

基于遗传算法的 Fuzzing 测试用例生成模型

王 丹, 高 丰, 朱鲁华

(江南计算技术研究所, 江苏 无锡 214083)

摘 要: 将遗传算法运用到 Fuzzing 的数据生成过程中, 根据具体的测试目标特点, 对遗传算法的各个部分进行了定义, 适应度函数的设计引入了代码覆盖率和多样性因素. 对模型进行初步实现, 实验证明了该模型在 Fuzzing 过程中引导测试方向的有效性.

关键词: Fuzzing; 安全测试; 遗传算法; 代码覆盖

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2011)05-0130-05

Fuzzing Test Cases Generating Model Based on Genetic Algorithm

WANG Dan, GAO Feng, ZHU Lu-hua

(Jiangnan Institute of Computing Technology, Wuxi 214083, China)

Abstract: In this paper, genetic algorithm is applied to data generation process of Fuzzing. According to the specific characteristics of the test objective, all parts of genetic algorithm are defined specifically. The code coverage and diversity are the two factors to be introduced into the design of fitness function. The model is implemented preliminarily, and the experiment proves the validity to guide the testing direction in the process of Fuzzing.

Key words: Fuzzing; security testing; genetic algorithm; code coverage

1 引言

Fuzzing 已经广泛的应用于各种软件的安全测试中, 这得益于其高度的自动化特性. 但 Fuzzing 本身具有盲目性, 在整个测试的过程中并没有一个明确的方向, 从而在面对巨大的测试例空间时导致测试效率很低. 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 Fuzzing 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例, 在保证其高度自动化的基础上使得 Fuzzing 的测试具有了方向性, 从而提高测试的效率. 遗传算法的适应度函数在整个测试过程中起到了决定性的作用, 文中采用基于代码覆盖率和多样性因素的适应度函数. 针对该思路提出了一个 Fuzzing 框架模型, 并通过初步实现的原型工具结合 IBM 的 Purecov 代码覆盖工具, 在 Linux 环境下

对其进行了测试验证.

2 Fuzzing 技术

Fuzzing 技术最早可追溯到 1989 年, 美国 Wisconsin - Madison 大学的 Barton Miller 教授将 Fuzzing 用来测试 UNIX 应用程序的健壮性. 目前, Fuzzing 已经被工业界逐渐接受并作为其产品发布前必须通过的测试步骤, 例如在 2006 年微软 70% 的漏洞是通过 Fuzzing 发现的. 传统的测试方法关注的是软件的正确性, 而 Fuzzing 关注的是发现软件潜在的脆弱性, 特别是可利用的漏洞. Fuzzing 定义: 一种通过提供非预期的半合法的输入并监视异常结果来发现软件脆弱性的方法. 由此定义可以看出, 严格的 Fuzzing 在分类上是属于主动测试中的黑盒测试.

最早的 Fuzzing 就是简单地向被测目标 (Implement Under Test, IUT) 发送大量的完全随机的数据, 直到引起 IUT 出现异常. 而现在绝大部分

Fuzzing 生成的测试例都是半合法的数据,以保证测试例不被 IUT 最初的输入过滤掉,直接进入 IUT 的内部,从而更加有效的触发潜在的脆弱性. Fuzzing 按测试例的产生方法分为两类:基于变异的,通过对获取到的合法数据样本应用变异技术来产生测试例;基于生成的,通过对 IUT 的输入格式进行建模的方法从头开始产生测试用例.

Fuzzing 的过程可以简单地总结为 TIGEMA (Targets, Inputs, Generate, Execute, Monitor, Analyze),具体分为识别目标、识别输入向量、生成 Fuzzing 测试例、执行测试例、监视异常、分析测试结果. 一个典型的 Fuzzing 框架如图 1 所示,需要说明的是识别输入向量部分. 由于 Fuzzing 是通过向 IUT 的交互接口发送半有效的数据来触发潜在的脆弱性,所以 Fuzzing 只能发现那些由外部输入可达的漏洞,即其测试范围必须在程序的攻击面(attack surface)中. 目前, Fuzzing 几乎已经覆盖了所有的攻击面,产生了针对各种攻击面的 Fuzzer,如文件 Fuzzer、网络协议 Fuzzer、浏览器 Fuzzer、命令行 Fuzzer、内存 Fuzzer、配置 fuzzer、环境变量 Fuzzer 等.

