Fuzzing 模糊测试技术研究报告

摘要：目前通过测试发现软件漏洞的最有效方法之一就是模糊测试（Fuzzing），Fuzzing 是一种通过向目标系统提供非预期的输入并监视异常结果来发现软件漏洞的方法，可用于测试软件抵抗漏洞的能力。本文具体介绍了模糊测试的概念、历史进程、主要步骤及其分类、目标、应用等，并对其做出评价与展望。

关键词：模糊测试；软件安全

1 概述

许多恶意攻击都基于漏洞的存在。当这些漏洞被利用时，安全缺陷就允许攻击者闯入系统。因此，识别漏洞是至关重要的。目前可以使用多种解决方案来发掘此类漏洞，其中一个重要的技术就是模糊测试。模糊测试是一种接口测试方法，旨在检测软件中的漏洞，而无需访问应用程序源代码。尽管近年来许多学术和应用研究已经解决了许多软件安全问题，但每个月都有数百个新的漏洞被发现或利用。企业和机构基础设施、系统和网络中存在的此类威胁可能会影响其整个企业信息技术领域。

模糊测试的概念最早在 1989 年由威斯康星州麦迪逊大学的 Barton Miller 教授提出。由于目前人们聚焦于开发更加安全、可靠的软件，模糊测试才被更加广泛应用并成为公认的代码测试方法。

已有文献中有许多关于模糊测试的定义。所有这些都可以概括为一个定义：模糊测试是一种基于向程序注入无效或随机输入的数据的安全测试方法，以此发现意外行为并识别错误和潜在的漏洞。由于模糊测试没有精确的规则，所以测试方法并不单一。它的效率取决于作者的创造力。它的关键思想是生成能使目标崩溃的相关测试，并选择最合适的工具来监视该过程。用于模糊测试的数据生成主要有两种方式。第一种方法是通过修改正确数据来随机生成输入，而不需要了解应用程序的详细信息，这种方法被称为黑盒模糊测试。另一种是白盒模糊测试，在生成测试时假设对应用程序代码和行为有完整的了解。还有一种是灰盒模糊测试，它介于两种方法之间，旨在兼具两者的优点。它只有对目标程序最低限度的了解。

近几年，模糊测试通常用于加强软件的安全性，因为这一技术可以检测出软件测试人员很难通过人工检查发现的一些奇怪的问题和缺陷。需要注意的是，这一测试技术并不是穷举测试或规范化的测试，它也只是采用随机数来覆盖部分系统行为的测试，只能发现涵盖范围内不能正确执行的系统功能，并不能证明系统行为的正确性和合理性。因此它只能作为有限范围内发现问题的工具，而不能成为全面的代码质量保证工具。在软件测试过程中，一个程序被随机产生的数据大量验证，称为模糊测试。

假如一个程序在应对任一类数据时崩溃，如程序运行时，一旦系统开始出现系统资源冲突、死锁、消耗大量内存或者产生不可控制的程序错误甚至死机、宕机。开发人员就有可能发现并定位代码中有些地方或环节出现了问题或缺陷，这样的话，就可以帮助开发人员在程序发布前定位相关缺陷并予以修复，从而使可能出现的漏洞不出现在程序最终发行版本中。而开发人员在软件开发过程中也总是想方没法使用模糊测试来发现并定位解决相关问题。

通常在进行模糊测试前，开发人员要根据程序架构和数据处理流程。设计测试用的测试数据，并对测试数据进行记录，一旦软件系统进行动态测试过程中引发系统崩溃，这些引发崩溃或错误的测试数据及其产生随机数的种子可以有效保留下来，以便重复使用，从而定位

问题。

总而言之，模糊测试是通过使用修改或模糊化的输入反复测试代码来确定输入处理代码中安全漏洞的过程，是发现软件安全漏洞的有效方法，并正在成为商业软件开发过程中的标准。现有的模糊测试工具因其数据输入的方式不同而有所不同，但在实际应用中，它们无法发现实际的程序中更多的冗余空间。因此，他们通常使用模糊启发式方法来区分下一个模糊输入的优先级。这种启发式方法可能是纯随机的，或可能针对特定的目标进行优化，例如最大化代码覆盖率。模糊测试是一项用于验证程序中真实错误的重要工具，也是所有意识到安全性问题且着力于程序健壮性的程序员们必备的工具。

