doi :10.3969/j.issn.1671-1122.2014.03.016

Fuzzing 测试技术综述

史记，曾昭龙，杨从保，李培岳

（中国人民公安大学网络安全保卫学院，北京 100038）

摘　要：Fuzzing 测试技术能够自动识别出二进制文件中的安全问题，这是一个新兴值得研究的领域。Fuzzing 技术从早期的手工方式逐渐发展为现在的自动化甚至智能化方式。文章对现有的 Fuzzing 测试方法进行了比较和研究，总结目前 Fuzzing 测试框架的特点，并预测了其未来发展方向。

关键词：软件安全；Fuzzing 测试；代码覆盖率

中图分类号：TP309　　**文献标识码：**A　　**文章编号：**1671-1122（2014）03-0087-05

The Summary of Fuzzing Testing Technology

SHI Ji, ZENG Zhao-long, YANG Cong-bao, LI Pei-yue

(*College of Network Security, People's Public Security University of China, Beijing* 100038, *China*)

Abstract: Fuzzing is a method that can identify security issues in binary files automatically. This is a new but promising field. Fuzzing has developed from manual ways to automatic and intelligent ways. In this paper we made comparisons between different fuzzing methods and summarized the characteristics of existing Fuzzing frameworks. Finally, we gave the future of fuzzing.

Key words: software security; fuzzing testing; code coverage

1 Fuzzing 技术介绍

在信息安全对抗过程中，防守方会采取各种产品和策略对威胁进行防护，例如杀毒软件、入侵检测设备等，这些措施尽管能在一定程度上减轻攻击所带来的损失，但这仅是一种被动的防御方法，且仅当恶意代码被公开以后才能够进行阻断，因此并不是防御的最好方式。

我们要能在被动防御的同时，主动找出软件中存在的未知安全漏洞，全面保护信息系统的安全。为此，人们在不断尝试着利用各种方法找出软件安全问题并对它们及时进行修复以阻止恶意攻击者利用漏洞进行攻击。在进行漏洞挖掘时，Fuzzing 技术是一种快速并且有效的方式。其在早期可以分为黑盒测试、白盒测试以及灰盒测试。在当前的商业软件中，软件提供商一般不会给出软件的源代码，因此多数情况下 Fuzzing 测试主要面向黑盒以及灰盒，其基本思想是利用半合法测试例对软件进行面向安全的测试。通过在测试过程中监视软件对于测试例的响应可以对潜在安全漏洞进行有效地发现和定位。

1.1 早期Fuzzing

2002 年之前，Fuzzing 技术主要利用随机生成的方式产生测试例，之后利用这些畸形测试例对软件进行测试，这种随机生成的方式虽然投入成本低，部署快速，但是盲目性也很大：由于事先未能对测试目标的协议进行建模分析，因此这些畸形测试例不能很好地深入到软件内部，大量代码域不能被有效测试。

1.2 Fuzzing发展

在随后的发展中，Fuzzing 测试技术逐渐向基于生成的和基于变异的两个方向发展。其中基于生成的测试模式主要思

收稿日期：2013-09-25

基金项目：公安部科技强警基础工作专项 [2012GABJC019]、中国人民公安大学基本科研业务费项目 [2013LGX01-1]

作者简介：史记（1990-），男，北京，硕士，主要研究方向：软件安全；曾昭龙（1976-），男，黑龙江，讲师，博士，主要研究方向：安全防范系统集成；杨从保（1976-），男，江西，硕士，主要研究方向：信息安全；李培岳（1988-），男，山东，硕士，主要研究方向：安全防范系统与工程。

想是：首先需要由测试人员对目标测试例的协议结构进行学习和了解，在此基础上构造出基本符合协议要求的测试用例。依据此种思想生成的测试用例能够对目标程序进行更深层的测试。图1展示了一个基于生成的测试模型的主要流程。而基于变异的测试模式则相对容易部署和实施，相比基于生成的测试模型，其最大不同就是模型在初始状态下需要有一定的正常样本，并对这些样本内容进行随机改变。这种模型能否成功的关键在于是否选取了足够多的样本，同时用这种测试方式生成测试例的有效性要略低于基于生成的测试模型。