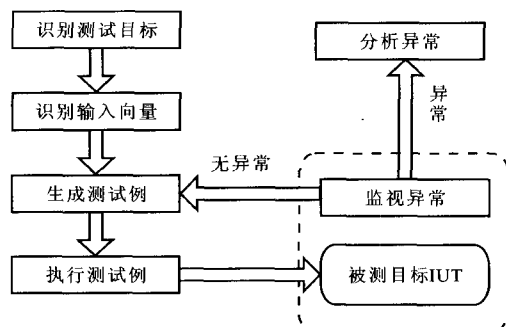


图 1 典型的 Fuzzing 框架

相对于其他的软件测试方法而言, Fuzzing 的优势体现在能够将大量的手工测试转换为高度的自动化测试,并且误报率很低,基本上通过 Fuzzing 触发的异常结果都会指向一个脆弱性. 同时 Fuzzing 存在很多缺点,如效率低、覆盖率难以确定、漏报率高、复杂脆弱点的发现困难、系统监视难、对发现的脆弱点进行定位分析难等. 这其中有些问题是 Fuzzing 测试方法本身所局限的,是不可能被改进的,比如无法检测到输入无法到达的代码部分所存在的脆弱点;而大部分的缺点可以通过采用各种理论方法来加以改进的.

通过分析各种 Fuzzing 工具,传统模式的 Fuzzing 只考虑了程序的输入格式和以往的测试经验,

忽视了测试过程中 IUT 反馈的大量有用的动态信息. 如果能够收集并利用这些动态信息,将极大地提高 Fuzzing 的智能化和测试效能. 从本质上看,如何从无限大的测试空间中找到好的 Fuzzing 测试数据的问题,是一个求解全局最优解的问题. 文中采用遗传算法与反馈的代码覆盖率信息相结合的方法,来实现一个更加自动化、智能化、高效的 Fuzzing 测试数据生成引擎.

3 遗传算法

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. GA 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,其核心思想是利用进化历史中获得的信息指导搜索或计算,优点是简单、通用、鲁棒性强和适于并行处理,近年来被逐渐应用到软件测试的测试数据生成中来.

遗传算法的原理是在由某种编码(coding)的个体(individual)组成的初始种群(population)上,循环使用交叉(crossover)、变异(mutation)操作来产生新的个体,并根据适应度(fitness)来进行优胜劣汰选择(selection)出新的种群,从而一代代的不断向前进化. 基本遗传算法 (Simple Genetic Algorithm, SGA) 可以定义为一个 8 元组: $SGA = (C, E, P_0, M, \Phi, \Gamma, \Psi, T)$. C 表示个体编码方法, E 表示个体适应度评价函数, P_0 表示初始种群, M 表示群体大小, Φ 表示选择算子, Γ 表示交叉算子, Ψ 表示变异算子, T 表示算法终止条件. SGA 的基本流程如图 2 所示.

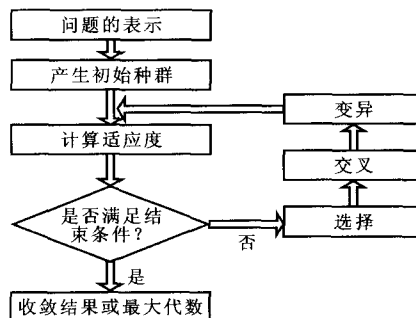


图 2 SGA 基本流程图

4 基于遗传算法的 Fuzzing 模型

文中的测试目标是 Linux 下的网络程序,系统框架的所有部分都是运行在 Linux 环境中.

4.1 基本框架模型

文中基于遗传算法的 Fuzzing 框架如图 3 所示,其中核心部分是生成测试例的 GAF(Genetic Algorithm Fuzzer),负责从 Mysql 中读取代码覆盖信息,并生成下一代测试例并发送到 IUT.