2 历史进程

随机输入的测试程序可以追溯到 20 世纪 50 年代，当时数据仍然存储在穿孔卡片上。程序员将使用取出的穿孔卡片或随机数卡片组作为计算机程序的输入。如果执行过程中发现了不希望出现的行为，则会报告一个错误并加以修复。

随机输入的执行也称为随机测试或猴子测试。

1981 年，Duran 和 Ntafos 正式调查了使用随机输入对程序进行测试的有效性，虽然随机测试被广泛认为是测试程序的最差方法，但作者可以证明，它是比更系统的测试技术更具成本效益的替代方法。

1983 年，Steve Capps 开发了“The Monkey”，这是一个可以为经典的 Mac OS 应用程序（如 Mac paint）生成随机输入的工具。这里的的“Monkey”是指无限猴子定理，即若一只猴子在打字机键盘上随机敲击按键的时间无限长，它最终可以打出莎士比亚的所有作品。测试中，猴子会编写特定的输入序列，从而触发程序崩溃。

术语“Fuzzing”源于 1988 年威斯康星大学的 Barton Miller 教授的一个课堂项目：模糊测试一个 Unix 实用程序，用于自动生成该实用程序的随机文件和命令行参数。该项目旨在测试 Unix 程序的可靠性，通过快速连续地执行大量随机输入，直到它们崩溃。它还提供了早期调试工具，以确定每个检测到的故障的原因和类别。为了让其他研究人员能够使用其他软件进行类似的实验，工具的源代码、测试程序和原始结果数据都被公开了出来。

1991 年，“CrashMe”工具发布，旨在通过执行随机机器指令来测试 Unix 和类 Unix 操作系统的健壮性。

1995 年，模糊测试器（Fuzzer）被用来测试基于 GUI 的工具（如 X Windows 系统）、网络协议和系统库 API 等。

2012 年 4 月，谷歌发布了 ClusterFuzz，一个基于云技术的针对 Chromium 浏览器关键安全组件的模糊测试基础设施。如果 ClusterFuzz 发现上传的 fuzzer 崩溃，安全研究人员可以上传他们自己的 fuzzer 并收集错误信息。

2014 年 9 月，ShellShock 作为广泛使用的 Unix bash shell 中的一系列安全漏洞被披露；ShellShock 的大多数漏洞都是使用 Fuzzer AFL 发现的。（许多面向 Internet 的服务会使用 bash 处理某些请求，例如一些 Web 服务器的部署。这允许攻击者控制脆弱的 bash 版本执行任意命令。攻击者就此可获得对计算机系统的未经授权的访问权限。

2015 年 4 月，Hanno Böck 展示了 Fuzzer AFL 是如何发现 2014 年的 Heartbleed vulnerability 漏洞的（2014 年 4 月 Heartbleed vulnerability 被披露，这是一个严重的漏洞，允许攻击者破译加密通信。）该漏洞被意外引入 OpenSSL，OpenSSL 实现了 TLS，并被 Internet 上的大多数服务器使用。Shodan 报告 2016 年 4 月仍有 23.8 万台机器处于脆弱状态。（2017 年 1 月为 20 万台）

2016 年 8 月，美国国防高级研究计划局（DARPA）举行了第一次网络挑战的决赛，这

是一次持续 11 小时的全自动 CTF 竞赛。目标是开发能够实时发现、利用和纠正软件缺陷的自动防御系统。模糊测试技术作为一种有效的进攻策略，被用于发现对手软件的缺陷。它在漏洞检测自动化方面显示出巨大的潜力。获胜者是由 David Brumley 领导的安全小组开发的一个名为“Mayhem”的系统。