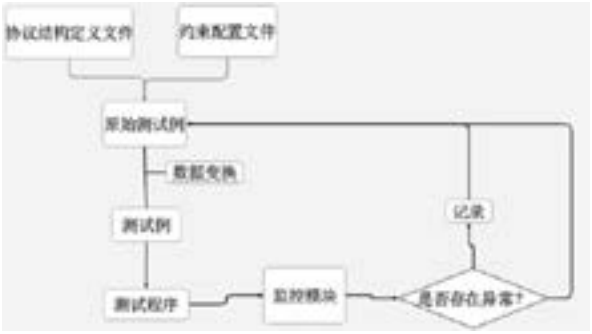

图1 基于生成的测试模型

1.3 Fuzzing研究现状

目前阶段的测试已经发展出了另一种测试模型，这种模型利用人工智能自动地对测试对象所采用的协议进行学习、解析。之后根据学习到的协议知识自动地生成测试例并利用这些测试例对程序进行测试。依据此种模式生成的测试用例可以有效的提高测试过程中的代码覆盖率。但是该模型在协议学习部分仍不是很成熟，现阶段很难对所有的协议进行完整的学习。除此之外，利用基因生成算法、模拟退火算法等模型的Fuzzing测试技术也不断发展。

2 现有 Fuzzing 测试框架

Fuzzing测试领域目前有许多针对不同测试对象的框架出现。接下来对目前已有的测试框架进行介绍，并针对不同的测试对象将它们进行分类。

2.1 面向文件格式

此类测试工具以文件作为测试对象，例如.doc，.xls .pdf等类型的文件。这类测试框架主要测试程序在解析文件时可能出现的问题，例如堆栈溢出，格式化字符串漏洞，这类测试工具有FileFuzz, Peach, SPIKE等。这些框架的优点很明显：它们能够自动化的对目标程序进行检测，并监视这一过程中程序对测试例的反应，记录异常。而缺点是通常情况下测试人员需要事先掌握文件协议，这对于测试未公布的文件协议来说相对困难。其次这些测试框架每次仅对文件中的单个元素进行测试，对于多因素共同引起的问题并不能很好地测试。

2.2 面向浏览器

浏览器可以对html进行动态的解析和渲染。除此之外，脚本，插件等都是主流浏览器所需处理的对象。浏览器所提供的功能越丰富，其在处理过程中出现安全问题的概率就越大。目前此类测试框架有NodeFuzz, AxMan, COMRainder。

2.3 面向网络协议

除了以上提到的针对文件和浏览器的测试框架外，还有一种更为通用的测试框架，其主要面向网络协议。这里虽然提到了网络协议的概念，但测试重点在于解析网络数据包中真正有意义的部分，例如在FTP协议中的USER字段或者PASS字段，对于这些字段的处理是由Server端来进行的，而在解析过程中如果考虑不周就有可能出现缓冲区溢出等问题。这类测试框架包括SPIKE，ProtoFuzz，ircfuzz，dhcpfuzz等。对网络协议测试时需要考虑到发送数据包时的一些限制因素，例如TCP中的SYN以及ACK，而这类信息不是测试框架所要关注的重点，因此往往针对网络协议的测试框架都会采用代理模块：将客户端发送的数据包直接发送至代理处，代理接收到数据包后交付到数据变换模块对有意义的数据进行变换，变换之后重新计算校验值并将数据包发送至Server端。这种处理方式的好处就是不用考虑数据包的限制问题从而使测试过程更为有效。

2.4 面向内核

面向内核的Fuzzing测试的主要思想是：通过给内核驱动输入大量畸形数据，观察操作系统对这些畸形数据处理后的反应，如果出现了蓝屏或者崩溃等信息，则再对dump信息进行静态分析定位漏洞。这类工具如IOCTL等。

3 Fuzzing 测试的缺陷

虽然Fuzzing测试能够自主地对目标程序进行测试，并能够记录错误信息，但是现阶段Fuzzing测试仍存在一些障碍。

首先，自动化测试工作中代码覆盖率较低。根据Fuzzing测试的特点，对于样本例不能达到的代码部分，Fuzzing测试框架是无法对这些代码进行检测的。针对此问题，一些人工智能算法被应用来解决此问题，例如基因生成算法。虽然这类算法在理论上确实能够有效增加测试过程中的代码覆盖率，但是在实际应用过程中是否有效仍值得深入研究。

其次，在基于变异的测试模式下，如何使测试例尽可能多的通过程序对测试例本身的验证。这类验证包括但不限于校验和，哈希，长度，完整性。这类验证通常在网络协议应用、复合文档处理程序中。如果测试例无法通过这类检测，就不能够深入到目标程序内部，无法更有效的检测出更多的安全问题。这种问题在前面所述的测试框架中普遍存在。例如,在测试过程中,测试过程确实持续了很久，但是很有可能这些测试用例仅仅维持在一个代码覆盖率水平上没有突破，这时就需要考虑到用例是否没能通过程序本身对测试例的约束条件而导致效率低下。

