基于细粒度信息同步的软件混合模糊测试方法及设备

### 技术领域

本发明属于软件自动化安全检测技术领域，具体涉及基于细粒度种子信息同步的软件混合模糊测试方法及设备。

### 背景技术

软件安全测试对于软件质量的保证具有重要意义。为了提升软件安全测试的效率，出现了多种软件安全测试技术，主流的自动化软件漏洞检测技术主要有模糊测试和符号执行等，但是这些技术均面临着路径组合爆炸的问题，导致很难覆盖程序中所有的路径。

为了解决路径组合爆炸导致的低覆盖率问题，出现了混合模糊测试方法，该方法的主要思想就是将模糊测试与符号执行结合在一起，以模糊测试为主，探索程序中的路径，在遇到难以突破的复杂条件分支时，则由符号执行器负责求解，由此弥补两者技术上的缺点，充分发挥两者的优势，提升软件安全测试的效率。

但是在现有的以基于覆盖率引导的模糊测试和动态符号执行所构成的混合模糊测试模型中，均未考虑两者之间的种子信息同步问题。动态符号执行器在选择种子的过程中，由于缺少必要的种子信息，可能会选择到低质量的种子，而对低质量的种子进行符号执行所能产生的新分支数会远小于高质量的种子所产生的新分支数，从而影响混合模糊测试的整体的效率。

现有基于混合模糊测试进行相关研究的主要有以下开源测试框架、专利和论文：

1.Driller是一款开源的混合模糊测试框架，其模糊测试器采用AFL，动态符号执行器采用Angr。其运行模式是先调用AFL进行模糊测试，当AFL无法触发新的分支时，就调用符号执行器Angr从AFL的种子池中随机挑选种子进行符号执行，在产生能触发新的分支的种子后，便将新的种子交给AFL，继续执行模糊测试，如此循环。QSYM主要是为了适应混合模糊测试而研发的一款开源的动态符号执行器，QSYM可以搭配类AFL模糊测试器组成混合模糊测试模型，与Driller相比，两种混合模糊测试模型中最主要的区别就是动态符号执行器不同，QSYM通过指令级的符号仿真和非完全约束求解，解决了传统动态符号执行器存在的效率瓶颈问题，从而可以提升混合模糊测试的整体效率。但是Driller和QSYM均未考虑模糊测试器和动态符号执行器之间的信息同步问题，动态符号执行器对于种子的选择均存在一定的盲目性，会选择到低质量的种子，影响整体的混合模糊测试效率，从而降低软件漏洞检测的效率。

2.CN104375942A提出了一种面向二进制的混合模糊测试方法，采用二进制代码覆盖率监测作为中间层处理，计算模糊测试的覆盖率，当模糊测试的覆盖率不再提高时，就调用符号执行处理。CN108845944A对于混合模糊测试的创新在于通过路径覆盖率构建调度参数，并依据调度参数判断模糊测试器的状态，只有当模糊测试器处于低速状态时才调用符号执行。CN108052825A设计了一种针对二进制可执行文件的模糊测试与符号执行相结合的漏洞检测系统，该系统引入缓存探索器和任务协作模块，用以在二进制文件漏洞检测场景中加强检测的深度、提高检测效率。在CN110059010A中，利用动态符号执行遍历程序中的所有路径并产生测试用例，以此为初始种子池，调用模糊测试。然而在这些方法中，均未考虑混合模糊测试中的种子信息同步，动态符号执行器在种子选择时依旧存在盲目性的问题，导致软件漏洞检测效率不高。

综上，针对基于覆盖率引导的模糊测试器和动态符号执行器组成的混合模糊测试系统，由于两者之间种子信息交互不足，导致符号执行器在种子选择过程中存在盲目性，进而影响代码模糊测试效率的问题，本发明提出了一种基于细粒度信息同步的软件混合模糊测试方法，通过在模糊测试器和符号执行器之间交互种子覆盖路径和路径频率等种子细粒度信息，可以优先选择执行触发路径频率低的种子，提高软件模糊测试效率，从而提升整体的软件漏洞检测效率。

