EWFT:基于程序执行过程的白盒测试工具

王 颖[1],谷利泽[1],杨义先[1],董宇欣[2]
(1.北京邮电大学信息安全中心,北京 100876;2.哈尔滨工程大学,黑龙江哈尔滨 150001)

摘 要: 应用动态测试技术检测二进制程序的脆弱性是当前漏洞挖掘领域的研究热点.本文基于动态符号执行和污点分析等动态分析技术,提出了程序路径空间的符号模型的构建方法,设计了 PWA(Path Weight Analysis)覆盖测试算法,实现了 EWFT(Execution-based Whitebox Fuzzing Tool)原型工具.实验测试结果表明,EWFT 提高了程序执行空间的测试覆盖率和路径测试深度,相比国际上同类测试工具,能够更加有效地检测出不同软件中存在的多种类型的程序漏洞.

关键词: 动态测试;软件脆弱性分析;测试用例生成;压缩存储
中图分类号: TP309.5 **文献标识码:** A **文章编号:** 0372-2112 (2014)10-2016-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2014.10.023

EWFT:Execution-based Whitebox Fuzzing for Executables

WANG Ying[1], GU Li-ze[1], YANG Yi-xian[1], DONG Yu-xin[2]
(1. *Information Security Center, School of Computer, Beijing University of Posts and Telecommunications, Beijing* 100876, *China*;
2. *Harbin Engineering University, Harbin* 150001, *China*)

Abstract: The dynamic testing for automaticlly identifing security vulnerabilities in binary executables has received increasingly interest in recent years. In this paper, we present a new automated whitebox fuzzing tool EWFT(Execution-based Whitebox Fuzzing Tool), which implements dynamic symbolic execution and taint tracing techniques during program execution. Our contributions are:1)we propose a ROBDD(Reduced Ordered Binary Decision Diagram)-based approach to analyse execution process,2)we introduce a new path weight analysis algorithm(PWA)for searching path space and automating test data generation, and 3)we build a prototype tool that automatically finds software vulnerabilities. Results of our experiments show that execution-based whitebox fuzzing is powerful to identify variety of security vulnerabilities in real applications. Compared to the related work in the research area, it explored deeper program paths on the average, and achieved higher structural coverage.

Key words: dynamic test; software vulnerability analysis; test generation; data compression

1 引言

当今国际信息安全领域内,基于动态分析方式的技术在软件安全性分析中占有越来越重要的作用.在此趋势下,动态符号执行和动态污点分析[2,3]技术在国际上得到了迅速的发展,它们与模糊测试技术相结合能够有效地克服软件系统日益增加的分析难度.这其中包括结合模糊测试和动态符号执行技术的测试方法[4,5]、结合模糊测试和动态污点分析的测试方法[6]、以及综合应用模糊测试、动态符号执行和动态污点分析的测试方法[7-9]等.

2 本文创新点与 EWFT 结构

综上所述,以动态测试技术检测二进制程序的脆弱性是当前国际漏洞挖掘领域的研究热点,本文提出两个方面的创新点:

①为准确地分析程序执行状态,实现了程序路径约束条件的符号模型的构建方法.根据程序执行路径和约束条件之间的映射关系,以简化有序二元决策图(Reduced Ordered Binary Decision Diagram, ROBDD)方法[10,11]构建程序执行路径约束条件的符号模型,对路径之间关系计算提供支持,克服了程序执行过程难以准确分析的问题,并具有较低的空间复杂度和时间复杂度.

②对于程序执行路径空间的搜索问题,实现了一种有效的 PWA(Path Weight Analysis)覆盖测试算法.主要包含基于度量权重的程序执行路径的选择算法和测试用例生成算法两个部分的内容.该算法能够有效地解决程序执行路径空间的覆盖率和路径测试深度难以提高、以及测试过程中路径状态空间爆炸等问题,具有效率高、实用性强等优点.

收稿日期:2013-06-30;修回日期:2013-12-17;责任编辑:郭游
基金项目:国家自然科学基金(No.61003285,No.61121061);中央高校基本科研业务费专项资金(No.2012RC0218,No.2012RC0219,No.2013RC0311)

在开源软件 Valgrind[12,13]和 BuDDy(Version 2.2)开源代码包[14]的基础上,开发出实现上述创新点的程序脆弱性测试工具——EWFT(Execution-based Whitebox Fuzzing Tool),综合应用动态符号执行、污点分析、以及遗传变异算法等技术直接面向二进制程序测试,系统结构如图 1 所示:

