模糊测试原理（学习笔记）

目录

- 0x01 什么是模糊测试
- 0x02 基本原理和组成
  - 1.基本原理
    - 基本思想
    - 基本概念
  - 2.系统组成
    - 值得一提：有关状态监控模块的处理
- 0x03 基础方法技术
  - 数据生成方法
    - 1.基本类型数据生成方法
    - 2.复合类型数据生成方法
    - 3.多阶段交互类型数据生成方法
  - 环境控制技术
    - 1.运行环境控制技术
    - 2.程序运行控制技术
    - 3.数据强制输入技术
  - 状态监控技术
- 0x04 模糊测试优化方法
- 0x05 工具推荐
- 0x06 尾声

原文:https://mp.weixin.qq.com/s/o_BHiyPg8FPDKMMIUZrHsQ

0x01 什么是模糊测试

模糊测试（Fuzz Testing）是一种广泛用于软件安全和质量测试的自动化测试方法。它的基本思想是向输入参数或数据中注入随机、不规则或异常的数据，以检测目标程序或系统在处理不合法、不正常或边缘情况下的行为。模糊测试通常用于寻找软件漏洞、安全漏洞和崩溃点，以改进软件的稳定性和安全性。

0x02 基本原理和组成

1.基本原理

基本思想

模糊测试的思想是构造所有可能的输入，并将输入传递给被测目标程序，然后监控目标程序在接收输入后是否出现异常情况，以此来发现软件中存在的缺陷和故障。

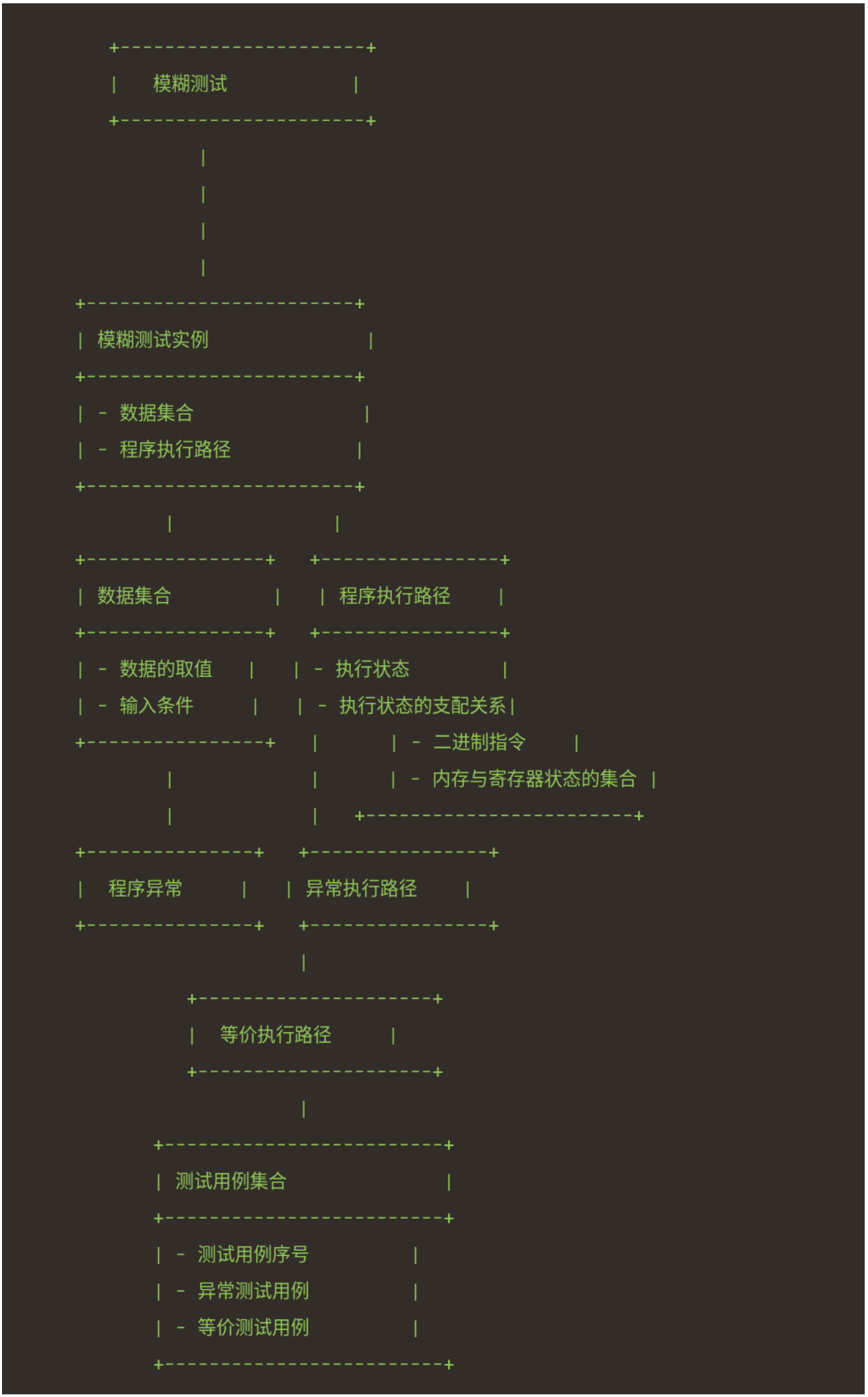
`构造输入 -> 输入 -> 监控状态 -> 判断异常 -> 报告`

抓住了大程序开发的痛点，一定程度上提升了安全测试的效率。

基本概念

> 其中内容加粗体的属于概念。在此我用简单的话语诠释其中的关系。

> 给一个简单的图阐释关系，可以配合文字理解。


模糊测试
模糊测试实例
- 数据集合
- 程序执行路径
数据集合
- 数据的取值
- 输入条件
程序执行路径
- 执行状态
- 执行状态的支配关系
- 二进制指令
- 内存与寄存器状态的集合
程序异常
异常执行路径
等价执行路径
测试用例集合
- 测试用例序号
- 异常测试用例
- 等价测试用例