2016 年 9 月，微软发布了 Springfield 项目，这是一个基于云的模糊测试服务，用于在软件中发现关键的安全错误。

2016 年 12 月，谷歌发布了 OSS-Fuzz，允许对几个关键的安全开源项目进行连续模糊处理。

在 2018 年的黑帽子大会上，Christopher Domas 演示了使用模糊测试技术来暴露处理器中隐藏的 RISC 核心的存在的过程。该核心能够绕过现有的安全检查，通过 RING 3 来执行 RING 0 命令。

3 主要步骤

一般步骤是，首先在程序集级别定义漏洞模式。这些模式有助于识别二进制文件中的函数。考虑到上一步获得的信息，将对二进制文件进行污染数据分析。潜在危险的数据将被标记为污染数据，并在执行时跟踪其传播。然后，将使用混合执行和搜索算法进行测试生成，以便审计测试中应用程序的最危险路径，并为下述 3.3 节所述的每个新执行生成更好的质量测试。生成和执行测试后，将通过覆盖率评估模糊测试的效果。分析技术将在执行时实施并使用检测故障的方法。将评估每个检测到的漏洞的潜在可利用性。

3.1 漏洞模式识别

第一步是识别二进制文件中潜在的易受攻击的代码序列，以便了解二进制文件中应该测试的最危险部分。其理念是基于已经发现的漏洞和 Vupen 安全专业知识来定义漏洞“模式”，以发现和利用二进制文件中的安全漏洞。漏洞模式表示程序集级别的模型，该模型可能是导致尤其是内存损坏类错误的原因。源代码中易受攻击的函数的一个例子是 strcmp，其中函数参数可以由用户控制。这种常见情况允许以一种不安全的方式访问内存，这种方式可能会导致缓冲区溢出。

3.2 污染数据分析

过去几年中，污染数据的分析方法发展势头有所提高。这一点的一个佐证是它使用次数的不断增加和许多框架的出现。该功能包括将来自不可信来源的数据（如网络和用户输入）标记为污染数据。例如，它可以用于跟踪受污染数据的传播，并检查何时以危险的方式使用受污染数据，例如覆盖返回地址。

数据污染[4]将通过使用有关漏洞的信息（如路径执行）来改进模糊化，以选择最有希望触发潜在漏洞并限制测试空间的测试序列。

许多污染数据分析工具已被发明。例如，Dytan 是一个针对 Windows 和 Linux 的动态污染分析框架，它不需要重新编译或访问源代码就可以检测二进制文件。MyNav 是 IDA Pro 反汇编程序的一个插件，它是为帮助工程师完成任务而创建的，有助于动态分析程序，虽然它不是一个污染分析框架，但在一些情况下可以使用。它的功能包括找到关键的函数和输入之间的路径。

3.3 数据生成

输入数据的生成是模糊测试过程中最重要的一步。此步骤可以以不同的方式执行。输入数据可以通过对现有数据进行变换或对目标（文件处理器/协议）建模来生成。可以使用源代码、学习方法或从目标规范创建模型。

为了触发可能出现故障的特定路径，测试值的选择至关重要。我们的目标是生成相关性最高的测试用例。为此，可以使用混合执行和搜索算法。混合执行是一种基于以具体值为输入执行程序的系统路径探测技术，收集这些输入的约束条件，求反并求解约束条件，以探索新的路径条件。该勘探策略已在多个模糊测试工具中实现，如微软的 P.Godefoird 开发的 SAGE 和 Fuzzgrid。还需要使用诸如 PIN、Valgrind 和 Dynamorio 等仪器工具来生成约束。生成约束后，必须使用约束解算器（如 STP）求解约束。混合执行允许我们通过生成新的输入来探索新的执行路径。