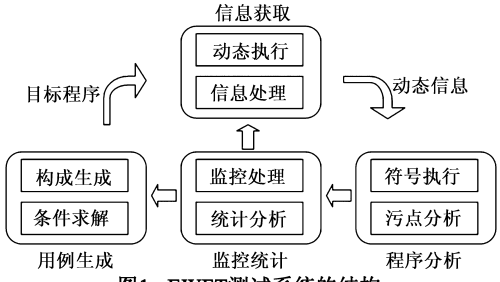

图1　EWFT测试系统的结构

EWFT 主要包括四个组成部分:信息获取模块、程序分析模块、测试用例生成模块、以及监控和统计分析模块.信息获取模块包含动态执行和信息处理功能组件,其中动态执行组件的功能为动态执行测试目标程序,并在程序运行时以 Valgrind 动态分析框架进行动态指令插桩,对程序执行过程进行相应动态分析,得到相应的执行信息.在信息处理组件中存储程序指令的执行轨迹,包括指令的地址和内容、内存的地址和内容、以及寄存器与其内容等,并对其进行预处理,根据预处理信息可以跳过对相同指令单元的重复分析,有效地提高效率.程序分析模块包含动态符号执行和动态污点分析功能组件,其中动态符号执行组件的功能为获取程序执行路径约束条件等动态执行信息,构建程序执行路径约束条件之间的动态结构.动态污点分析组件的功能是构建程序执行过程的数据流图、控制流图、以及变量之间的依赖关系,并对相关敏感操作进行脆弱性分析.测试用例生成模块包括条件求解和构造生成功能组件,其中条件求解功能组件根据程序执行路径的约束条件集合求解得到新的可执行路径的测试用例,构造生成组件基于遗传变异算法构造新的测试用例.监控和统计分析模块包括监控处理和统计分析功能组件,其中监控处理功能组件对其它模块的执行状态进行监控,并控制各部分之间的协同操作.统计分析功能组件主要功能为计算程序执行路径权重,并对程序执行路径进行分类和统计.下面分别对本文提出的两个创新点进行具体地阐述.

3　符号模型的构建方法

3.1　数据的动态压缩存储

二元决策图方法(Binary Decision Diagram-BDD)[15,16]可以有效地表示和存储数据结构.目前,在有序二元决策图方法[17-19]的研究基础上,BDD 方法进一步发展为简化有序二元决策图(ROBDD)方法[10,11].本文应用 BuDDy(Version 2.2)开源代码包[14]实现的 ROBDD 方法对程序动态执行信息、以及动态污点分析和符号执行相应处理和分析所生成的数据进行高效地压缩存储.例如对于程序执行轨迹 $R_1=\{0,1,2,3,6,7,9,10,11\}$,其对应的 ROBDD 向量表示如表 1 所示:

表 1　R_1 的向量表示

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
6	0	1	1	0
7	0	1	1	1
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1

EWFT 在程序测试过程中,应用 ROBDD 方法对数据进行相应地压缩存储,不仅可以优化数据压缩存储的表示形式,并且能够使数据结构集合(例如程序执行路径)共享相同数据.相比 BDD 方法,数据压缩存储的复杂度以指数级别降低.

3.2　符号模型的建立

EWFT 对程序的可执行路径进行遍历测试时,以程序指令基本块为分析单位,应用 ROBDD 方法有效地构建和存储其约束条件所对应符号模型的关系结构,如图 2 所示.

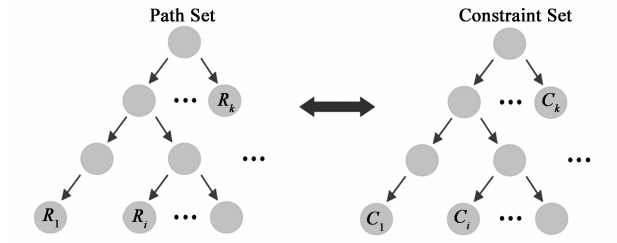

图2　程序执行路径与符号模型之间的映射关系

该过程以程序指令基本块的约束条件为单位结点,设 P 为程序执行时获取的路径约束条件的结点集合,L 为已经存储的程序执行路径的约束条件模型的结点集合,a 和 b 分别为程序不同执行过程所对应的一系列约束条件的结点序列.若存在关系:$a\in P$,$b\in L$,$n_i\in a$,$n_i\notin b$,则路径约束条件模型的构建和存储算法如下:

$$\begin{aligned}&if\ a\neq b\ or\ a\supset b\\&then\ Add\ symbolic(n_i)\ model(L)\end{aligned}\tag{1}$$

即对 a 中新结点进行符号化,并存储到 L 的存储结构中相应位置,如图 3 所示.

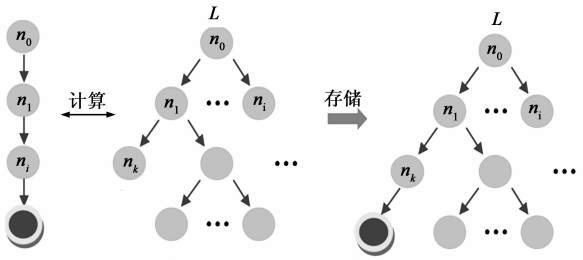

图3 a中新结点n_x存储到L中对应结构

为提高存储效率,在路径约束条件的符号建模和存储过程中,EWFT 为测试进程中各线程单独分配缓存,并在各次测试结束后统一进行相应的数据存储.