### 发明内容

本发明的一个目的是针对上述问题，提出了一种基于细粒度信息同步的软件混合模糊测试方法。本发明中的模糊测试器基于AFLFast开发，符号执行器基于QSYM开发，在此混合模糊测试系统上实现了细粒度信息同步方法。模糊测试器在每一轮测试开始前，会从种子池中选择当前轮次需要变异的种子，AFLFast的选择依据是优先选择能够触发低频路径的种子，通过细粒度信息同步方法，种子对应的路径信息会同步给符号执行器，由符号执行器对种子的优先级进行排序，优先选择路径覆盖频率最低的种子进行解析。// 整体思路概括

本发明提出了一种基于细粒度信息同步的软件混合模糊测试方法，主要包括四个阶段：测试准备阶段、模糊测试阶段、符号执行阶段、结果信息反馈阶段。

进一步的，本发明所述的测试准备阶段，具体包含以下步骤：

1-1. 基于aflfast自带的gcc编译工具对待测的开源目标程序进行插装编译，产生提供覆盖率信息的二进制可执行文件；

1-2. 初始化种子池，以供模糊测试器使用；其中的种子指目标程序的输入。

进一步的，本发明所述的模糊测试阶段，具体包含以下步骤：

2-1. 种子选择

将种子池中所有种子依次输入给目标程序进行测试运行，同时获取所有种子测试运行时的种子信息；模糊测试器依据上述所有种子的种子信息，从中选择优质种子，并将其加入到优质种子队列。

所述种子信息包括种子的路径覆盖情况（种子在被程序执行时所能覆盖到的程序中的代码块）、种子对应的路径覆盖频率信息和种子的变异次数等。

所述的优质种子选择标准为：1.能覆盖新路径的种子。2.种子文件的大小尽可能的小。

2-2优质种子信息同步

将优质种子的细粒度信息保存到details文件；其中细粒度信息包括种子名称、路径覆盖频率信息；

2-3 种子变异及测试

对优质种子队列中每个优质种子进行种子变异并测试，直至所有优质种子遍历完成，返回至步骤2-1，并重置优质种子队列为空；

所述的种子变异并测试过程如下：

1）对优质种子进行随机性变异，产生测试用例。

2）用例测试

将种子变异产生的测试用例通过进程管道输送给目标程序，记录目标程序运行过程中所获得的种子信息，并检测程序是否异常。如果程序运行异常（即认为该异常为软件漏洞），则将测试用例保留至crashes目录中。如果测试用例能覆盖新的路径，则将其加入到种子池。

进一步的，本发明所述的符号执行阶段，具体包含以下步骤：

3-1.获取种子细粒度信息并排序。

从details文件中获取种子的细粒度信息，并依据种子对应的路径覆盖频率进行从低至高排序，产生种子优先级序列。

路径覆盖频率越低的种子所具有的优先级越高。

3-2.获取种子，进行符号执行。

依据种子优先级序列，从模糊测试器的种子池中依次挑选出对应的种子，对其进行冗余性检验，即判断该种子之前是否已经被执行过，若未执行过则进行符号执行，反之则跳过该种子，进行下一个种子的冗余性检验。

3-3. 对符号执行的结果进行处理并将其输出。

目标程序进行动态符号执行后会产生新的种子，这些新的种子进行冗余性检验，即判断该种子之前是否已经存在于输出目录，若不存在则添加，反之则跳过该种子，返回至步骤3-2，直至种子优先级序列对应的种子全部完成符号执行。

作为优选，步骤3-3具体是符号执行器会通过seekPath()搜索种子在目标程序中所覆盖的路径，然后在各条件分支处对约束条件进行取反并利用getConstraint()生成约束表达式，通过solve()对约束表达式求解产生能覆盖新条件分支的新种子；将新种子的路径覆盖情况与历史路径覆盖情况做对比，若能覆盖新的路径，则将种子输出，并存于种子输出目录；