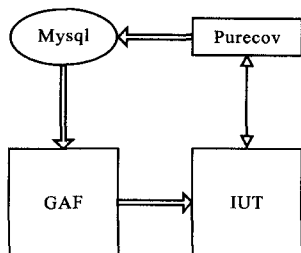


图 3 系统框架

框架的整体流程是,在测试例到达 IUT 并被执行的同时,代码覆盖工具 Purecov 通过监视 IUT 的执行过程获得测试例的代码覆盖信息,并把代码覆盖信息发送到 Mysql 中保存.当代的测试例执行结束后,GAF 会从 Mysql 读取当代测试例的代码覆盖信息,通过遗传运算得出下一代的测试例,紧接着执行下一代测试例.如此不停的运行,直到满足了停止条件.比如发现了一个漏洞或者某一代测试例的代码覆盖率达到了预定值.

4.2 Fuzzing 中的遗传算法

文中实现了一个基于遗传算法的 Fuzzing 框架模型,其中 GAF(Genetic Algorithm Fuzzer)模块是整个 Fuzzing 框架的核心,负责测试例的生成.图 4 说明了 GAF 中采用遗传算法生成测试例的基本流程.设种群 $G(p_1, p_2, \dots, p_n)$:

- (1) 首先通过对嗅探到的正常网络报文进行变异处理来生成初始种群 G_0 ;
- (2) 执行当代种群中所有的测试例;
- (3) 对当代种群中的测试例逐个计算代码覆盖率,并输入到适应度函数中计算适应度;
- (4) 判断在最大迭代次数内是否发现漏洞或者代码覆盖率达到某个预定值,是则结束,否则继续;
- (5) 按照适应度大小对当代种群中的测试例进行排序;
- (6) 采用轮盘赌法,从排序后的种群中选出 n 个测试例作为父代种群;
- (7) 在父代种群上进行交叉操作,生成 n 个新的测试例;
- (8) 按照变异概率对新的 n 个测试例进行变异操作;
- (9) 由此生成新的下一代种群,转到(2)继续.

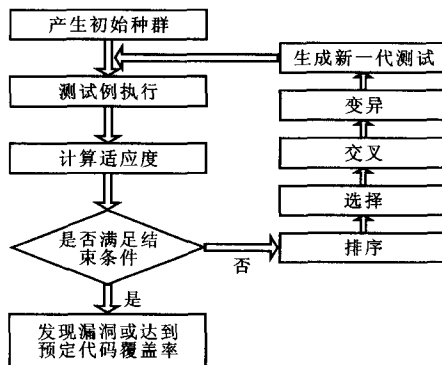


图 4 GAF 中的遗传算法

4.2.1 编码

编码是一个很关键的问题,遗传算法中进化过程是建立在编码机制基础上的.编码对于算法的性能,如搜索能力和种群多样性等影响很大.针对不同的问题,应该采用不同的编码与之相适应.

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;否则将会导致进化过程中个体的基因被破坏,无法将个体与编码相对应.如同在生物遗传过程中,染色体有基本的基因单元组成,如果在遗传过程中破坏了最基本的遗传单元,那么拥有此染色体(chromosome)的个体将是有缺陷的个体.在基于遗传算法的 Fuzzing 过程中,如果个体最基本的遗传单元被破坏,则该个体将会无法由被测试目标识别和接受,从而不会进一步的深入到程序内部,甚至会退化为随机数据,以至失去了遗传算法应起的作用.

GAF 的测试目标是网络程序,测试例是由数据报构成的,每一个数据报由多个最基本的协议单元组成,这些最基本的协议单元就是不可再分解的基因部分,并按照一定的顺序组成一个完整的个体.按照最基本协议单元进行基因编码的策略,能够保证在整个基于遗传算法的 Fuzzing 过程中,测试例中的基本协议格式不被破坏,否则将会被协议检查模块拒绝掉导致无法到达 IUT 的内部.

定义基因的数据结构如下:

```

struct gene {
    int position;
    int length;
    int type;
    char * data;
    struct gene * prev;
    struct gene * next;
};

```

每个个体实际上是染色体带有特征的实体,由

多个基因组成,由此定义个体的数据结构:

```
struct individual {
    int total_genes;
    struct gene * first_gene;
    struct individual * prev;
    struct individual * next;
};
```