4 PWA 测试算法

为解决程序执行路径空间搜索过程中路径爆炸和覆盖率问题,*EWFT* 应用 *PWA*(*Path Weight Analysis*)算法对目标程序的执行路径空间进行覆盖测试,主要包括程序执行路径选择算法和测试用例生成算法两部分内容.

4.1 程序执行路径选择算法

EWFT 在目标程序执行过程中对其进行测试,其初始输入为依据目标程序特性和功能而进行构造的测试用例集合,然后根据程序执行路径空间的测试方法递归生成新一代的测试用例集合,使目标程序的执行不同的路径,达到对其执行路径空间的覆盖测试.因为程序执行路径与约束条件之间为同构映射,程序执行路径空间的测试方法基于程序路径约束条件的关系结构实现对程序执行路径的相关计算,算法中以基本块为单位结点,并在程序执行路径约束条件的存储过程中计算相关信息,其中程序执行路径长度保存在相应的数据结构中.

程序执行路径选择算法以路径相似度和路径距离的计算值对已执行路径排序,生成新的测试用例集合,对程序的可执行路径空间进行覆盖测试.算法如下所示:

程序路径选择算法首先在动态构建的程序执行路径存储结构中选择长度最大的未标记路径 r,对其进行标记.根据程序执行路径约束条件的存储结构,计算集合 P 中程序执行路径与 r 之间共同的路径前缀结点序列 Ancesor,以及相同结点序列 Identical,计算得到 P 中程序执行路径与 r 之间的相似度和距离,选择 PathSum * threshold 条具有最大相似度和 PathSum * (1-threshold) 条具有最大距离的路径分别存入候选路径队列 candidate1 和 candidate2,作为构造下一代测试用例的生成算法的输入,实现对测试目标程序的可执行路径空间进行覆盖搜索,PathRequest 为动态设定的常量.其中路径的相似度算法和距离算法分别如公式(2)和(3)所示:

```
int PathSelect(PathSetP, PathQueue * candidate1, PathQueue * candidate2){
    Path * temp = mull;
    PathQueue * equality = mull;
    SelectLongPath(r);
    if((PathSum = GetSum(P)) > PathRequest)
      PathSum = PathRequest;
    while((temp = GetPath(P))! = null)
      GetAncestor(r, temp, ancestor);
    if((equality = SortSimilarity(P, candidate1; threshold))! = null)
      SortMaxBlock(equality, candidate1, threshold);
    if((equality = SortDistance(P, candidate2, threshold))@ = null)
      SorMiniBlock(equality, candidate2, threshold);
    return PathSum;
}
```

图 4 程序执行路径选择算法

$$Similarity(Path_1, Path_2) = \left\| \frac{Ancestor(Path_1, Path_2)}{Path_1} \right\| \tag{2}$$

$$Distance(Path_1, Path_2) = \| Path_1 + Path_2 - \bigcup_{i=1}^{n} 2 * Identical_i(Path_1, Path_2) \| \tag{3}$$

公式(2)和(3)中,路径结点以基本块为单位,$Ancesor_i(Path_1, Path_2)$为程序执行路径 $Path_1$ 和 $Path_2$ 共有的路径前缀结点序列,$Identical_i(Path_1, Path_2)$为 $Path_1$ 和 $Path_2$ 相同的路径结点序列.例如,对于图 6 中程序执行路径 R_j 和 R_k,其相应的计算值如下:$Ancesor(R_j, R_k) = \{n_1, \cdots, n_i\}$,$Identical(R_j, R_k) = \{(n_1, \cdots, n_i), (n_x)\}$,路径相似度 $Similarity(R_j, R_k) = \frac{n_i}{(n_i + 3)}$,路径距离 $Distance(R_j, R_k) = 3$.

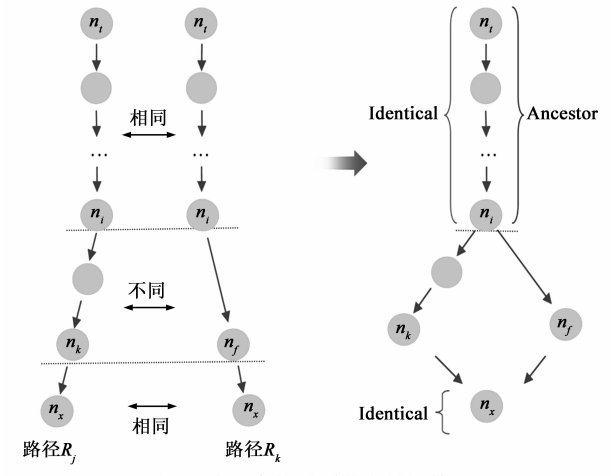

图5 R_j与R_k之间关系的定量计算

根据上述演算结果,算法过程通过计算相应权重

对集合 P 中路径进行排序,选择并存储候选路径,主要步骤如下:

①SortSimilarity 函数中以 Similarity 的值为权重,对集合 P 中未标记的程序执行路径降序排列,并选择前面 PathSum * threshold 条未标记路径为初选序列. 如果存在关系: PathSum * threshold > PathSum * SimiMax, 或 PathSum * threshold < PathSum * SimiMin, 则执行⑤;若 equality = = null, 依次存储序列前面 PathSum * threshold 条路径到队列 candidate1 中,执行③. 否则,执行②.