这里可以使用搜索算法，如遗传算法。每次执行之后测试质量都会得到提高，这要归功于测试群体的进化，即对初始群体应用操作（变异、交叉等）。

污染数据分析可应用于程序集级别，以收集潜在危险数据的信息；然后使用具体的值执行程序，以生成探索新路径的输入，并选择能够触发潜在漏洞的最有希望的测试序列，同时考虑到在前一版本中收集的信息。在第一次执行之后，我们将搜索算法（如遗传算法）应用于先前生成的输入群体，以改进测试生成过程，这得益于交叉和突变操作，当这些被应用于测试群体时，将会为下一次执行生成更好的质量测试。

3.4 代码覆盖率分析

一旦产生了模糊测试的输入数据并使用这些新输入执行目标，就必须使用代码覆盖技术评估模糊测试过程的有效性。模糊评估是衡量程序测试的好坏并确定扩大覆盖范围所需的修改，其思想是实现一种技术，以便跟踪二进制的执行。它包括使用 Pydbg 跟踪基本块的执行，在每个基本块中设置断点，并检查是否命中，以评估覆盖率。IDA2SQL 通过将程序（基本块、函数…）的信息导出到 MySQL 数据库来帮助完成此任务。

预先估计模糊测试的效率有助于最大限度地发挥它的作用。

3.5 异常监控

在模糊测试过程中，需要监控程序执行时可能出现的故障。如果这些特性在执行时被附加到程序上，那么分解和调试特性有助于这种监视。

这里不仅建议监控异常情况，还建议在运行时系统地识别违规情况，以避免隐藏的漏洞。以这种方式检测故障的发生可以提高故障检测能力。

由于程序集的复杂性，在程序集级别工作会使此任务更加困难。

3.6 漏洞分析

如果在最后一步中发现了故障，则必须对其执行一个漏洞分析以确定其潜在的可利用性。当漏洞被触发时，分析从二进制文件（如堆栈或堆内容）收集的信息将是评估其潜在可利用性的一个关键方向。目前有多种工具，例如 Miller 的二进制分析工具 BitBlaze，可帮助确定崩溃是否由潜在的可利用漏洞引起，但是这些工具不能提供完全可靠的信息。

4 模糊测试分类

目前使用的模糊测试技术主要有三种类型:

（1）黑盒模糊测试

（2）白盒模糊测试

（3）基于语法的模糊测试

其中黑盒和白盒模糊测试是全自动化的,历史证明这两种方法在查找二进制文件解析器中的安全漏洞方面非常有效。与之相对的是，基于语法的模糊测试并不完全自动化，它需要一个输入语法来测试应用程序的输入格式。这种语法通常是人工编写的,整个过程非常费力、耗时且容易出错。然而，基于语法的模糊测试是目前已知的最有效的模糊测试技术，可用于测试有复杂的结构化输入格式的应用程序。

微软给出的模糊测试分类及其应用场景如下:

类型	情况描述
Dumb fuzzing	数据结构未知的情况下数据包损坏
Smart fuzzing	数据结构已知情况下的数据包损坏，例如编码（base-64 编码等）和相关信息（校验和、表示某些字段存在的位、表示偏移量或其他字段长度的字段）。
黑盒模糊测试	在未实际验证哪些代码路径被命中，哪些没有的情况下发送格式错误的数据。
白盒模糊测试	在已验证所有目标代码路径都被命中的情况下，发送格式错误的数据，修改软件配置和模糊测试数据以遍历测试代码中的所有待验证数据。
基于生成的模糊测试（Generation）	自动生成模糊测试数据，不基于任何先前的输入。
基于变异的模糊测试（Mutation）	根据缺陷模式破坏有效数据，产生模糊测试的输入。
变换模板（Mutation template）	代表输入的等价类的格式良好的缓冲区。模糊测试器以突变模板作为输入，产生一个模糊缓冲区，发送给测试软件。
代码覆盖技术（Code coverage）	指允许检查测试期间执行了哪些代码路径的技术（如 Microsoft Visual Studio 2005 中应用的技术）。这对于验证测试有效性和提高测试覆盖率很有用。

5 模糊测试目标

5.1 边界输入可信

模糊测试是一种使边界输入可信的验证方式，其中典型的待测边界有:

