# 基于符号执行和模糊测试的混合模糊测试技术综述

\*\*摘要\*\*：模糊测试现在已经发展成为一种高效的漏洞挖掘方法，符号执行也是当前流行的软件漏洞挖掘技术，两者都是网络与信息安全领域的研究热点。混合模糊测试是在传统模糊测试的基础上加入了符号执行技术，目前已经发展成为模糊测试的新分支。文章对现有的混合模糊测试方法进行研究，综述了混合模糊测试的发展演变过程和技术核心，并通过一种基于符号执行的实验方法对目前知名的混合模糊测试进行性能对比。最后讨论了目前混合模糊测试领域研究存在的问题，并尝试对其未来发展趋势进行了展望。

\*\*关键词\*\*：混合模糊测试、模糊测试、符号执行、覆盖率

## 1. 引言

近年来，漏洞已经成为安全领域，漏洞报告数量由2010年的5,223起【引-cnvd报告】增长为2019年的16,016起【引-cnvd报告】，已经对人们的日常生活产生了严重影响【引-报告】。随着软件数量的井喷式增长，以及恶意软件漏洞利用事件的相继爆发，自动化软件漏洞挖掘技术开始变得尤为重要。在自动化软件漏洞挖掘领域中，有两项发展迅猛的技术：模糊测试技术和符号执行技术。

模糊测试发展至今已经成为新世纪以来实践效果良好，且得到学术界和工业界广泛应用的软件漏洞挖掘技术【引-操作系统内核综述】。

模糊测试（Fuzzing）是目前最流行的自动化软件漏洞挖掘工具，通过将大量随机的测试用例输入到待测试程序（经过插桩处理的测试程序），获取覆盖率等测试信息以及可能会出现的崩溃信息。模糊测试具有自动化程度高、消耗率低、计算资源利用率高等优点，属于程序动态分析技术，已经被很多实践证实效果良好[1],[2],[3]，得到了学术界和工业界广泛应用。在漏洞挖掘方面，模糊测试已经发挥了非常惊艳的效果，仅拿AFL（America Fuzzy Lop，以下简称AFL）[1]这一典型基于变异的模糊测试工具举例，从2014年开发至今，已经发现了数百CVE，这还不包括诸多基于AFL的论文所发现的CVE漏洞[4]，[5]，[6]，以及未被披露的漏洞。但是高度自动化同时也导致模糊测试做不到对待测试程序的所有可能路径进行测试，不能覆盖待测试程序的所有状态，因此面对分支复杂的应用程序只能做到尽快达到路径的深度，做不到全覆盖状态的广度。

符号执行是一种程序分析技术，关键思想在于将程序的输入（input）用符号来表征而不是具体值，然后通过分析程序来得到程序的输入，并保证输入可以让特定区域代码执行[7]。如果将待测试程序按照符号执行树的形式进行直观展示，就可以很容易的理解了，符号执行在复杂分支面前有很大的优势，通过程序路径探索可以突破浅位置的复杂分支，做到程序状态全覆盖，但是因为计算量限制等问题并不能到达更深的路径。显然模糊测试和符号执行各有优缺点，混合模糊测试技术就是结合了双方的优点，从而到达更深的路径，获得更高的覆盖率。混合模糊测试比单独使用符号执行或者模糊测试效果都要更好[8],[9]。

本文将对混合模糊测试这一自动化软件漏洞挖掘技术进行发展回顾，对关键技术进行讨论，对技术发展过程中的典型工具进行综述。最后讨论了混合模糊测试今后的发展方向，并对本文进行了相关总结。

## 2. 技术概述

### 2.1 漏洞挖掘技术

当前用于漏洞挖掘的技术有很多，最主要的有三种：静态分析、动态分析和符号执行[10]，这三项技术各有优缺点。

静态分析是一种在不执行待测试程序的情况下对其进行分析的软件测试技术，常用的分析技术有词法分析、语法分析、抽象语法树分析、语义分析、控制流分析、数据流分析、污点分析、无效代码分析等，广泛应用于漏洞挖掘。静态分析的优势在于对上万行代码可以进行快速分析，并用形式化的方法给出了可证明的、完整的分析结果[11]。静态分析的缺点也很明显，分析结果未能给出精准的POC（漏洞触发证明样例），分析的目的是挖掘软件的漏洞，得到的分析结果却不能产生可触发漏洞的输入，比如对于PNG图片读取型待测试程序，即使用静态分析得到了结果，但是不能得到可以触发某漏洞的PNG图片，需要根据分析结果自己构建PNG文件。