模糊测试通常包括**模糊测试实例**，而模糊测试实例包括了两个重要元素：**数据集合**和**程序执行路径**。

数据集合即输入数据的集合，**程序执行路径**指的是程序指令序列和对应的内存和寄存器状态。

我们再分开谈谈数据集合和程序执行路径。

数据集合的元素由数据组成，数据包括**数据的取值**和**输入条件**（网络输出或窗口输出）。

程序执行路径由**执行状态**和**执行状态的支配关系**组成。**执行状态**包括了**二进制指令**（比如此时是 `mov rax,1` ）和**内存与寄存器状态的集合**（比如此时 `rbx = 0, rcx = 1` ……内存中 `0x404080 = '\x90'` ）。支配关系指的是下一个到哪里，条件判断等等。

套娃结束了，接下来就是程序异常、异常执行路径，等价执行路径，测试用例集合，测试用例序号，异常测试用例，等价测试用例。

- **程序异常**：程序异常是在程序执行过程中出现的不正常行为，通常导致程序崩溃、错误或漏洞暴露。模糊测试的目标之一是发现程序异常。
- **异常执行路径**：异常执行路径是程序在异常情况下的执行路径，它包括了导致异常的条件和操作序列。模糊测试可以通过检测异常执行路径来发现潜在的漏洞。
- **等价执行路径**：等价执行路径是指在不同输入情况下，程序可能会采取相似或等效的执行路径。理解等价执行路径有助于减少测试用例的冗余。
- **测试用例集合**：测试用例集合包含了多个测试用例，每个测试用例都是一个输入数据和对应的测试条件的组合。模糊测试通常会生成大量的测试用例，以检测各种可能的情况。
- **测试用例序号**：测试用例序号是测试用例在测试用例集合中的唯一标识，用于跟踪和管理测试进度和结果。

1. **异常测试用例（Exception Test Case）**：

    - 异常测试用例是用于测试软件系统如何处理异常情况的测试实例。
    - 这些异常情况可能包括输入无效数据、越界访问、不合法的操作、资源不足等。
    - 目标是验证软件是否能够优雅地处理这些异常，而不会导致崩溃或不正常行为。
    - 例如，一个异常测试用例可能是尝试输入一个非数字字符到一个要求输入数字的字段中，以测试程序是否能够捕获和处理这种非法输入。