- 从网络接收的文件
- 网络套接字（socket）
- 管道（pipes）
- RPC 接口
- 驱动器 IOCTL
- ActiveX 对象

模糊测试也可以用于其它不太典型的待测边界输入

- 在数据库中存储为 blob 的结构化数据（例如 xml）且能被不同用户读写的
- 一个用户写的被另一个用户读的配置文件
- 消息队列系统（持续发送消息的数据库或 Windows 消息）
- 不同进程间传递结构化信息的共享内存

5.2 模糊测试的主要和次要目标

在典型的组件化软件设计中，解析输入数据的代码将结果放在内部数据结构或类中，并与随后使用这些结构的逻辑完全分离。解析代码是模糊测试的主要目标；对其中的数据结构的任何未经验证的假设都会导致意外的结果。但是，进一步使用已解析数据的代码也可能对已解析的数据有假设，对该辅助代码应用模糊测试也可能会发现安全缺陷。

因此，在模板选择和代码覆盖率方面，也应该考虑辅助代码的分析。

5.3 托管代码的模糊测试

模糊测试可以找到的大部分缺陷发生在使用 C 语言或 C++语言开发的软件中，这些源于程序员留下的内存管理问题。而模糊测试可以发现隐藏在内存管理语言中的缺陷，包括 C 语言、Visual Basic 和 Java 等。在托管代码中可能发现的错误有未处理的异常、死锁或内存达到峰值。通常这些错误是拒绝服务（DoS）或信息公开类的错误。

6 模糊测试应用

模糊测试主要是作为一种自动化技术，用于暴露安全关键程序中可能会被利用的漏洞，模糊测试用来证明漏洞的存在，而非证明不存在。长时间运行模糊测试而没有发现错误并不能证明程序是正确的。毕竟对于尚未执行的输入，程序仍然可能失败；而对所有输入执行程序的成本是令人望而却步的。如果测试目的是证明一个程序对所有输入都是正确的，那么必须存在一个正式的规范，并且必须使用来自正式方法的技术。

6.1 发现漏洞

为了暴露错误，模糊器必须能够区分预期（正常）和意外（错误）程序行为。然而，机器无法每次都区分开一个缺陷和一个特性。在自动化软件测试中，这也被称为测试 Oracle 问题。

通常，模糊测试在没有规范的情况下被区分成碰撞输入和非碰撞输入，并使用简单客观的测量方法。程序的崩溃很容易被识别，并可能指示潜在的漏洞（例如，拒绝服务或任意代

码执行)。但是，程序未崩溃并不意味着没有漏洞。例如，当输入导致缓冲区溢出时，用 C 编写的程序可能会崩溃，也可能不会崩溃。程序的行为是未定义的。

为了使模糊器对崩溃以外的故障更为敏感，可以使用漏洞注入在检测到故障时使程序崩溃的断言。对于不同类型的错误，有不同的漏洞可供注入为断言：

- 检测与内存相关的错误，如缓冲区溢出，并在释放后使用（使用内存调试程序，如 AddressSanitizer)。
- 检测竞争条件和死锁（ThreadSanitizer)，
- 检测未定义的行为（未定义的 BehaviorSanitizer)，
- 检测内存溢出（LeakSanitizer)，或
- 检查控制流完整性（CFISanitizer)。

如果参考实现可用，模糊测试还可以用来检测“差异”错误。对于自动回归测试，生成的输入在同一程序的两个版本上执行。对于自动差异测试，生成的输入在同一程序的两个实现上执行（例如，lighttpd 和 httpd 都是 Web 服务器的实现)。如果两个变量在相同的输入上产生不同的输出，那么其中一个可能有问题，应该对其进行更仔细地检查。

6.2 验证静态分析的结果

静态程序分析允许在不实际执行程序的情况下分析程序。这可能会导致误报，因为工具可能报告实际不存在的程序问题。模糊测试与动态程序分析相结合，可用于生成一个实际执行情况下的输入数据。