进一步的，本发明所述的结果信息反馈阶段，具体是：

模糊测试器定期同步符号执行器输出目录中的所有种子，将能覆盖新路径的种子添加入种子池中。

本发明的另一个目的是提供基于细粒度信息同步的软件混合模糊测试设备，包括：

测试准备模块，用于基于aflfast自带的gcc编译工具对待测的开源目标程序进行插装编译，产生提供覆盖率信息的二进制可执行文件；并初始化模糊测试器的种子池；

模糊测试模块，用于将模糊测试器的种子池中所有种子依次输入给目标程序进行测试运行，同时获取所有种子测试运行时的种子信息；模糊测试器选择优质种子，然后将优质种子的细粒度信息保存到details文件；对每个优质种子进行种子变异并测试；

符号执行模块，用于从details文件中获取种子的细粒度信息，并产生种子优先级序列；依据种子优先级序列，从模糊测试器的种子池中依次挑选出对应的种子，对其进行初步冗余性检验，若未执行过则进行动态符号执行，反之则跳过该种子；

结果信息反馈模块，用于模糊测试器定期同步符号执行器输出目录中的所有种子，将能覆盖新路径的种子添加入模糊测试器的种子池中。

本发明的有益效果是：

1. 本发明采用了基于细粒度信息同步的软件混合模糊测试方法，对种子的优先级按照种子的细粒度信息进行划分，可以有效的提升混合模糊测试的整体效率。
2. 相对于已有工具Driller这类传统的混合模糊测试器，本发明提出的基于细粒度信息同步的软件混合模糊测试方法，可以有效的解决传统混合模糊测试器中由于种子信息交互不足而导致的符号执行器在种子选择时存在的盲目性的问题。在本发明中，通过在模糊测试器运行过程中与符号执行器交互细粒度种子信息，符号执行器可以选择到更优质的低频路径种子，发现更多未曾覆盖的路径，从而提升混合模糊测试的效率。
3. 使用本发明方法，可以提升软件漏洞的检测效率，在相同的测试时间内，能探索软件中更深层次的代码部分，发现更多的软件漏洞。

### 附图说明

图1为本发明基于细粒度信息同步的软件混合模糊测试方法的流程示意图。

图2为本发明基于细粒度信息同步的软件混合模糊测试方法中的进程间通信图。

图3为本发明基于细粒度信息同步的软件混合模糊测试方法中模糊测试流程示意图。

图4为本发明基于细粒度信息同步的软件混合模糊测试方法中符号执行流程示意图。

### 具体实施方式

下面将结合本发明实施例中的附图，对本发明中的技术方案进行完整描述。

如图1所示，基于细粒度信息同步的软件混合模糊测试方法，整体步骤分为四个阶段，测试准备阶段、模糊测试阶段、符号执行阶段、结果信息反馈阶段。

1 测试准备阶段包含以下步骤：

1.1 借助aflfast的编译工具，对目标程序进行插装编译，用于进行基于覆盖率引导的模糊测试。

1.2 构建初始种子池SeedPool，完成模糊测试的初始化。SeedPool中的seed是指目标程序的输入。

2 模糊测试阶段主要包括以下步骤（如图3所示）：

2.1 种子选择。模糊测试器会进行多轮次的反复测试，在每一轮的测试之前，会依据种子的相关信息对种子进行综合性选择，优先选择能覆盖新的路径且对应路径覆盖频率最低的种子，这些被选择出来的优质种子在变异时有更大的概率覆盖新的路径，提高目标代码的覆盖率。

2.2 优质种子信息同步。在每一轮种子选择完成之后，模糊测试器都会将种子的细粒度信息输出至details文件中，过程如图2中所示。种子的细粒度信息指的是种子的名字及种子覆盖路径的频率信息——（seedname, frequence），示例如下：