②对于该序列中最后一条路径 l_1, 如果序列中存在 n_1 条路径,且 P 中存在 n_2 条路径,它们与 l_1 的 Ancestor长度相等,则依次存储序列中前面 PathSum * threshold - n_1 条路径到队列 candidate1 中,并存储该 $n_1 + n_2$ 条路径到队列 equality. 分别从 Ancestor 序列之后下一个结点开始,通过 SortMaxBlock 函数计算 equality 中各路径包含 r 中的基本块结点的比率,实现有效搜索路径的二次排序和选择,计算公式如下:

$$\mathrm{BlockSimilarity}(\mathrm{Path}_1,\mathrm{Path}_2)=\left\|\frac{\mathrm{Identical}_i(\mathrm{Path}_1,\mathrm{Path}_2)-\mathrm{Ancestor}(\mathrm{Path}_1,\mathrm{Path}_2)}{\mathrm{Path}_1-\mathrm{Ancestor}(\mathrm{Path}_1,\mathrm{Path}_2)}\right\| \tag{4}$$

SortMaxBlock 函数以路径相同基本块的比率为权重排序和选择 equality 中 n_1 条路径存储到队列 candidate1 中.

③SortDistance 函数中以路径距离 Disdance 的计算值为权重,对集合 P 中未标记的程序执行路径降序排列,并选择前面 PathSum * (1-threshold)条未标记路径为初选序列. 如果 equality = = null, 依次存储路径到队列 candidate2 中,执行步骤⑥. 否则,执行步骤④.

④对于该序列中最后一条路径 l_2, 如果序列中存在 m_1 条路径,且 P 中存在 m_2 条路径,它们与 l_2 的距离长度相等,则依次存储序列中前面 PathSum * (1 - threshold) - m_1 条路径到候选队列 candidate2 中,并存储该 $m_1 + m_2$ 条路径到队列 equality. 通过 SortMiniBlock 函数对 equality 中各路径以相似度为权重进行升序排列,选择其中相似度最小的 m_1 条路径存储到队列 candidate2 中.

⑤调整 threshold 阈值,调整方向与 PathSum * threshold 的值成反比. 执行①.

⑥执行测试用例生成算法.

上述计算步骤中,算法遍历程序执行路径空间进行覆盖测试. 其中变量 threshold 是可以动态调整的阈值,取值范围为: SimiMin ≤ threshold ≤ SimiMax. 全局变量 SimiMax 和 SimiMin 根据程序的特性和功能状态设定. 算法通过调用 CheckLoop 函数单独处理执行路径中循环结构的指令序列. 在路径存储过程中,具有相同指令序列的循环结构被压缩存储,其指令序列的长度只计算一次. Checkloop 函数中结合 Control-Flow Directed Search 算法[20,21]选择分支路径. 而对于控制依赖结构的路径约束条件,求解器通常无法直接求解, Checkloop 函数中结合"Fitnex strategy"算法[22]选择有效路径.

4.2　测试用例生成算法

以 4.1 节选择的路径作为输入, EWFT 系统用例生成模块对程序执行路径的约束条件求解,得到新的可执行路径的测试用例. 同时,根据程序执行路径的存储结构,应用之前工作中提出的遗传变异算法[23]生成新的 Fuzzing 测试用例,提高路径空间的测试覆盖率和平均测试深度. 算法如下:

```
CaseSet TestGreneration(PathQueue * candidate1, PathQueue * candidate2){
    CaseSet * TestCase = null;
    Case * solution = null;
    PathQuerer * ShortPath = null;
    Path * temp = null;
    for(i = 0; i < = candidate1. amount; i + + ){
      temp = GetPath(candidate1);
      for(j = temp. baselocation + 1; j < = temp. count; j + + )
        if((solution = NodeNegationSolver(temp, j))! = null)
          AddSolution(TestCase, solution);
    }
    for(i = 0; i < = candidate2. amout, i + + ){
      temp = GetPath(candidate2);
      for(j = temp. baselocation + 1; j < = temp. count, j + + )
        if((solution = NodeNegationSolve(temp, j))! = null)
          Addsolution(TestGase, solution);
    }
    if(TestCase. count( = CaseRequeset)
      Ascend(TestCase. count / CaseRequest);
    if((SortLength(P, ShortPath, CaseRatent))! = null)
      MutateGeneration(TestCase, ShortPath);
    returnTestCase;
    }
```

图 6　测试用例生成算法

上述算法的主要步骤如下:

①依次取出队列 candidate1 中程序执行路径,前向遍历路径查找到 baselocation 位置,对于其所有后续结点,分别执行以下操作:对路径结点相应的约束条件执行取反操作,保持该结点在路径中位置不变,并求解路径对应的约束条件序列. 如果有解,则存储对应的测试用例信息{case, baselocation}到集合 TestCase 中,其中 case 为求解得到的实例, baselocation 为该节点位置.