动态分析可以执行待测试程序，通常利用插桩方式来获得各类路径，通过路径可以获得覆盖率信息。不同于静态分析，动态分析可以监控待测试程序的运行，当程序出现运行错误时，将触发此错误的输入保存下来，如果同样运用到PNG图片读取型待测试程序，就可以保存得到能触发错误的相应PNG图片文件。动态分析还可以检测静态分析无法检测到的依赖项，比如多态的动态依赖关系。动态测试的缺点主要是来自于插桩技术，插桩技术是在待测试程序中插入特定操作，依此来探测程序达到测试的目的，因此插入的此部分代码（高级语言或编程语言）一定会对原来的待测试程序产生一定的负面性能影响。另外一个缺点是无法得到全部的覆盖率，因为待测试程序的运行是基于用户交互或自动化测试，并不能保证程序所有可能位置的全部覆盖。

符号执行技术的出现可以追溯到1976年，King J C. 等人提出这一思想[12]。符号执行利用程序解释和约束求解技术生成程序输入来探索待测试程序的状态空间，以此来触发漏洞。但是因为符号执行要尽可能把所有程序路径覆盖，比如对于一个条件判断语句就要生成两条路径来保证当前环节全覆盖，所以通常会在待测试程序中触发大量的路径，也就是“路径爆炸”。[12]

虽然漏洞挖掘的主流是以上三类，但现在的研究已经不再单纯使用某一种技术，而是倾向于多种技术结合。模糊测试最初的设计思路是属于动态分析[13]，通过插桩技术对待测试程序进行动态监控，但是目前主流的模糊测试工具中也加入了静态分析技术，比如基于模糊测试和污点分析技术的T-Fuzz[14]。

### 2.2 模糊测试技术

模糊测试技术最早于1995年由Miller B P, Koski D等人提出，针对UNIX系统进行安全测试并发现组件中诸多漏洞[13]。模糊测试的核心在于非预期的输入，不管是基于变异还是基于生成的模糊测试，都是为了得到不同于人工测试的异常输入，而这些输入一旦触发程序异常就会保存下来，最后再由漏洞利用人员进行分析。如图1.所示 模糊测试的核心过程是这样的，给模糊测试工具初始种子，然后经过变异得到输入数据流，模糊测试工具将输入数据流传给待测试程序，并通过插桩技术对程序进行跟踪，获得覆盖率信息，然后覆盖率指导之前的流程，周而复始，直到遇到崩溃情况，最后由研究人员对崩溃进行评估与POC验证。在模糊测试的流程图中，每个阶段都有改进空间。这其中最重要的有三样：种子集合、模糊测试工具、待测试程序。

种子集合是模糊测试的基石，大量的随机输入都是来自于种子的变异技术，高质量的种子对于模糊测试的效率来说很重要，所以像AFL等工具都会提供种子库。对于像SLF[15]这种可自发生成种子的模糊测试工具来说可以不需要提供种子集合，但是在SLF的测试程序之初也会生成预定的种子以数据流的形式代替种子的作用，所以从根本上来说种子必不可少。

模糊测试工具是整个测试的核心，目前的主流模糊测试工具都要具备两种能力：一是产生大量随机输入的能力，当得到种子文件（数据流）后，需要快速生成大量的输入，每个输入都会以数据流的形式传送给待测试程序，并且执行一定的路径，如果想要尽可能多的走过程序的各条路径，就需要尽可能多的生成不同的输入；二是同时执行多程序的协调能力，模糊测试虽然是一个死循环，但是并不是意味着一次测试结束后才进行第二个输入的测试，因为大量的输入从种子文件的基础上生成，按序进行作业的情况下，待测试程序执行的阶段会让大量的输入数据流暂存在内存中，使程序执行成为流程的瓶颈，也会严重拖慢模糊测试整体的效率，因此快速有效执行待测试程序是模糊测试技术的第二个重要的能力。