图5说明一个拥有 m 个最基本协议单元的数据报的具体编码。

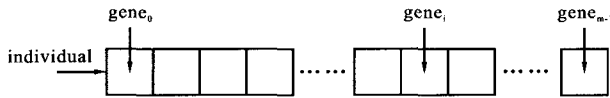


图5 基因编码

4.2.2 初始种群

通过网络嗅探获取与 IUT 正常交互的网络数据报,然后对这些数据报应用变异技术来生成半合法的测试例,由这些测试例组成初始种群。半合法测试例的合法部分保证了测试例能够通过协议模块的检查,从而避免在应用程序处理前被过滤掉,这些合法的部分主要包括测试例的几个协议头部字段,特别是长度和校验和字段。测试例中的非法部分是测试的关键,是测试例中真正触发漏洞的部分,非法部分是通过合法报文的某些字段进行替换来进行变异的,具体的替换方法是由被替换字段的语法和语义决定的,比如针对命令、SQL、二进制、字符型等等。同时也跟要根据不同的数据类型而进行相应的变异,比如针对整数型则可以用边界附近的几个值来替换,对于字符串可以采用字符串重复、字段分隔符、格式化字符串、目录遍历、命令注入等来进行替换。

4.2.3 适应度函数

适应度函数用来计算单个测试例的适应度,是遗传算法与具体应用进行交互的唯一接口。遗传算法一方面从外部应用接收用于计算适应度的数据,另一方面通过适应度的计算来指导下一代测试例的生成,适应度函数决定了整个进化的方向。适应度函数要充分表达遗传算法要解决的问题,从而能够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,那些能够触发更多代码覆盖率的测试例触发漏洞的可能性更大,而影响测试例触发漏洞几率大小的因素除了代码覆盖率以外还有其他因素,文中仅考虑测试例的代码覆盖率和多样性。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

样性最大的测试例。

适应度函数定义如下:

$$\text{Fitness} = \text{Coverage} + \text{New_coverage}$$

Coverage: 代码覆盖。文中使用的是源码行覆盖,表示测试例触发 IUT 所执行的源码行数。

New_coverage: 表示测试例触发的新的源码行数,既那些在当代之前没有被触发的源码行。该因素的引入可以在一定程度上防止进化过程中早熟现象的出现,同时可以增强遗传算法的全局搜索能力。

对当代测试例逐个按照适应度函数进行计算后,按照适应度由大到小进行排序,适应度大的测试例拥有更优秀的基因,将获得更多交配的机会,从而使得自身优秀的基因能够尽量多的体现在下一代中。记最优测试例为 p_{best} 。

4.2.4 选择算子

在进行完适应度计算和排序后进行选择操作,从排序后的种群 G 中进行父代个体的选择,用于生成新一代种群。

文中采用轮盘赌法来进行选择操作。令 f_i 表示第 i 个个体 p_i 的适应度,则种群适应度为 $\sum f_i$,其中 $f_i / \sum f_i$ 表示第 i 个个体 p_i 的适应度在整个种群累计适应度中所占的比率。对于种群中的 n 个个体,轮盘按照每个个体的适应度在整个种群总适应度中所占的比率 $f_i / \sum f_i$ 来进行划分,拥有较大适应度的个体将在轮盘上获得较大的划分,从而获得更大的选择机会。轮盘每转动一次则选择一个个体,轮盘随机的转动 n 次就获得了用于交叉操作的父代种群 G' ,其中那些拥有较大适应度的个体在父代种群中多次出现的概率将很大,而拥有较小适应度的个体很可能不会出现在父代种群中。

4.2.5 交叉算子

交叉算子是进化过程中生成新一代种群的重要手段,利用生物学中杂交的原理,通过将两个父个体的部分基因进行交换重组,从而得到两个新的个体。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等,以及各种改进的自适应的交叉方式。根据文中的测试目标及编码方式,采用单点交叉算子和保留最优个体策略,具体交叉算子为:

①从父代种群 G' 中依次选取两个父代个体 Parent_1 和 Parent_2 ;

②每个个体有 m 个基因,从 $[1, m-1]$ 区间中随

机选取一个整数 c 作为交叉点;

③将 $Parent_1$ 和 $Parent_2$ 的基因从随机交叉点 c 处进行互换,从而生成两个新的个体 $Child_1$ 和 $Child_2$,如图 6 所示;

④反复前三个步骤,直到所有父代个体都进行过交叉操作;

⑤从新一代种群 $G_{new}(Child_1, Child_2, \dots, Child_n)$ 中,随机删除一个个体 $Child_i$,同时将上一代 G 中的最优个体 p_{best} 添加到新种群 G_{new} 中.