2. **等价测试用例（Equivalence Test Case）**：

    - 等价测试用例是用于测试软件系统在不同输入情况下是否表现相似或等效的测试实例。
    - 这些测试用例通常分为等效类别，每个等效类别代表一组具有相似特征的输入数据。
    - 目标是在每个等效类别中选择一个或多个测试用例，以代表该类别的所有可能输入情况。
    - 通过这种方式，测试用例集合可以更全面地覆盖各种输入情况，而不必测试每个可能的输入。
    - 例如，对于一个登录功能，等价测试用例可以包括一个有效的用户名和密码组合、一个无效的用户名和有效的密码组合，以及一个有效的用户名和无效的密码组合，以代表不同的等效类别。

    简单说，测试用例是用于测试软件系统的具体实例，异常测试用例用于验证异常情况的处理，而等价测试用例用于覆盖不同等效类别的输入情况，以确保软件在各种情况下都能正确运行。有效的测试用例设计是软件测试的关键，它有助于发现潜在问题并提高软件的质量。

2.系统组成

模糊测试分为测试数据生成，数据交互与控制，测试结果反馈三个阶段。

`测试数据生成` -> `数据交互与控制` -> `测试结果反馈`

根据这些过程，对应着模糊测试一般分为三个模块：`数据生成模块`，`环境控制模块`和`状态监控模块`。

模糊测试通常分为三个关键模块，即数据生成模块、环境控制模块和状态监控模块。这些模块在模糊测试中起着不同但关键的作用：

1. **数据生成模块**：
   - **用途**：数据生成模块的主要目标是生成模糊测试用例，这些测试用例将用作目标程序或系统的输入。这些测试用例通常包括各种异常、不合法或随机的数据，以检测程序在面对不正常输入时的反应。
   - **方法**：数据生成模块可以使用多种技术来创建测试用例，包括但不限于以下几种：
     - **随机生成**：生成随机数据，包括字符、数字、二进制数据等。
     - **模式匹配**：根据已知的数据模式或协议规范生成数据。例如，HTTP请求中的URL、参数和头部可以按照HTTP协议规范生成。
     - **变异**：基于已有的有效输入，通过添加、删除或修改数据的方式来生成测试用例，使其变得不正常或异常。
     - **基于语法的生成**：使用语法规则来生成数据，确保生成的数据符合语法结构，如JSON、XML等。
2. **环境控制模块**：
   - **用途**：环境控制模块负责模拟测试环境，包括目标程序的运行环境以及可能的外部条件，如网络、文件系统等。它确保模糊测试过程中的环境是可控制和可复制的。
   - **方法**：环境控制模块可以采用以下方法来实现环境模拟和控制：
     - **虚拟化**：在虚拟环境中运行目标程序，以隔离测试过程，确保系统稳定性和安全性。
     - **模拟网络环境**：模拟不同网络条件，如高延迟、丢包等，以测试程序在不同网络条件下的表现。
     - **文件系统操作**：模拟文件操作，包括文件创建、删除、修改等，以测试程序对文件操作的鲁棒性。
     - **资源限制**：限制CPU、内存和网络带宽等资源，以模拟资源受限的情况。
3. **状态监控模块**：
   - **用途**：状态监控模块用于监视目标程序的执行状态和异常情况。它捕获程序的行为，包括崩溃、错误、异常输出等，并将这些信息用于后续的分析和报告。
   - **方法**：状态监控模块可以采用以下方法来监控目标程序的状态：
     - **日志记录**：记录程序的输出、错误消息和运行时信息，以便后续分析。
     - **异常检测**：监测程序是否发生崩溃或异常，例如，通过监视进程是否终止或产生错误代码。
     - **内存分析**：检查程序的内存使用情况，以发现内存泄漏或非法内存访问。
     - **性能分析**：监控程序的性能指标，如响应时间、CPU利用率等，以评估程序的性能稳定性。

> 鲁棒性（Robustness）是计算机科学和软件工程领域的一个重要概念，它指的是系统或软件在面对异常或不正常输入、外部条件或行为时能够保持稳定性和可靠性的能力。