再者，由单一因素引起的漏洞越来越少，目前的安全漏洞多数都是由多种因素共同影响而产生的。而传统Fuzzing测试中每次仅对单一元素进行数据变换，这样的后果就是无法发现多因素共同作用下产生的安全漏洞。但是若要进行多因素测试则又会引发测试用例数量爆炸的问题，因此这类由多因素引起的漏洞自动化挖掘也是Fuzzing测试过程中面临的障碍。

最后，目前的Fuzzing测试框架每次仅对一个测试例进行测试，而未能将测试例的测试结果考虑到下一次测试过程中。也就是说每一次测试过程都是独立不相关的，这样的测试方式必然会带来一些弊端。

4 Fuzzing 测试的改进框架

4.1 针对提高代码覆盖率的Fuzzing方式

目前阶段提高代码覆盖率的主要方法是预先对测试对象的协议知识进行人工学习，之后利用学习的协议知识对生成的测试对象进行约束限制，从而满足尽可能多的约束条件，使测试能够更为深入。其中Peach就是这类测试框架中的代表。下面以Peach对Microsoft Office文档进行的测试过程为例进行说明。微软的Office文档是一种复杂的文件格式，其内部文件结构中存放了大量的结构化文件信息以及相应的数据。在一个Office文件中，用户可以嵌入表格、图像、视频等内容，而这些信息经过索引标记后，相应的索引项会被存储到文件中特定的位置。微软在其官方网站已经公开了部分关于Office文档的二进制文件结构信息，依据已有的文档格式，测试人员可以构建Office的文件的Peach Pit并以此进行Fuzzing测试。其测试框架中的工作组件如图2所示。

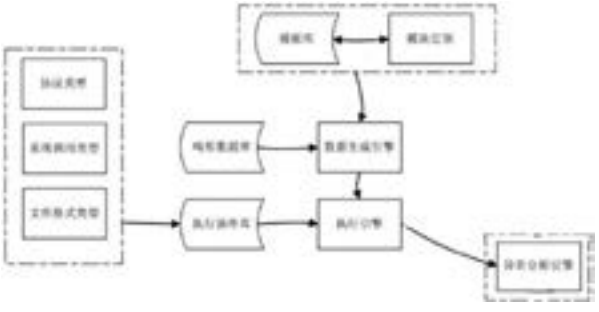

图2 Peach组件

对于一些没有公开协议或文档信息的被测试对象，只能通过逆向工程与Fuzzing技术相结合的方式进行漏洞挖掘。逆向工程（reverse engineering）是根据已有的成型的程序，通过分析推导出源程序的具体实现方法，其主要目的在于对他人的软件进行分析，了解其工作模式，从而按照自己的需要重新设计软件或协议。在这里本文所指的逆向工程是对上述未公开协议或者格式的测试对象进行分析从而获得文件或者协议的具体含义和信息，同时包括程序在处理文档或协议时的处理逻辑。将逆向工程与Fuzzing相结合的一般做法为：首先利用反汇编工具将测试对象进行解析，根据其指令代码进行人工分析，得到程序对文件的处理流程并推测出相对应的文件结构；接下来根据推断的文件结构信息构建Fuzzing所需要的测试例生成规则并将其应用于实际的测试；最后在测试过程中需要不断的观察和完善生成规则以使其更加准确。相比于公开知识的测试过程来说，这种方式需要花费大量的人力对测试对象的协议知识进行分析。

Peach测试框架采用了当前主要的两种数据生成方式：基于数据变异的生成方式和基于数据生成的方式。其中基于生成的Fuzzing方式类似于上述事前有一定的关于测试对象的结构知识，并以这些知识为基础生成测试例；而基于变异的测试方式则是在事先有一定数量样本的前提下对

这些样本进行随机变异后进行测试。这两种方式都各有优缺点。基于生成的测试方式能够有效地提高代码覆盖率，达到更好的测试效果，但是事先需要人工对测试对象的协议或结构进行学习。基于变异的测试方式则不需要对测试对象的结构进行学习，只需要找到一定数量的初始样本便可以进行 Fuzzing，但是所能达到的代码覆盖率取决于初始样本的种类和数量。Peach 将二者有机的结合，即将测试对象的知识形成相应的 Peach Pit, 之后 Peach 会根据 Pit 文件生成测试例并以此为基础对测试例进行变异，变异的约束则是根据协议或文件结构知识实现。因此只要对相应的 Pit 文件设置合理，Peach 能够大幅度提高测试过程中的代码覆盖率。