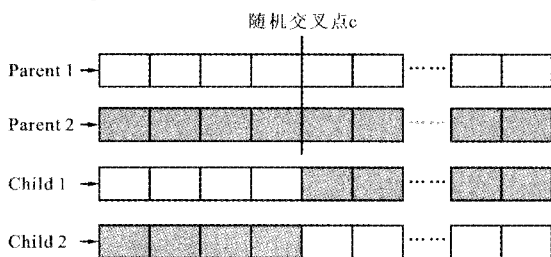


图 6 交叉算子

4.2.6 变异算子

变异算子利用生物学中个体的基因突变现象,通过对个体的部分基因进行随机改变,从而增强群体的多样性,避免早熟的发生.在变异算子中,变异概率不能过大,超过 0.5 遗传算法就会退化为随机搜索,太小则会减缓对新空间的搜索速度.对于需要进行变异操作的个体,随机地确定个体中需要变异的基因块,根据基因块的数据类型采用不同的数据变异策略进行变异操作.对于字符串类型的数据,则向其中插入一段特定的字符串;对于数字类型的数据,则对其中的某些 bit 进行位翻转操作.由于采用保留最优个体策略,最优个体 p_{best} 不进行变异操作.

5 代码覆盖

在整个 Fuzzing 系统框架中,代码覆盖是将遗传算法部分和 IUT 联系起来的唯一纽带.适应度函数通过对代码覆盖的计算来对每个测试例进行评价,从而能够进一步执行优胜劣汰的遗传操作. Purecov 是 IBM 公司的 Rational PurifyPlus 测试工具集的一个子模块,可以完成代码行和函数覆盖.相对于其他的代码覆盖工具, Purecov 的优势在于提供了多种覆盖方式和多种覆盖结果查看分析方式,特别是 Purecov 提供的 API 接口可以满足用户进行自定义应用的需求.在测试过程中,主要通过源码中适当的位置插入以下两个 API 函数来完成代码覆盖功能:

`purecov_set_filename(filename)` 用于设置代

码覆盖输出文件的名称;

`purecov_save_data()` 用于保存当前代码覆盖到指定的 filename 文件中.

完成 API 的插入后,使用如下命令进行源码编译: `% purecov gcc -g ntpd.c`

执行编译后的程序即可获得代码覆盖文件 `*.pcv`.

6 实验分析

针对基于遗传算法的 Fuzzing 框架 GAF 进行了初步的实现,并通过对网络时间协议(Network Time Protocol, NTP)的测试,验证了整个框架的有效性. NTP 是目前网络上普遍采用的时钟同步化的网络协议,基于客户机/服务器模式,通过 UDP 123 端口提供服务.测试代码是 ntp-4.2.0,共有 93444 行代码,能够被网络报文触发的代码大概占 35%,这是测试过程中所关心的代码部分.本次测试过程中, GAF 的参数设置:交叉概率为 0.9,变异概率为 0.05,初始种群个体数量为 100,个体基因个数为 14.

使用 Purecov 对 ntp-4.2.0 进行编译、配置后,运行并测试 NTP. 经过 500 代的运行测试后,群的代码覆盖率达到了 26%,整个测试过程的累积代码覆盖率是 32%.通过对代码覆盖率变化发现,该模型在拥有较好的局部搜索能力的同时,由于在适应度函数中加入了多样性因子,从而增强了全局搜索能力.但由于 NTP 的代码实现比较成熟,以及变异数据设计的不完善,未能触发远程漏洞,这需要对框架模型进一步改进.

7 结束语

在综合分析当前现有的 Fuzzing 技术基础上,文中提出了一种基于遗传算法生成测试例的 Fuzzing 框架模型,该模型主要针对网络服务程序进行测试.针对测试目标的特点,完成了遗传算法的编码、选择算子、交叉算子、变异算子以及适应度函数定义和设计,并对该框架进行了初步实现.通过实验分析发现,该模型能够克服在传统 Fuzzing 过程中的盲目性,能够较好的按适应度函数所指定的方向进行测试.另一方面,即使覆盖了 100% 的可触发代码,也不能保证能够触发漏洞或者 IUT 的安全性.为了进一步提高测试的效率和触发漏洞的概率,下一步的改进主要集中在测试数据的变异技术和适应度函数两个方面.