值得一提：有关状态监控模块的处理

状态监控模块在捕获异常时需要对测试对象异常的三种情况进行分别处理：

1. **被测试对象内置的异常处理流程捕获的异常**：这种异常情况是指目标程序或系统能够识别和捕获异常，并按照其内置的异常处理流程来处理。这些异常通常不会导致程序崩溃，因为它们得到了适当的处理。这类异常情况通常不会揭示漏洞，因为程序已经处理了异常情况。
2. **无法被测试对象内置的异常处理捕获的，但因异常被中断执行的异常**：这种异常情况是指目标程序或系统无法正确捕获或处理异常，但异常的出现导致了程序的中断或崩溃。尽管程序崩溃，但这些异常情况通常具有更高的价值，因为它们表明存在漏洞或错误，可能需要进行修复。
3. **无法被测试对象内置的异常处理捕获的，但异常不被中断执行，导致非预期的结果的异常**：这种异常情况是指异常没有导致程序的崩溃，但却引发了程序内部的非预期行为或错误结果。这类异常情况同样具有较高的价值，因为它们可能揭示了潜在的漏洞或问题，尤其是在用户体验和系统可靠性方面。

其中，后两种异常情况更有价值，因为它们通常指示了潜在的漏洞或问题，这些问题可能需要开发人员的关注和修复，以提高系统的鲁棒性和稳定性。模糊测试的目标之一就是发现并报告这些异常情况，以帮助改进软件的质量。

0x03 基础方法技术

数据生成方法

1.基本类型数据生成方法

1. **预定义序列**：
   - **经验数据**：经验数据是基于先前的测试经验或实际使用中的数据样本来定义的。它们通常反映了一组合理的、常见的或已知的输入值。经验数据有助于测试基础功能，确保软件在正常情况下运行。例如，对于一个登录页面，经验数据可能包括一组有效的用户名和密码组合。
   - **特别数据**：特别数据是为了测试特定边界条件或较少常见情况而定义的数据。这些数据通常不符合常规输入，但可能会揭示潜在问题。例如，测试一个计算器应用程序时，特别数据可能包括除以零的情况。
2. **随机数序列**：
   - 随机数序列是根据随机性生成的一系列数字。它们广泛用于模糊测试，因为它们可以模拟未知或不可预测的输入情况。随机数序列可以包括整数、浮点数或其他数字类型。
   - 随机数生成器可以根据不同的分布（如均匀分布、正态分布等）生成随机数。在模糊测试中，可以指定随机数的范围和分布来控制生成的数据。
3. **小数值覆盖大数值**：
   - 这是一种测试策略，其中测试用例倾向于包括较小的数值，以测试系统对较小值的处理。这是因为较小的值可能更容易导致不正常行为，如除以零或下溢。
   - 例如，在测试一个计算器应用程序时，可以生成一系列小于1的随机浮点数来检查除法操作的鲁棒性。

2.复合类型数据生成方法

文件、网络数据包分别按照文件格式网络协议格式将基本类型数据组合成符合类型数据。

1. **文件生成**：

- 对于文本文件，可以创建一个文本文件对象，然后将字符串数据写入文件。在不同的编程语言中，有文件写入和操作的库和函数可供使用。
- 对于二进制文件，可以使用二进制文件的格式规范来构建文件头和内容部分，然后将它们写入文件。

2. **网络数据包生成**：
   - 根据特定的网络协议规范，创建网络数据包对象。这可以包括数据包头和数据包主体。
   - 在各种编程语言中，有库和工具可用于构建和编码网络数据包，如 `struct` 模块（Python）、`ByteBuffer`（Java）等。
3. **XML和JSON生成**：
   - 对于XML，可以使用XML解析库或API创建XML文档对象，然后添加元素和属性。
   - 对于JSON，可以创建JSON对象，包括键值对和嵌套的数组和对象。
4. **数据库记录生成**：
   - 使用数据库连接库或ORM（对象关系映射）工具创建数据库记录对象。不同的编程语言和数据库系统有不同的方法来操作数据库。
   - 设置记录的字段值，然后执行插入或更新操作。
5. **HTML生成**：
   - 创建HTML文档对象，然后使用HTML标签和属性来构建文档。
   - 在许多编程语言中，可以使用字符串拼接或HTML构建库来构建HTML文档。
6. **复杂对象生成**：
   - 根据数据结构的定义，构建复杂的数据对象。这可能涉及创建嵌套的数据结构、图形对象等。
   - 使用编程语言的数据结构和面向对象编程的功能来创建对象。