|  |
| --- |
| *output//slave1/queue/id:002771, src:002752,op:havoc,rep:16#12* |

输出信息中的每一行代表一个种子记录，用“#”将种子名字与种子覆盖路径频率分开。

为了实现种子信息同步，需要在模糊测试器的*cull\_queue()*函数中添加代码，其伪代码如下：

|  |
| --- |
| **Input:** 模糊测试器的种子池*SeedPool* |
| **Output:** 种子细粒度信息列表*LowSeedList* |
| **Function:** *SyncSeedInfo* 种子信息同步 |
| 1. *details.clear(); // 清除前一轮的种子细粒度信息* |
| 1. ***foreach*** *seed* ***in*** *SeedPool* ***do*** *//遍历种子池中的种子* |
| 1. ***if***  *isHighQualitySeed(seed)* ***then*** *//判断种子是否是优质种子* |
| 1. *frequence = getPathFrequence(seed); //获取种子对应的路径频率* |
| 1. *LowSeedList.add(seedname, frequence); //存储种子名字及对应路径的频率* |
| 1. ***end*** |
|  |
| 1. *details.save(LowSeedList);* |

2.3 种子变异及测试。通过模糊测试器的变异函数fuzz\_one()对优质种子队列中的种子依次进行bitflip、arith、interest、dictionary、havoc、splice 变异生成新的测试用例。然后将种子变异产生的新的测试用例通过管道输送给目标程序，并监测目标程序的状态，如果程序运行异常，则将测试用例保留至crashes目录中。

3 符号执行阶段主要包括以下步骤（如图4所示）：

3.1 获取种子细粒度信息并排序。相较于之前的混合模糊测试方法中符号执行器对种子的盲目性选择，本发明提出通过利用模糊测试过程中的一些过程信息来指导符号执行器进行种子选择。符号执行器会同步模糊测试器所输出的种子信息，并对其依据路径频率进行排序。具体步骤如下：

3.1.1 种子信息读取。符号执行器会通过函数ReadSeedInfo()定期去读取details文件中的种子信息，在对种子信息进行处理后存入PrioritySeedList。

3.1.2 优先级排序。对PrioritySeedList种子队列中的种子依据其路径覆盖频率从小到大排序，路径频率越低的种子其优先级越高。

3.2 获取种子，进行符号执行。

3.2.1 获取种子，进行冗余性检验。符号执行器在对目标程序进行符号执行之前，会对选出来的种子进行冗余性判断，排除覆盖路径已经被解析过的种子。种子冗余性判断的具体过程是：首先运行目标程序，从PrioritySeedList中选择种子输入给目标程序。目标程序在对种子进行处理的过程中会触发程序中的插装点，该信息会被记录到bitmap（代表种子的路径覆盖情况）中。之后，将种子的bitmap与历史的路径覆盖情况GlobalMap进行对比，如果bitmap能覆盖GlobalMap中未曾覆盖的路径，该种子便为非冗余种子，可以用于进行符号执行。该过程的伪代码如下所示：

|  |
| --- |
| **Input:** 待检验的种子*seed，*全局位图*GlobalMap* |
| **Output:** *True* (种子冗余)/*False* (种子非冗余) |
| **Function:** *JudgeSeed* 种子冗余性判断 |
| 1. *bitmap = getBitmap(seed); // 获取种子的路径覆盖位图* |
| 1. ***if*** *bitmap ∪ GlobalMap > GlobalMap* ***then*** |
| 1. *GlobalMap = bitmap ∪ GlobalMap; // 更新历史路径覆盖位图GlobalMap* |
| 1. *return False;* |
| 1. ***end*** |
| 1. *return True;* |

3.2.2 符号执行。在对PrioritySeedList中的种子进行冗余性判断之后，符号执行器会通过seekPath()搜索种子在目标程序中所覆盖的路径，然后在各条件分支处对约束条件进行取反并利用getConstraint()生成约束表达式，通过solve()对约束表达式求解可以产生能覆盖新分支的种子。

|  |
| --- |
| **Input:** 符号执行种子*seed* |
| **Output:** 能覆盖新分支的种子队列*resSeedList* |
| **Function:** *SymbolicExecution*符号执行 |
| **1 *while*** *( condstate = seekPath(seed) ) != null* ***do*** *// 搜索路径中的条件分支* |
| **2**  *cons\_expr = getConstraint(condstate) // 生成约束表达式* |
| **3** *resSeed = solve(cons\_expr) // 约束表达式求解* |
| **4** *resSeedList.add(resSeed)* |
| **5 done** |