(下转第 139 页)

5 结束语

文中提出的改进算法是在经典 SOC 算法的基础上增加了两步有效的处理. 根据实验结果可知,文中算法可以全面提升 SOC 算法对矢量量化索引的压缩性能,特别是在使用较大码书时提升的效果更明显. 要得到越清晰的图像就需要越大的码书. 但是经典算法 SOC 在小码书时压缩效果明显,在大码书时效率迅速下降. 文中算法在 128 码书时可以对 SOC 算法的效率提升约 3%,而在使用 1024 码书时效率提升约 7%,这表明,文中算法具有实际应用的意义. 同时,文中算法思路简单、计算复杂度低、易于硬件实现.

参考文献:

- [1] 孙圣和,陆哲明. 矢量量化技术及应用[M]. 北京:科学出版社,2002.
- [2] 张霞,万国金,吴建华,邹道文. 矢量量化在图像压缩编码中的应用进展[J]. 数据采集与处理,1999,19(1).
- [3] 王卫,蔡德钧,万发贵. 用于图像编码的相关矢量量化研究[N]. 电子学报,1995,23(4):30—34.
- [4] 蔡光跃,董恩清. 一种改进的基于 Hadamard 变换的快速码字搜索算法[J]. 微电子学与计算机,2007,24(2):154—156.
- [5] 乔阳,潘志斌,乔瑞萍,等. 基于 Hadamard 变换和自适应顺序搜索的码字快速搜索算法[J]. 微电子学与计算机,2009,26(1):118—121.
- [6] 谢敏,乔瑞萍,潘志斌,等. 一种改进的绝对误差不等式删除算法在 DM642 上的实现[J]. 微电子学与计算机,2010,27(4):182—185.
- [7] Hsieh ChaurHeh, Tsai JyiChang. Lossless compression of VQ index with search—order coding[J]. IEEE Transactions on Image Processing, 1996, 5(11):1579—1582.
- [8] Shie ShihChieh, Chen LongTai. Image compression based on side—match VQ and SOC[C]// Digital Image Computing: Techniques and Applications, 2009: 369—373.
- [9] Ho ChunYang, Hsieh ChaurHeh, Chao ChungWoei. Modified search order coding for vector quantization indexes[J]. Tamkang Journal of Science and Engineering, 1999, 2(3):143—148.
- [10] Chen PeiYin, Chen RenDer. An index coding algorithm for image vector quantization[J]. IEEE Transactions on Consumer Electronics, 2003, 49(4):1513—1520.
- [11] Somasundaram K, Domnic S. Adaptive index coding scheme for VQ indices compression[C]// Proceedings of the International Conference on Computing: Theory and Applications (ICCTA '07). Kolkata, India: IEEE, 2007:639—644.

作者简介:

张冰清 女,(1988—),本科. 研究方向为图像编码.

潘志斌 男,(1963—),教授. 研究方向为图像编码、图像检索、视频处理.

高风娟 女,(1986—),硕士. 研究方向为图像编码.

(上接第 134 页)

参考文献:

- [1] Michael Sutton, Adam Greene, Pedram Amini. Fuzzing brute force vulnerability discovery[M]. 北京:机械工业出版社,2009.
- [2] Miller B P, Fredrikson L, So B. An empirical study of the reliability of unix utilities[J]. Communication of the ACM, 1990, 33(12):32—44.
- [3] 刘晓东,孙圣和. 基于遗传算法的自适应测试生成[J]. 微电子学与计算机, 2002(3):14—16.
- [4] 吕军,冯博琴,李波. 免疫遗传算法及其应用研究[J]. 微电子学与计算机,2005(6):221—224.
- [5] Goldberg D E. Genetic algorithms in search, optimization and machine learning[M]. Boston, MA, USA: Addison—Wesley Pub Co, 1989.
- [6] Mills D. Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI. IETF Internet RFC, RFC 2030[R]. USA:University of Delaware, 1996.

作者简介:

王丹 男,(1980—),博士研究生. 研究方向为软件测试、网络信息安全.

高丰 男,(1972—),高工,博士. 研究方向为操作系统、网络信息安全.

朱鲁华 男,(1964—),教授,博士生导师. 研究方向为操作系统、网络信息安全.