3.多阶段交互类型数据生成方法

当测试对象向外界提供服务的过程包含多次数据交互时，客户端与服务端的数据包必须根据对方的请求与相应进行构造，双方按照协议约定的过程按次序发送数据。多阶段交互类型数据生成方法用于模拟具有多次数据交互的复杂场景，通常在客户端和服务端之间进行数据包的交互。这种方法模拟了真实世界中的数据通信过程，涵盖了多个阶段的数据生成和交互，以测试系统的互操作性和完整性。

> 简单来说就是通过交互进行数据生成。

以下是多阶段交互类型数据生成方法的一般步骤：

1. **定义交互协议**：
   - 首先，需要明确定义客户端和服务端之间的交互协议。这包括请求和响应的消息格式、顺序、字段和协议规范。
2. **生成请求数据**：
   - 从客户端的角度开始，生成符合协议规范的请求数据包。这可能包括构建请求头、请求体、参数等信息。
3. **发送请求**：
   - 将生成的请求数据发送到服务端，模拟客户端向服务端发出请求。
4. **解析请求**：
   - 在服务端接收到请求后，解析请求数据包，检查其有效性和合法性。服务端需要遵循协议规范来处理请求。

5. **生成响应数据**:
    - 根据接收到的请求，服务端生成符合协议规范的响应数据包。这包括构建响应头、响应体、状态码等信息。
6. **发送响应**:
    - 将生成的响应数据发送回客户端，模拟服务端对客户端的响应。
7. **解析响应**:
    - 客户端接收到响应后，解析响应数据包，验证响应是否符合协议规范。
8. **继续交互**:
    - 根据协议规范，可能需要进行多轮的数据交互。客户端和服务端依次生成请求和响应，模拟多阶段的交互。
9. **结束交互**:
    - 最后，根据测试需求，可以结束交互并生成测试报告，分析交互期间发现的问题和异常。

多阶段交互类型数据生成方法可以帮助测试人员或工具模拟复杂的数据通信场景，以确保系统在实际使用中能够正确处理多个数据交互步骤。这对于测试网络应用程序、通信协议、API等非常重要，因为它可以发现系统中的互操作性问题、数据丢失、顺序错误等潜在问题。

环境控制技术

1.运行环境控制技术

运行环境控制技术在模糊测试中扮演着重要的角色，它涉及到管理和控制模糊测试的执行环境，以确保测试的可控性和可重复性。

> 简单来说就是构建一个符合测试实际的环，能够控制和维护，并且进行恢复的技术。

`构建环境` -> `控制和维护` -> `恢复`

以下是关于运行环境控制技术的一些常见内容:

1. **虚拟化和容器化**:
    - 使用虚拟机或容器技术可以创建隔离的测试环境，使测试过程不会对实际系统产生影响。
    - 通过虚拟化或容器化，可以轻松创建多个独立的测试环境，每个环境可以运行不同的测试用例。
2. **快照和还原**:
    - 创建运行环境的快照，以记录环境的状态和配置。在测试结束后，可以还原环境到快照状态，确保每次测试都从相同的起点开始。
    - 这对于确保可重复性和排查问题非常有用。
3. **资源隔离**:
    - 控制测试过程中的资源使用，以避免测试对主机系统的影响。
    - 可以限制CPU、内存、网络带宽等资源的使用，确保测试不会导致系统崩溃或不稳定。
4. **环境变量控制**:

