERT(( BVSUB(8, a[0hex003d2375],0hex36)=0hex00));
ASSERT(NOT((a[0hex003d2375]=0hex00)));
ASSERT(( BVSUB(8, a[0hex003d2376], 0hex37)=0hex00));
```

图 5 STP 语言形式的路径约束条件

```
ASSERT(a[0hex003D2370] = 0hex31);
ASSERT(a[0hex003D2371] = 0hex32);
ASSERT(a[0hex003D2372] = 0hex33);
ASSERT(a[0hex003D2373] = 0hex34);
ASSERT(a[0hex003D2374] = 0hex35);
ASSERT(a[0hex003D2375] = 0hex36);
ASSERT(a[0hex003D2376] = 0hex37);
Invalid.
```

图 6 STP 约束求解器输出的解

输入字符为“1234567”时，通过密码验证，程序到达正确的目标点。

采用本文提出的路径引导的方法借助反馈机制最多需要对输入数据做 7 次调节，即可生成正确的输入密码，击中通过验证的目标节点。这里的对输入数据的 7 次修改的位置是在函数 verify_password 内完成对输入各个字节的验证。本文给出的算法生成测试数据最多需被测程序运行 7 次，可以有效地生成覆盖关键代码位置的测试数据。

3.2.2 结果分析　在 3.2.1 节的实例中，如果采用随机法生成测试数据，即使将密码由较长的字符串“1234567”改为较短的“123”，需要被测程序运行 20 多万次(235363)才会击中通过验证的目标节点；密码为“1234567”时，运行时间极长(每次运行时间大于 12 h)。

采用爬山法，为 3.2.1 节中的实例生成路径测试数据，测试 1000 次，得出生成路径测试数据需要被测程序平均运行的次数为 213 次。这里由于实例程序中的路径谓词单一，采用爬山法很有效。在程序中的路径谓词多样性的情况下，爬山法难以有效发挥效用，需要的平均运行次数会大大的增加。

采用遗传算法，为 3.2.1 节中的实例生成路径测试数据，生成路径测试数据需要被测程序平均运行的次数 1000 次实验的平均值为 127614。采用的策略是：(1)交叉概率 90%；(2)突变概率 40%; (3)位操作；(4)精英策略。

采用本文提出的路径引导的方法，在密码为“1234567”时，最多需运行 7 次，平均运行时间只有 25 s，大大降低了运行次数，运行效率远高于爬山法和遗传算法，大大提高了生成效率。

4　总结

从软件安全测试的观点看，有针对性地生成测试数据覆盖被测程序中关键代码区域，可以提高软件测试的效率。本文给出了一种基于路径回溯与引导的关键代码区域覆盖的测试数据自动生成方法，通过路径回溯与路径引导的方法，生成覆盖被测程序中关键代码区域的测试数据，该方法首先在静态分析的基础上回溯出从输入点到关键代码区域的路径,然后基于符号执行与实际执行，同时利用反馈机制不断调整输入，最终生成覆盖关键代码区域的测试用例。从理论分析和实验结果看，本文给出的基于路径回溯与引导的覆盖关键代码区域的测试数据生成方法，相对传统的测试数据生成方法具有较高的效率。

参 考 文 献

[1] Del Grosso C, Antoniol G, *et al*.. Detecting buffer overflow via automatic test input data generation [J]. *Computers & Operations Research*, 2008, 35(10): 3125–3143.

[2] Wang Tie-lei, Wei Tao, Zou Wei, *et al*.. TaintScope: a checksum-aware directed fuzzing tool for automatic software vulnerability detection [C]. 31st IEEE Symposium on Security and Privacy, Oakland, 2010: 497–512.

[3] McCabe T J. A complexity measure [J]. *IEEE Transactions on Software Engineering*, 1976, SE-2(4): 308–320.

[4] 单锦辉, 王戟, 齐治昌. 面向路径的测试数据自动生成方法述评[J]. 电子学报, 2004, 32(1): 109–113.
Shan Jin-hui, Wang Ji, and Qi Zhi-chang. Survey on path-wise automatic generation of test data [J]. *Acta Electronica Sinica*, 2004, 32(1): 109–113.

[5] Domínguez-Jiménez J J, Estero-Botaro A, García-Domínguez A, *et al*.. Evolutionary mutation testing[J]. *Information and Software Technology*, 2011, 53(10): 1108–1123.

[6] KING J C. A new approach to program testing[C]. Proceedings of the International Conference on Reliable software, New York: ACM, 1975: 228–233.

[7] 陈立前, 王戟, 侯苏宁. 单变量区间线性不等式抽象域[J]. 计算机学报, 2010, 33(3): 427–439.

Chen Li-qian, Wang Ji, and Hou Su-ning. An abstract domain of one-variable interval linear inequalities [J]. *Chinese Journal of Computers*, 2010, 33(3): 427–439.

[8] 崔宝江, 梁晓兵, 王建新. 基于整数遗传算法的整数溢出漏洞检测技术研究[J]. 清华大学学报, 2010, 50(s1): 1603–1608.
Cui Bao-jiang, Liang Xiao-bing, and Wang Jian-xin. Integer overflow vulnerability detection based on an integer-valued genetic algorithm [J]. *Journal of Tsinghua University Science and Technology*, 2010, 50(s1): 1603–1608.

[9] Gupta N, Mathur A P, and Soffa M L. Automated test data generation using an iterative relaxation method[J]. *ACM SIGSOFT Software Engineering Notes*, 1998, 23(6): 231–244.

[10] Miller W and Spooner D L. Automatic generation of floating-point test data[J]. *IEEE Transactions on Software Engineering*, 1976, SE-2(3): 223–226.

[11] Korel B. Automated software test data generation[J]. *IEEE Transactions on Software Engineering*, 1990, 16(8): 870–879.

[12] http://www.hex-rays.com/idapro/, 2011, 7.

[13] Russell Stuart and Norvig Peter 著, 姜哲, 金亦江, 张敏, 等译. 人工智能 —— 一种现代方法[M]. 第 2 版, 北京: 人民邮电出版社, 2010: 88–91.

[14] http://people.csail.mit.edu/vganesh/STP_files/stp.html, 2011,7.

崔宝江： 男，1973 年生, 副教授, 研究方向为软件安全与信息安全.

梁晓兵： 男，1978 年生，博士生，研究方向为软件安全.

王 禹： 男，1987 年生，硕士，研究方向为软件安全.

王建新： 男，1972 年生，副教授，研究方向为数据挖掘、人工智能.