4.2 针对提高约束通过率的Fuzzing方式

为了在 Fuzzing 测试过程中尽可能有效地解决约束问题，一些研究者将符号化执行应用到其中，这种方式在进行约束求解等方面有着很好的应用。符号化执行的主要思想是利用符号值代替传统的数据作为输入，之后在程序的执行过程中记录路径的约束条件，最后解出符号的具体数值，此数值便会作为一个执行到具体路径的“期待值”。因为符号化执行是在未知协议知识的前提下求解出期待值，因此这种约束求解的方式也可以看作是 4.1 节中的基于数据生成的方式的一种。

符号化执行的实现思路为：利用一个原始的数据作为程序初始运行的输入，在这里数据即输入文件或者数据包，之后利用符号化执行，这些输入数据被符号化成为变量。当符号化执行完成后，执行路径中的数据信息和约束被信息表示出来，再将各个路径分支部分的约束提取并输入到约束求解器当中进行求解。求解之后便可以动态生成所需要的测试例并能够引导程序执行到指定位置。

尽管符号化执行能够在一定程度上解决约束问题，并且能够结合诸多优势，但是其自身也存在一定的局限性，并且这种限制也给 Fuzzing 测试带来了不小的困难，影响到基于符号化执行的 Fuzzing 测试技术在实际测试过程中的应用。以下是两个主要的缺陷：

1）求解复杂：尤其是在动态执行时的数据信息和约束信息收集过程中需要对目标程序进行二进制级别的分析，插桩，约束生成和约束求解。由于这些工作本身具有很大的难度和复杂度。这些影响使得其在实际应用中遇到很大困难。

2）计算资源需求大：动态符号化执行过程中，指令级的插桩和分析以及约束路径的生成都需要大量的资源，而当程序规模增加时，这种资源消耗更为严重。

4.3 针对多因素的Fuzzing方式

传统 Fuzzing 框架仅对测试对象中的单一元素进行变异测试，这种方式不能够发现由多因素共同作用引起的漏洞。因此有研究者提出了基于多维的 Fuzzing 测试模式，所谓多维测试即在一次测试过程中同时对多个因素进行变异。

例如，文献 [3] 将多维 Fuzzing 技术实现过程中采用的基本步骤归结为：1）通过静态分析技术定位目标程序中的脆弱语句的位置；2）通过动态分析技术找到输入元素和潜在脆弱语句之间的影响映射关系；3）针对每个潜在脆弱点同时变异影响它的所有输入元素不断生成新的测试用例以触发潜在漏洞。

4.4 智能Fuzzing

除了上述提到的 Fuzzing 测试框架，还有研究者将人工智能算法与 Fuzzing 测试相结合，如模拟退火算法，基因生成算法等。下面以基因生成算法为例进行介绍，基因生成算法的主要目的在于解决测试过程中仅对单一文件进行变异而未将测试例集合进行有机融合，不考虑测试过程中反馈的问题。

首先介绍该测试方式的一些基本概念。初始种群：第一轮测试时所用到的测试例集合；基因：测试例的基本特征，如 Office 文档中的属性字段、属性值等；适应度：本意是指个体在群体中的适应情况的度量，在 Fuzzing 测试中特指单个测试例所能达到的代码覆盖率。适应度越高的测试例说明测试效果越好，能够达到更高的代码覆盖率，在此基础上进行的变异、交叉等操作会带来更好的结果。

基于基因算法的 Fuzzing 测试基本思想为：首先对初始测试例集合进行测试，并计算每个测试例在测试过程中的适应度，将高适应度的测试例保留，低适应度的测试例舍弃。之后将高适应度的测试例进行相互组合、变异，得到下一轮新的测试例，如此往复直到适应度达到一个稳定