6.3 浏览器安全

现代网络浏览器经历了广泛的模糊测试。谷歌 Chrome 的 Chrome 源代码还在不断被拥有 15000 核的 Chrome 安全团队进行模糊测试。对于 Microsoft Edge 和 Internet Explorer，微软在产品开发期间用 670 台机器进行了模糊测试，从 10 亿个 HTML 文件中生成了 4000 多亿个 DOM 操作。

7 评价与发展

只有安全的设计和实现网络协议才能保护用户传输的敏感信息，但如何测试网络协议安全性一直是个非常困难的问题。特别是测试未公开网络协议的安全性对于网络安全有着重要的意义，因为大量厂商并没有公开自己软件系统。且随着网络应用越来越复杂和重要，对网络协议的安全性要求也越来越高。

模糊测试作为一种重要的测试手段，通过大量数据的注入来测试网络协议的安全，能够发现拒绝服务、缓冲区溢出和格式化字符串等多种重要漏洞。其优点如下

- 测试设计非常简单，且没有关于系统行为的先入为主的观念。
- 快速、准确。随机输入的方法使其可找到人工经常忽略的错误。
- 当被测系统完全关闭时（如一部 SIP 电话)，模糊测试是评估其质量的手段之一。
- 误报率低。通过模糊测试技术发现的漏洞一般都是实际存在的。
- 无需预先了解网络协议的语法。可节省大量的手工分析时间，因而在网络协议安全性检测、网络风险分析、模拟网络服务的蜜罐系统方面具有重要的应用价值。

可见相比其它漏洞挖掘方法，模糊测试有着巨大的优势。然而，其不足之处也十分明显。

目前模糊测试技术仍然存在许多局限性。如

- 对访问控制漏洞无能为力。
- 无法发现程序中糟糕的设计逻辑。
- 无法识别多阶段的安全漏洞。
- 无法识别多点触发的安全漏洞。
- 手工进行模糊测试需要精确了解网络协议细节并需要繁琐的工作来构成大量测试数据集，导致覆盖率有限，效果也不好。
- 典型的模糊测试不能保证输入数据的语法和语义有效性，也不能保证将覆盖多少输入空间。

这里提出一些未来的研究方向供参考：

- 增大模糊测试输入数据的覆盖范围，同时提高测试用例的代码覆盖率。
- 协议自动化分析，实现基于协议生成或变异样本，产生具有很好覆盖率的测试用例。弥补需要在测试前进行大量协议分析带来的时耗问题
- 实现故障的检测与定位，提高错误定位的准确度。
- 如何有效地绕过目标程序对样本数据进行验证。
- 设计更加高效、通用性好的测试平台，使其具有自主的协议分析能力、代码监控能力，获取测试用例执行信息的能力，同时还需有高效的算法来协调各组件的运行，提高整体测试效率。

8 参考文献

[1] Huning Dai, Christian Murphy, Gail Kaiser. Configuration Fuzzing for Software Vulnerability Detection. 2010 International Conference on Availability, Reliability and Security.

[2] Konstantin Bottinger, Patrice Godefroid , Rishabh Singh. Deep Reinforcement Fuzzing. 2018 IEEE Symposium on Security and Privacy Workshops.

[3] Sofia Bekrar, Chaouki Bekrar, Roland Groz, Laurent Mounier. Finding Software Vulnerabilities by Smart Fuzzing. 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation.

[4] Fuzzing-Wikipedia. https://en.wikipedia.org/wiki/Fuzzing

[5] Automated Penetration Testing with White-Box Fuzzing. https://docs.microsoft.com/en-us/previous-versions/software-testing/cc162782(v=msdn.10)

[6] 邓承志，朱卫东. 浅谈代码安全质量保障中的模糊测试技术. 实践探讨. 2009. 2

[7] 李伟明，张爱芳，刘建财，李之棠. 网络协议的自动化模糊测试漏洞挖掘方法. 计算机学报. 2011. 2

[8] 张雄，李舟军. 模糊测试技术研究综述[J]. 计算机科学, 2016, 20(05): 1-4 .