- 通过设置环境变量，可以控制测试过程中使用的配置和参数。这包括路径、文件配置、网络地址等。
- 确保测试环境与实际环境的配置一致。

5. **随机性控制**：
   - 有时模糊测试需要随机生成输入数据，但也可能需要控制随机性，以确保测试可控。
   - 可以使用种子值来控制伪随机数生成器的行为，以在多次测试中生成相同的随机数据。
6. **日志和监控**：
   - 记录测试过程中的日志和监控数据，以便后续分析和排查问题。
   - 这包括记录测试用例、异常情况、资源利用率等信息。
7. **恢复机制**：
   - 在测试过程中，可能会发生不正常的情况，如崩溃或异常。具备恢复机制可以在测试中自动处理这些问题，使测试能够继续进行。
8. **并发和分布式测试**：
   - 在模糊测试中，可能需要同时执行大量测试用例，使用并发和分布式测试可以提高测试效率。
   - 这涉及到控制多个测试实例的协同工作，确保它们不会相互干扰。

运行环境控制技术的目标是确保模糊测试的可控性、可重复性和安全性。通过这些技术，可以更好地管理测试环境，从而更有效地发现潜在的问题和漏洞。

2.程序运行控制技术

1. **程序启动和终止**：
   - 控制模糊测试程序的启动和终止，确保它可以在需要时启动，以及在测试结束后正确终止。
2. **暂停和继续**：
   - 允许在测试过程中暂停 fuzz 测试程序的执行，以便检查状态、调试问题或进行其他操作。之后可以继续执行测试。
3. **调试和修改**：
   - 提供调试功能，允许测试人员在运行时检查程序状态、变量值和执行路径，以排查问题。
   - 在需要时，还可以修改程序的行为，例如修改输入数据、修改配置或注入调试语句。
4. **进程句柄控制**：
   - 使用进程句柄，可以监控和控制 fuzz 测试程序的执行。这包括获取进程状态、发送信号、终止进程等。
5. **API接口**：
   - 提供各种API接口，以便外部程序与 fuzz 测试程序进行通信和控制。这些接口可以用于启动测试、发送测试用例、检索测试结果等。

3.数据强制输入技术

Fuzz测试的数据强制输入技术是用于将模糊测试生成的测试数据传递给目标程序的方法。这些技术涵盖了多个方面，包括网络数据输入、文件数据输入、用户操作输入以及内存数据修改。

> 分别用于网络、文件、图形用户界面和内存攻击。

以下是关于这些技术的详细说明：

1. **网络数据输入技术**：

   - 在网络数据输入技术中，模糊测试工具通过模拟网络通信方式将生成的测试数据发送给目标程序。这通常涉及以下步骤：
     - **网络形式强制发送**：工具使用适当的网络协议连接到目标程序，按照协议规范将测试数据发送给目标。
     - **连接协议**：模糊测试工具使用与目标程序通信所需的协议，例如HTTP、FTP、SMTP等。
     - **数据包构造**：根据协议规范构造符合格式的数据包，将测试数据包含在数据包中。
   - 这种方法适用于测试网络应用程序、服务器、通信协议等，以验证它们对不规范或恶意输入的鲁棒性。

2. **文件数据输入技术**：

   - 文件数据输入技术用于将生成的测试数据传递给目标程序的方式，通常通过文件传递。这包括以下方法：
     - **命令行参数传入**：模糊测试工具将测试数据作为命令行参数传递给目标程序。
     - **进程交互机制**：工具可以通过与目标程序的进程进行交互，将数据传递给正在运行的程序。
     - **文件读取**：测试工具可以创建临时文件，将测试数据写入文件，然后通过文件读取操作将数据提供给目标程序。
   - 这种方法适用于测试本地应用程序、命令行工具等，以验证它们对不同数据源的处理。

3. **用户操作输入技术**：

   - 用户操作输入技术模拟用户与目标程序进行交互的方式，包括模拟鼠标、键盘输入和其他用户界面操作。这包括以下方法：
     - **模拟鼠标和键盘输入**：工具模拟用户通过键盘输入文本、通过鼠标点击按钮或执行其他用户界面操作，将测试数据输入到目标程序中。
   - 这种方法适用于测试图形用户界面（GUI）应用程序，以验证它们对用户输入的鲁棒性。