的水平，则可以判断测试例种群达到收敛状态。这种思想的好处是可以动态地得到测试例的反馈并将此反馈应用于下一轮的测试过程中。其工作模型如图 3 所示：

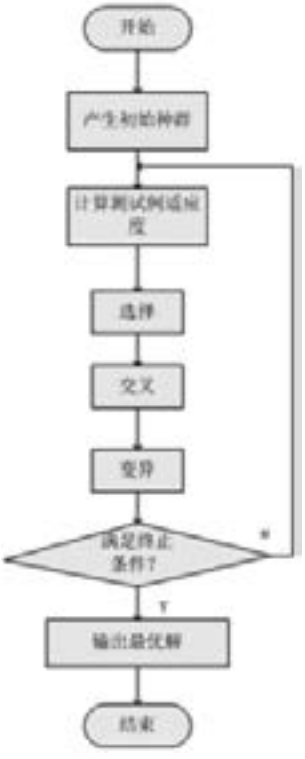

图3 基因算法工作模型

5 结束语

根据之前所总结的目前阶段 Fuzzing 测试所存在的问题和缺陷，预测未来 Fuzzing 会向以下方面进行发展：

1）智能化 Fuzzing。人工智能算法将会得到更深入的研究和发展以帮助测试工具能够更加智能地对目标程序进行测试。

2）协议自学习。协议自学习可以帮助测试工具对被测试程序所使用的协议进行了解，提高生成测试例的半有效性，从而突破程序对样本的约束验证。现有的对未知协议进行自学习的方法仍不成熟，未来此类协议自学习也将是发展的一大方向。

3）缺陷定位自动化。Fuzzing 测试工具在发现程序崩溃后仅仅是安全漏洞发现的第一步，往往后期需要大量的人工分析去判定造成崩溃的测试例是否会构成安全问题。而 Fuzzing 测试仅仅会记录整个测试例的完整副本，如果能够具体表明出哪些元素造成的最后崩溃，则会给后期分析带来很大便利，减少了人工分析的工作量同时也提高了整体的挖掘效率。

4）代码覆盖率进一步提升。代码覆盖率是 Fuzzing 测试过程中不得不提到的一个关键问题，目前的测试框架在提高代码覆盖率方面仍有可以提升的地方，目前的人工智能算法在实验室应用时的确初有成效，但是在实际应用过程中仍会有许多问题，这也是 Fuzzing 测试未来发展的方向。（责编 吴晶）

参考文献

[1] 于璐，沈毅 . Fuzzing 测试中样本优化算法的分析与改进 [J]. 计算机安全 , 2011，(04): 17-20.

[2] 吴志勇 , 夏建军 , 孙乐昌，张旻 . 多维 fuzzing 技术综述 [J]. 计算机应用研究 , 2010,(08): 2810-2813.

[3] 王丹 , 高丰 , 朱鲁华 . 基于遗传算法的 fuzzing 测试用例生成模型 [J]. 微电子学与计算机 , 2011,(05): 130-134,139.

[4] 高峻 , 徐志大 , 李健 . 针对复合文档的 fuzzing 测试技术 [J]. 计算机与数字工程 ,2008,(12): 116-119.

[5] 黄奕 . 基于模糊测试的软件安全漏洞发掘技术研究 [D]. 合肥：中国科学技术大学 , 2010.

[6] Kim, Hyoung Chun, Young Han Choi and Dong Hoon Lee. Efficient File Fuzz Testing Using Automated Analysis of Binary File Format[J]. Journal of Systems Architecture,2011(3): 259-268.

[7] 刘驰 . 基于协议分析的漏洞挖掘技术研究 [D]. 北京：北京邮电大学 ,2010.

[8] Sutton, Michael and Adam Greene. The Art of File Format Fuzzing[C]. Blackhat USA Conference, 2005.

[9] 岳彩松 , 李建华，银鹰 . 基于 fuzz 的 ms Office 漏洞检测 [J]. 信息安全与通信保密 ,2007,(09): 111-113.

[10] 梁晓兵 . 面向二进制程序漏洞挖掘的相关技术研究 [D]. 北京：北京邮电大学，2012.

[11] 张翀斌 . 基于模型检测技术的软件漏洞挖掘方法研究 [D]. 济南：山东大学，2006.

[12] Luo, Cheng, Yuqing Zhang, Long Wang and Qixu Liu. Automatic Network Protocol Analysis and Vulnerability Discovery Based on Symbolic Expression[J]. Journal of the Graduate School of the Academy of Sciences, 2013,(2): 278-284.

[13] Hwang, Seong Oun. Finding Vulnerabilities in Binary Codes Using Tainting/Fuzzing Analysis[J]. In Convergence and Hybrid Information Technology, edited by G. Lee, D. Howard, D. Slezak and Y. S. Hong, 310, 2012(277-286).

[14] Zhang, Dazhi, Donggang Liu, Yu Lei, David Kung, Christoph Csallner, Nathaniel Nystrom and Wenhua Wang. Simfuzz: Test Case Similarity Directed Deep Fuzzing[J]. Journal of Systems and Software, 2012,(1): 102-111.

[15] 杨俊 . 基于函数摘要的二进制漏洞挖掘技术研究 [D]. 合肥：中国科学技术大学 ,2011.