待测试程序是模糊测试的最终目的，一切测试都是围绕着待测试程序进行，对于不同的测试目标待测试程序会出现一定形式上的差异，而且需要研究人员手动干预，针对不同类型的目标进行相应的调整。比如对于库的测试，通常需要测试人员对于源码有一定的理解，然后根据不同的目标函数，编写不同的程序。

### 2.3 混合模糊测试技术

\*\*添加部分技术细节500字\*\*

正如前面所述，现在的模糊测试在各个阶段都有一定的改进空间，通过结合其他技术已经发展出很多分支，其中混合模糊测试技术就是结合了模糊测试技术和符号执行技术，已经有工作证明，混合模糊测试技术在漏洞自动化挖掘方面表现出不错的效果【9】，并且两者结合比单独使用任意一种，代码覆盖率和漏洞发现效率方面都要得到更好的效果。

## 3. 混合模糊测试研究现状

2007年，Miller B.P.等人提出混合测试的概念【13】，将符号执行和自动化软件测试结合起来。符号执行的其中一个优点是能够在指定位置的邻近区域探寻到全部到状态，模糊测试的其中一个优点是快速深入程序的状态，作者将两者结合提出一种混合测试的方式，并进行实验证明混合后的效果比两者单一情况下要好很多。在试验阶段，通过在CUTE的程序设计顶层加入混合执行接口，分别对一个红黑树结构程序和VIM软件进行漏洞挖掘测试，并给出实验数据。作者没有明确指出两种技术是否互相弥补缺点，而且从实验结果来看混合测试结合了两者的优点，作者也并未对两种技术进行改进。虽然在本文中作者并没有明确提出混合模糊测试技术，但是其中用到的随机测试思想跟模糊测试一致，所以此项研究成为后来众多混合模糊测试的概念原型。2012年Pak B S. 等人首次提出混合模糊测试（Hybrid Fuzzing）这一名词概念[17]。

在混合模糊测试被提出之后，其在漏洞挖掘及软件测试领域却一直并没有大的进展。随着计算机的算力得到不断的提升和算法的研究，符号执行技术在约束求解等等方面得到了一定的突破。与符号执行相比模糊测试一直没有标志性的进展，混合模糊测试也一直没有得到突破性的进展。2015年，AFL[5]的出现给工业界和学术界带来了新思路，是模糊测试里程碑式发展，也是混合模糊测试技术的转折点，覆盖率为导向的思想给混合模糊测试带来了设计思路，并且 AFL 中的 fork-server 技术也为后来的混合模糊测试技术提供了实现思路。

2016年，N.J等人提出了一种全新的混合模糊测试技术——Driller。不同于2007年的混合测试，Driller利用fork-server思想结合了当时效率最好的模糊测试技术【1】和符号执行技术，并实现了对二进制文件的高效漏洞挖掘。Driller奠定了混合模糊测试技术的发展方向，模糊测试技术加符号执行技术。通过监测AFL的执行，我们可以决定什么时候开始符号执行以探索新路径。如果AFL执行了x轮后，bitmap上显示没有发现新的状态转换（也即新的代码块转移），说明AFL卡住了，这时候调用angr进行符号执行。每个具体输入对应于PathGroup中的单个路径， 在PathGroup的每一步中，检查每个分支以确保最新的跳转指令引入先前AFL未知的路径。 当发现这样的跳转时，SMT求解器被查询以创建一个输入来驱动执行到新的跳转。这个输入反馈给AFL，AFL在未来的模糊步骤中进行变异。 这个反馈循环使我们能够将昂贵的符号执行时间与廉价的模糊时间进行平衡，并且减轻了模糊对程序操作的低语义洞察力.