②依次取出队列 candidate2 中程序执行路径,执行与步骤①相同的计算过程,把得到的测试用例信息存储到集合 TestCase 中.

③计算 TestCase 集合中用例数量,如果数量小于设定阈值 CaseRequest,则计算其与 CaseRequest 之间比率,应用 Ascend 函数实现的启发算法对 CaseParent 动态调

节.

④根据程序执行路径的存储结构,对路径长度按升序排列,选择 CaseParent 条未标记的最小长度的路径,并存储对应的输入用例作为遗传父代集合.

⑤以遗传父代集合为输入,根据遗传变异算法构造新的测试用例,并存储生成的测试用例信息到集合 TestCase 中.

4.3 算法优化与分析

4.3.1 算法优化

为降低 PWA 算法复杂度,提高 EWFT 的测试效率,进行如下优化:

①建立路径信息链表 InfoList,以路径长度的降序排列方式存储未标记的程序执行路径的相关信息.

②在构建和存储路径约束条件的符号模型过程中,对结点进行符号赋值,并把路径的符号序列、路径长度、存储索引等信息存入 InfoList 链表中相应结点.

③计算程序执行路径之间的 Ancesor 序列时,化简该序列,并存储相应化简结果.

④计算路径权重时,顺序遍历 InfoList 中前面 Volume 个结点,分别计算对应路径的 Similarity、Distance、以及 BlockSimilarity 权值,其中 Volume >> PathRequest.

⑤在选择路径存储到队列 candidate1 和 candidate2 过程中,标记已选择路径,并删除其在链表 InfoList 中对应结点.

下面分析 PWA 算法的复杂度:①计算 Similarity 和 Disdance 等路径权重值复杂度为:$O(N) * \text{Volume} = O(N)$;②候选队列 candidate1 和 candidate2 中程序执行路径的选择算法的复杂度为 PathRequest * O(N).因为这里 N = Volume,且 PathRequest 为常量,所以复杂度为 $O(1)$.③以候选队列中程序路径生成测试用例过程的复杂度为:$O(N) * (\text{PathSum} * \text{threshold}) + O(N) * (\text{PathSum} * (1 - \text{threshold})) = O(N) * \text{PathSum} = O(N)$.

综上分析可知,PWA 算法的复杂度为:$O(N) * O(1) * O(N) = O(N^2)$.

4.3.2 算法比较分析

目前国际上相关研究工作中 Godefroid 等学者在 SAGE 测试工具中实现的测试算法[4,24]处于领先水平.与传统算法相比,SAGE 算法具有更高的程序执行路径空间覆盖率,并能够有效地检测出应用程序中漏洞,具有效率高、实用性强等优点.该算法的复杂度为 $O(N^2)$,主要内容为:①构造初始测试用例,以其作为输入运行测试目标程序.②获取程序执行路径,抽取各路径的约束条件序列,并依次存储到队列中.③从队列中取出一条序列,分别取反序列中未标记的约束条件,如果新构成的约束条件序列有解,则存储相应实例到测试用例集合.④以集合中测试用例作为输入,运行被测试程序.如果能够获取新的程序执行路径,执行步骤②.否则,算法结束.

下面对 SAGE 测试算法与 PWA 算法的有效性进行综合的对比分析:

⑴被测试程序执行的充分程度.SAGE 算法中,被测试程序执行的充分程度依赖于初始测试用例的有效性和全面性.但在实际测试过程中,符合该要求的初始测试用例通常难以构造.因此,应用该算法无法有效保障被测试程序得到充分执行.

PWA 算法中应用四个方面技术,能够保证被测试程序的充分执行:①程序执行路径选择算法中以相似度 Similarity 为权重选择候选路径,生成的测试用例能够保障程序复杂结构的指令代码得到充分执行;②程序执行路径选择算法中以距离 Disdance 的计算值为权重选择候选路径,能够有效地避免测试搜索范围的局部聚集性问题;③算法在测试过程中动态分析并选择已执行的最长程序执行路径,能够有效地提高程序执行路径空间的测试覆盖率和路径测试深度;④应用遗传变异算法构造新的测试用例,可以扩展程序执行路径的搜索空间,发现新的程序执行路径.

⑵符号模型的构建和求解效率.SAGE 算法采用了基于代(generation)的测试用例生成、实例替换(concretization)、冗余条件消除(unrelated constraint elimination)、以及取反上限约束(flip count limit)等技术,具有较高的工作效率.

在 SAGE 算法所采用技术的基础上,PWA 算法还采用了以下三类优化技术:①利用 ROBDD 方法有效地构建和存储符号模型;②选择与当前最长路径具有最大相似度和距离的多条路径构成候选队列,并求解生成可以执行不同程序轨迹的测试用例集合;③对程序执行路径的 Ancesor 序列的优化存储技术.上述优化技术有效地降低了符号模型的构建和求解过程的时间和空间复杂度.并且生成的测试用例,能够有效地提高程序执行路径的测试覆盖率和平均测试深度.

根据以上综合分析可知:PWA 算法能够更加有效地提高测试过程的整体效率.