3.3 对符号执行的结果进行处理并将其输出。

3.3.1结果处理。由于不同种子所覆盖的路径可能存在部分重叠，所以导致符号执行中产生的种子会存在重复的可能，需要对其进行冗余性检验后方才可以输出。将待测种子的路径覆盖情况与历史种子路径覆盖情况做对比，若能覆盖新的路径，则将种子输出，否则，种子会被抛弃。其实现思想与3.2.1节中种子冗余性判断相同。

3.3.2 结果保存。如图2中所示，将经过处理的种子存放到种子输出目录下，以供模糊测试器使用。

4 结果信息反馈阶段主要包括以下步骤：

4.1 如图2中所示，模糊测试器会定期同步由符号执行器所产生的种子，将其存入种子池中。

综上所述，本发明基于细粒度信息同步的软件混合模糊测试方法，通过模糊测试器和符号执行器之间的种子信息同步，指导符号执行器进行种子选择，选择更优质、能产生更大收益的种子。

为了验证本发明方法的有效性，进行了实验验证，其中对照组采用未经优化的AFLFast+QSYM组成的混合模糊测试系统（qsym-old），而实验组则在对照组的基础上使用了本发明方法进行优化（qsym-new）。采用UNIFUZZ平台中所集成的6款测试软件作为测试集,并且为了保证实验数据的准确性，每次实验都会重复三次，结果取平均值，最终的实验数据结果如下表所示：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 符号执行器种子生成数 | | | | | |
|  | mujs | cflow | pdftotext | imginfo | jhead | mp42aac |
| qsym-old | 255 | 385.3 | 403.3 | 233.67 | 190 | 580.33 |
| qsym-new | 699 | 757 | 499.7 | 268.67 | 271 | 997.33 |
| 提升率 | +174% | +96.47% | +23.9% | +14.98% | +42.63% | +71.86% |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 符号执行中种子的非冗余率 | | | | | |
|  | mujs | cflow | pdftotext | imginfo | jhead | mp42aac |
| qsym-old | 80.07% | 63.72% | 67.34% | 91.63% | 76.6% | 88.25% |
| qsym-new | 96.01% | 97.9% | 93.98% | 99.31% | 95.84% | 92.48% |
| 提升率 | +19.9% | +53.64% | +39.56% | +8.38% | +25.12% | +4.79% |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 发现的漏洞数 | | | | | |
|  | mujs | cflow | pdftotext | imginfo | jhead | mp42aac |
| qsym-old | 0 | 58.3 | 2.3 | 0 | 1 | 409 |
| qsym-new | 0 | 63.6 | 7.3 | 0 | 0 | 431.67 |
| 提升率 | +0% | +9.1% | +217% | +0% | 无 | +5.54% |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 路径覆盖数 | | | | | |
|  | mujs | cflow | pdftotext | imginfo | jhead | mp42aac |
| qsym-old | 3714 | 2680.6 | 1223.3 | 1132.3 | 508 | 2977 |
| qsym-new | 3783 | 2752 | 1347 | 1162.33 | 524 | 2971 |
| 提升率 | +1.86% | +2.66% | +10.1% | +2.65% | +3.14% | 无 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 位图覆盖密度 | | | | | |
|  | mujs | cflow | pdftotext | imginfo | jhead | mp42aac |
| qsym-old | 6.987% | 3.733% | 2.80% | 5.85% | 0.96% | 5.54% |
| qsym-new | 7.14% | 3.77% | 2.89% | 5.98% | 0.98% | 6.38% |
| 提升率 | +2.19% | +1.0% | +3.21% | +2.22% | +2.08% | +15.16% |

从实验数据中可以看出，使用了本发明方法后，相较于之前的混合模糊测试系统，在符号执行器种子生成数、种子的非冗余率、漏洞数、路径覆盖数和位图覆盖密度方面均有效果上的提升。因此，本发明方法可以有效的提高软件漏洞检测的效率。

上面结合附图对本发明的实施方式做了详细说明，凡依本发明申请发明范围所作的均等变化与修饰，都属于本发明保护的范围。