混合模糊测试技术出现爆发，国内外研究人员纷纷被吸引过来。2018年由Yun I, Lee S 等人提出的QSYM[18]，2019年Chen Y. 等人提出的SAVIOR[19]，在深入分析了混合模糊测试的优缺点的同时，都表现出了优异的效果。中国由谢肖飞、李晓红等人提出的Afleer【9】通过一种基于分支覆盖的方法将AFL【1】和KLEE【20】结合，利用AFL生成大量测试用例，再基于KLEE对AFL得到的覆盖信息进行搜索，将搜索结果用于指导测试用例生成，得到只覆盖未覆盖分支的测试用例。作者通过LAVA-M数据集进一步证实混合模糊测试技术比单独使用模糊测试或者符号执行都要得到更好的测试结果。

2020年，混合模糊测试技术已经成为最重要的覆盖率为导向模糊测试分支之一，研究人员开始不满足于对结合方式的改进，通过改进符号执行来更好的服务模糊测试技术。Rongxin Wu. 等人提出的 PANGOLIN 【21】通过对符号执行中的约束求解部分进行修改来提高漏洞挖掘效率的。首先PANGOLIN提出了一种多面体路径求解的方式来提升约束求解的效率，并且将这种约束求解的结果用来指导模糊测试阶段的变异过程，将符合执行与模糊测试过程更紧密的结合在一起，实现漏洞挖掘效率提升。

|  |  |  |
| --- | --- | --- |
| Year | Method | Features |
| 2007 | Hybrid testing | The concept of hybrid testing was first proposed. |
| 2012 | Hybrid fuzzing | Hybrid fuzzing is officially proposed for the first time. |
| 2016 | Driller | Driller is based on the fuzz tool AFL and the symbolic execution tool angr. When the fuzzy program is stuck, the symbolic execution is called to solve the input that can reach the new path, so that the fuzz can quickly break through the conditional judgment statement. |
| 2018 | QSYM | A hybrid fuzzing technique that focuses more on symbolic execution. QSYM is a practical concolic execution engine tailored for hybrid fuzzing. |
| 2019 | Afleer | The trend of domestic research on hybrid fuzzing begins. |
| 2020 | PANGOLIN | For the first time, it is optimized for symbolic execution in hybrid fuzzing, not fuzzing. |

混合模糊测试在2012年和2016年的两次跃进，分别是以符号执行为核心的进步，和以模糊测试为核心的进步。前者是因为KLEE等众多符号执行引擎的出现，使得符号执行的效率得到极大进步，从而使符号执行和模糊测试结合成为了可能，符号执行中的求解速度能够跟得上模糊测试。后者是因为模糊测试的发展，像AFL、Libfuzzer等的出现，使得覆盖率为导向的模糊测试成为主流，而这也使得混合模糊测试能有迎来新的技术思想。将约束求解和覆盖率进行结合也成为之后几年的主流。到目前为止，混合模糊测试已经有了很多效果不错的工具，不管是在学术上还是在工业界。

## 4.发展方向【2000字】

在模糊测试方面，多种技术融合的模式已经成为主流，作为最重要的分支之一：混合模糊测试技术，是基于符合执行技术和模糊测试技术相结合的，但是本质上思想的主体还在于模糊测试。而覆盖率为导向的模糊测试也是最重要的一种模糊测试技术，常被用于结合进混合模糊测试技术，因此可以从有利于提升覆盖率的方面着手，对符合执行的结合方式进行改进，比如利用插桩信息进行约束求解，来加快模糊测试过程中覆盖率的提升速度。因此从如果利用符合执行提升覆盖率为出发点，即保持了混合模糊测试中符合执行的优势又提升了原本模糊测试的覆盖率提升速度，是未来的趋势。除此之外，在如何将符合执行与模糊测试更好的结合方面也是一个发展方向，更高效的结合方式会提升模糊测试程序本身的效率，从而提升漏洞挖掘的效率。

虽然针对混合模糊测试的研究已经有很多，也有很多发展期潜力。但是必须要承认，混合模糊测试技术面临着诸多缺陷和挑战，第一，在很多情况下混合模糊测试依旧是低效的，比如针对一些大型软件的漏洞挖掘，不管是Driller还是PANGOLIN，都没有表现出很好的效果，甚至因为加入符号执行，使得混合模糊测试比单纯使用模糊测试效率还低。第二种情况是开销大，在计算资源的分配中，约束求解是昂贵的，当覆盖率低于20%时，Driller执行程序可能会导致超过程序本身100倍的运行开销。

正如PANGOLIN所做的，改进符号执行本身也是一种提升效率的方式，通过使用更先进的符号执行引擎或者使用新的约束求解算法，来提升符号执行的效率，从而达到提升混合模糊测试效率的目的。混合模糊测试不仅包含符号执行，还包括模糊测试。使用比AFL更快，效率更高的模糊测试工具，也可以提升混合模糊测试的整体漏洞挖掘效率。