5 实验验证

实验选取的基准测试程序如下:

表 2 中"Xpdf"是一款 PDF 文件的阅读器,"Transmission"是一款下载工具,Dia 为绘图工具.基准测试程序都是开源的,且源代码中包含已知程序漏洞.这样通过对它们进行安全测试,就能够准确验证和评估 EWFT 检测程序漏洞的有效性和实用性.

实验设备硬件配置为 Intel Core 2 Duo(CPU 2.4GHz)、2GB 内存和 4MB 二级缓存,系统环境平台为

X86/Linux (kernel 2.6).

表 2　基准测试程序

基准程序	程序描述
bzip2	数据压缩工具
gzip	文件压缩程序
Xpdf	PDF 阅读器
Transmission	BitTorrent 客户端
Kaffeine	媒体播放器
Dia	画图工具

5.1　程序漏洞检测功能验证

根据 MITRE 公司和美国系统网络安全协会(SANS Institute)联合发布的近几年最新的"CWE/SANS TOP 25 Most Dangerous Software Errors"[25]和"CWE/SANS Top 25 – On the Cusp"[26]的统计报告可知,2009 ~ 2011 年期间"Risky Resource Management"这一类程序漏洞具有较大的危害.其中的{CWE – 22, CWE – 119, CWE – 120, CWE – 129, CWE – 131, CWE – 134, CWE – 190, CWE – 476}等程序代码缺陷类型重复入选,并且程序缺陷代码相对复杂、具有较大的危害.

通过 CVE、NVD 中程序漏洞数据可知,表 2 中给出的基准测试程序中包括相关的程序漏洞和代码缺陷类型.对它们的有效检测,能够实际地验证程序安全测试工具对程序漏洞的检测能力.实验中以 EWFT 原型工具对基准测试程序进行检测,检测的结果如表 3 所示.

表 3　EWFT 检测结果

基准程序	CWEID	漏洞编号	漏洞描述
bzip2	CWE-190	CVE-2010-0405	整数溢出
	CWE-125	CVE-2008-1372	缓冲区越界读
gzip	CWE-476	CVE-2006-4334	引用空指针
	CWE-129	CVE-2006-4335	栈缓冲区越界
	CWE-190	CVE-2006-4336	缓冲区下溢
	CWE-787	CVE-2006-4337	缓冲区越界写
Xpdf	CWE-190	CVE-2009-3609	整数溢出
	CWE-131	CVE-2009-3608	堆溢出
		CVE-2009-3606	
	CWE-789	CVE-2010-3604	受控内存分配
		CVE-2009-1188	
Tiansmission	SWE-22	CVE-2010-0012	重写任意文件
Kaffeine	CWE-120	CVE-2006-0051	缓冲区溢出
Dia	CWE-134	CVE-2006-2480	格式化字符串漏洞
		CVE-2006-2453	

表 3 中第 2、3 列分别为检测出的代码缺陷类型和相应具体的程序漏洞.其中,程序缺陷类型"越界读(CWE – 125)"、"越界写(CWE – 787)"、以及"受控内存分配(CWE – 789)"都与上述重点检测的程序代码缺陷类型密切相关,它们都是 CWE – 20 类型的子类,而 CWE – 20 是 CWE – 120、CWE – 129、CWE – 134、以及 CWE – 190 等类型的父类,可知这些程序代码缺陷类型具有相近的性质.根据实验结果得到,EWFT 能够准确地检测出基准测试程序中包含的不同类型的程序漏洞,验证了该系统检测程序漏洞的有效性和实用性.

5.2　测试算法有效性验证

目前,国际上验证一个测试算法的有效性,主要包括路径的覆盖率和平均测试深度这两方面的评估指标.为验证 PWA 算法的有效性,实验中同时以 SAGE 算法进行测试,并对两种算法得到的实验数据进行对比分析,从而验证 EWFT 的有效性.实验中根据特性与功能,分别为每个基准测试程序构造、或选取 10 个不同的初始输入文件,以提高执行路径空间的遍历程度.测试结束后,对所有测试数据进行统计分析,计算相应的平均值.根据这样分析的结果,能够对算法的有效性进行较为全面和充分的评估.

二进制程序的测试覆盖率通常以分支覆盖(Branch Coverage)测试指标评估.应用 PWA 算法和 SAGE 算法进行测试的实验分析结果如下图 7 所示.其中,横轴是应用 PWA 测试时,以遗传变异算法构造的测试用例在其总体测试用例中所占比值,纵轴为 PWA 算法与 SAGE 算法的分支覆盖率之间的差值.根据测试结果可知,PWA 算法的测试覆盖率高于 SAGE 算法的覆盖率.并且随着其遗传变异算法生成的测试用例所占比值的增大,两种算法实现的分支覆盖率的差值不断增大.

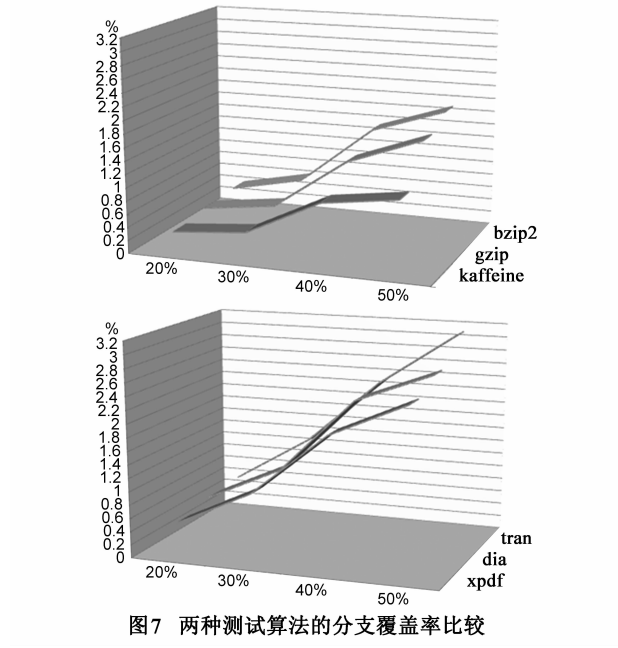

图7　两种测试算法的分支覆盖率比较

对二进制程序测试有效性的另一主要评估指标是路径平均测试深度.实验中分别应用两种算法对基准测试程序进行检测,并获取测试过程中的路径长度.对获得的实验数据进行分析,结果如下图 8 所示.

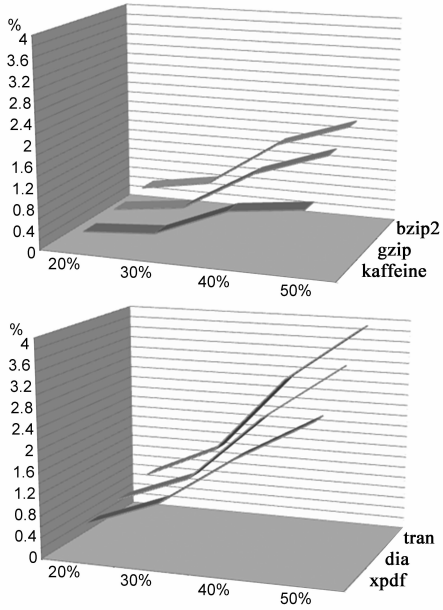

图8 两种测试算法的路径测试深度比较

图 8 中,横轴是应用 PWA 测试时,以遗传变异算法构造的测试用例在其总体测试用例中所占比值,纵轴为 PWA 算法计算得到的 l_1 与 SAGE 算法计算得到的 l_2 之间的差值:$L = l_1 - l_2$.其中 $l = \frac{S_l}{S}$,S_l 是大于平均路径长度的路径数量,S 是获得的所有测试路径数量.以 L 值进行测试路径深度分析,不仅可以从整体上评估较深测试路径的分布情况,而且能够有效地比较两种算法的测试效果.根据实验分析结果可知,PWA 算法能够测试更多数量较深程序路径.基准测试序列{kaffeine, gzip, bzip2, xpdf, dia, transmission}中,程序结构的复杂度和容积依次递增.根据实验结果分析得到,随着测试程序结构复杂度和容积的增大,L 值相应增加.这一实验结果验证了 PWA 算法的实用性和有效性.

6 结论

根据实验结果可以得到以下结论:① EWFT 构建的符号模型可以完整地复现路径执行的约束条件,准确地描述程序的执行路径空间,为污点分析和基于程序路径的测试数据生成等技术提供了充分的信息和数据;②基于 PWA 算法定量计算得到的路径相似度和距离等特征度量值,可以生成更加有效的测试用例,有效提高了程序执行路径空间的测试覆盖率和路径测试深度;③ EWFT 能够有效地分析软件脆弱性,在程序漏洞检测领域具有较强的实用性.

在现代网络攻防对抗中,应用 EWFT 分析程序的脆弱性,能够有效地增强系统的安全性与网络防范能力.例如在僵尸网络反制技术研究中,通过检测网络组成软件中存在的漏洞、或分析和利用僵尸程序中存在的各类代码缺陷能够进行有效的攻防对抗.

参考文献

[1] Cadar C, Godefroid P, et al. Symbolic execution for software testing in practice: preliminary assessment[A]. Proceedings of the 2011 33rd International Conference on Software Engineering[C]. New York: ACM, 2011. 1066 – 1071.

[2] Jim Chow, Ben Pfaff, et al. Understanding data lifetime via whole system simulation[A]. Proceedings of the 13th conference on USENIX Security Symposium [C]. California: USENIX Association Berkeley, 2004. 22 – 22.

[3] James Clause, Wanchun Li, Alessandro Orso. Dytan: a generic dynamic taint analysis framework[A]. Proceedings of the 2007 international symposium on Software testing and analysis[C]. New York: ACM, 2007. 196 – 206.

[4] Patrice Godefroid, Michael Levin, and David Molnar. Automated whitebox fuzz testing[A]. Proceedings of the Network and Distributed System Security Symposium[C]. California: Internet Society, 2008. 151 – 166.