总结下来，发展方向分为三类：改进模糊测试和符号执行的结合方式；提升混合模糊测试的效率或者降低资源开销；从模糊测试或者符号执行上下手来提升效率。

## 5. 结束语

自混合模糊测试被提出以来，经历了两次热潮，第一次是正式被提出，被命名为 Hybrid Fuzz【17】，从此研究人员开始关注这样一个分支。第二次是结合afl【1】的driller【10】，确定了符合执行加模糊测试的模式，大量研究人员开始重视此类研究，混合模糊测试技术成为模糊测试技术最重要的分支之一。随着符合执行的发展和模糊测试的发展，混合模糊测试技术也会有持久的发展。

## 参考引用

[1]. M. Zalewski, *American fuzzy lop,* <https://lcamtuf.coredump.cx/afl/>

[2]. Google: honggfuzz (2018). <https://github.com/google/honggfuzz>

[3]. Serebryany K. OSS-Fuzz-Google's continuous fuzzing service for open source software[J]. 2017.

[4]. Gan S, Zhang C, Qin X, et al. Collafl: Path sensitive fuzzing[C]//2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 679-696.

[5]. C. Lyu, S. Ji, C. Zhang, Y. Li, W. H. Lee, Y. Song, R. Beyah(2019). MOPT: Optimized mutation scheduling for fuzzers. In *28th USENIX Security Symposium. (USENIX Security 19)* , 1949-1966.

[6]. Böhme M, Pham V T, Roychoudhury A(2017). Coverage-based greybox fuzzing as markov chain. IEEE Transactions on Software Engineering, 45(5): 489-506.

[7]. Baldoni R, Coppa E, D’elia D C, et al. A survey of symbolic execution techniques[J]. ACM Computing Surveys (CSUR), 2018, 51(3): 1-39. 【符号执行综述】

[8]. Majumdar, Rupak , and K. Sen . "Hybrid Concolic Testing." International Conference on Software Engineering IEEE, 2007.

[9]. 谢肖飞, XIE XiaoFei, 李晓红,等. 基于符号执行与模糊测试的混合测试方法[J]. 软件学报, 2019, Vol.30Issue(10):3071-3089.

[10]. Stephens N, Grosen J, Salls C, Dutcher A, Vigna G(2016, February). Driller: Augmenting Fuzzing Through Selective Symbolic Execution. NDSS, 16(2016): 1-16.

[11]. Chess B, McGraw G. Static analysis for security[J]. IEEE security & privacy, 2004, 2(6): 76-79.

[12].King J C. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394.

[13]. Miller B P, Koski D, Lee C P, et al. Fuzz revisited: A re-examination of the reliability of UNIX utilities and services[R]. University of Wisconsin-Madison Department of Computer Sciences, 1995.

[14]. 【T-Fuzz污点分析模糊测试】Peng H, Shoshitaishvili Y, Payer M. T-Fuzz: fuzzing by program transformation[C]//2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018: 697-710.

【15】. You W, Liu X, Ma S, Perry D, Zhang X, Liang B(2019, May). SLF: fuzzing without valid seed inputs. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 712-723.

[17]. 【2012年混合模糊测试】Pak B S. Hybrid fuzz testing: Discovering software bugs via fuzzing and symbolic execution[J]. School of Computer Science Carnegie Mellon University, 2012.

[18]. QSYM Yun I, Lee S, Xu M, et al. {QSYM}: A practical concolic execution engine tailored for hybrid fuzzing[C]//27th {USENIX} Security Symposium ({USENIX} Security 18). 2018: 745-761.

[19]. 【SAVIOR】Chen Y, Li P, Xu J, et al. SAVIOR: towards bug-driven hybrid testing[C]//2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020: 1580-1596.

[20]. KLEE 【】 Cadar C, Dunbar D, Engler D R. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs[C]//OSDI. 2008, 8: 209-224.

[21]. Pangolin

PANGOLIN: Incremental Hybrid Fuzzing withPolyhedral Path AbstractionHeqing Huang†, Peisen Yao†, Rongxin Wu‡, Qingkai Shi†, Charles Zhang††The Hong Kong University of Science and Technology, China‡Xiamen University, China