[5] Papadakis M, Malevris N. Automatic mutation test case generation via dynamic symbolic execution[A]. Proceedings of the 21st International Symposium on Software Reliability Engineering[C]. Washington: IEEE Computer Society, 2010. 121 – 130.

[6] Rawat S, Mounier L. Offset-aware mutation based fuzzing for buffer overflow vulnerabilities: few preliminary results[A]. Proceeedings of the Fourth International Conference on Software Testing, Verification and Validation Workshops[C]. Washington: IEEE Computer Society, 2011. 531 – 533.

[7] Ganesh V, Leek T, Rinard M. Taint-based directed whitebox fuzzing[A]. Proceedings of the IEEE 31st International Conference on Software Engineering[C]. Washington: IEEE Computer Society, 2009. 474 – 484.

[8] Tielei Wang, Tao Wei, Guofei Gu, Wei Zou. Checksum-aware fuzzing combined with dynamic taint analysis and symbolic execution[J]. ACM Transactions on Information and System Security, 2011, 14(2): 1 – 28.

[9] Tielei Wang, Tao Wei, Guofei Gu, Wei Zou. Taintscope: a checksum-aware directed fuzzing tool for automatic software vulnerability detection[A]. 2010 IEEE Symposium on Security and Privacy[C]. Washington: IEEE Computer Society, 2010. 497 – 512.

[10] Shinichi Minato. Zero-suppressed bdds and their applications [J]. International Journal on Software Tools for Technology

Transfer,2001,3(2):156 – 170.
[11] Towhidi F,Lashkari A H,Hosseini R S. Binary decision diagram (BDD)[A]. International Conference on Future Computer and Communication[C]. Washington: IEEE Computer Society,2009.496 – 499.
[12] 网址[OL]:http://valgrind.org/
[13] Nicholas Nethercote,Julian Seward. Valgrind:a framework for heavyweight dynamic binary instrumentation[J]. ACM SIGPLAN Notices,2007,42(6):89 – 100.
[14] JØrn Lind-Nielsen. BuDDy:Binary Decision Diagram Package (Version 2.2)[OL]. 网址:http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.itu.dk/research/buddy/index.html
[15] S Chakravarty. A characterization of binary decision diagrams. IEEE Transactions on Computers,1993,42(2):129 – 137.
[16] Randal E Bryant. Binary decision diagrams and beyond:enabling technologies for formalverification[A]. Proceedings of the 1995 IEEE/ACM International Conference on Computer-aided design[C]. Washington:IEEE Computer Society,1995,Page(s):236 – 243.
[17] Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams[J]. ACM Computing Surveys,1992,24(3):293 – 318.
[18] Randal E Bryant,Christoph Meinel. Ordered binary decision diagrams[A]. Logic Synthesis and Verification[C]. US:Springer,2002.654:285 – 307.
[19] Rolf Drechsler,Detlef Sieling. Binary decision diagrams in theory and practice. International Journal on Software Tools for Technology Transfer[J],2001,3(2):112 – 136.
[20] Burnim J,Sen K. Heuristics for scalable dynamic test generation[A]. Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering[C]. Washington:IEEE Computer Society,2008,Page(s):443 – 446.
[21] Burnim J,Sen K. Heuristics for scalable dynamic test generation[R]. Berkeley:Electrical Engineering and Computer Sciences University of California at Berkeley,2008.1 – 10.
[22] Xie T,Tillmann N,de Halleux J,and Schulte W. Fitness-guided path exploration in dynamic symbolic execution[A]. Proceedings of the 39th International IEEE/IFIP Conference on Dependable Systems and Networks[C]. Washington:IEEE Computer Society,2009.359 – 368.
[23] 王颖,杨义先等.基于控制流序位比对的智能 Fuzzing 测试方法[J].通信学报,2013,34(4):114 – 121.
WANG Ying,YANG Yi-xian,NIU Xin-xin,GU Li-ze. Smart Fuzzing method based on comparison algorithm of control flow sequences[J]. Journal on Communications,2013,34(4):114 – 121.(in Chinese)
[24] Patrice Godefroid,Michael Y Levin,David Molnar. SAGE:Whitebox Fuzzing for Security Testing[J]. Communications of the ACM,2012,55(3):40 – 44.
[25] 网址[OL]:http://www.sans.org/top25-software-errors/,或 http://cwe.mitre.org/top25/
[26] 网址[OL]:http://cwe.mitre.org/top25/archive/2011/2011_onthecusp.html

作者简介

王　颖　女,1978 出生于吉林长春,博士研究生,主要研究领域为网络与信息安全.
E-mail:boblee2002@163.com

谷利泽　男,1965 年出生于辽宁省营口,北京邮电大学副教授,主要研究领域为现代密码学及其应用.

杨义先　男,1961 年出生于四川盐亭,北京邮电大学教授、博士生导师,主要研究领域为现代密码学、网络与信息安全.

董宇欣　女,1974 年出生于黑龙江省,哈尔滨工程大学副教授,工学博士,主要研究领域为社会网络、信任演化、智能信息处理等.