4. **内存数据修改技术**：

   - 内存数据修改技术允许模糊测试工具直接在目标程序的内存中修改数据，以模拟恶意攻击或异常情况。这包括以下方法：
     - **直接在内存中进行修改**：工具通过访问目标程序的内存空间，修改特定的内存位置，注入测试数据或更改程序状态。
   - 这种方法通常用于测试漏洞、缓冲区溢出等安全问题，以评估程序的鲁棒性。

这些数据强制输入技术允许模糊测试工具将测试数据传递给目标程序，以评估程序的鲁棒性和安全性。根据测试目标和应用程序的性质，可以选择适当的技术来进行模糊测试。

状态监控技术

状态监控技术在模糊测试中起着关键作用，它允许测试人员监控目标程序的生命周期、执行状态、异常状态以及输入输出，从而更好地评估程序的鲁棒性和安全性。以下是关于这些监控技术的简要说明：

1. **生命周期监控技术**：

   - 生命周期监控技术用于追踪目标程序的整个生命周期，包括启动、执行和终止阶段。
   - 实现方法：
     - 启动和终止：监控程序的启动和终止，可以通过记录程序的启动时间和结束时间来实现。
     - 进程监控：使用操作系统提供的工具或库来监控目标程序的进程，以确保它始终处于活动状态。

    - 日志记录：记录程序的运行日志，包括启动和终止事件，以便进行后续分析。

2. **输入输出监控技术**：

- 输入输出监控技术用于捕获目标程序与外部环境的数据交互，包括输入数据和输出结果。
- 实现方法：
    - 输入捕获：截获模糊测试工具生成的输入数据，包括网络数据、文件数据、用户操作等。
    - 输出监控：捕获目标程序的输出结果，包括响应数据、日志、错误信息等。
    - 数据流追踪：使用数据流分析工具来追踪输入数据在程序内部的处理过程，以检测数据修改或异常行为。

3. **执行状态监控技术**：

- 执行状态监控技术用于实时监控目标程序的执行状态，以检测异常行为和问题。
- 实现方法：
    - 异常检测：监控程序的执行过程，检测是否出现异常情况，如崩溃、死锁、超时等。
    - 内存检查：定期检查程序的内存使用情况，以检测内存泄漏或溢出问题。
    - CPU利用率：监控程序的CPU利用率，以确定是否存在高负载情况。

这些监控技术可以通过使用各种工具和库来实现，具体方法取决于测试环境和测试工具的要求。通过监控目标程序的生命周期、输入输出和执行状态，测试人员可以及时发现异常情况并进行更准确的控制和分析，以提高模糊测试的效率和效果。

0x04 模糊测试优化方法

让我分别解释一下这些方法：

1. **灰盒模糊测试**：

- 灰盒模糊测试是介于白盒（静态分析）和黑盒（仅关注输入输出）之间的一种测试方法。
- 在灰盒模糊测试中，测试人员或工具通常会逆向工程目标程序，分析其内部结构和逻辑，以更好地理解程序的运行方式。
- 然后，根据这些了解，有针对性地生成测试用例，限定字段值或者注重测试一些关键路径和敏感函数，以提高发现漏洞的机会。

2. **白盒模糊测试**：

- 白盒模糊测试进一步加强了灰盒测试的概念，引入了符号执行等高级分析技术。
- 在白盒模糊测试中，测试工具会使用逆向工程技术分析目标程序的源代码、控制流、数据流等内部信息。
- 这些信息用于构建符号执行模型，以理解程序的执行路径，然后生成测试用例，以测试这些路径。
- 白盒模糊测试可以更全面地覆盖程序的各种执行路径，从而提高漏洞的发现概率。

3. **基于反馈的模糊测试**：

- 基于反馈的模糊测试方法侧重于收集和分析测试过程中产生的反馈信息，以优化后续测试用例的生成。
- 在此方法中，测试工具会统计分析模糊测试用例的特征和测试结果的特征。
- 使用这些信息，测试工具可以动态地调整测试用例生成策略，以生成更有潜力的测试用例，进一步提高发现漏洞的效率。
- 这种方法通常使用统计方法（如u测试）来分析和调整测试用例的生成，从而实现智能的测试。

这些高级模糊测试方法旨在通过更深入的分析和优化来提高模糊测试的效率和效果。它们通常需要更多的专业知识和复杂的工具支持，但可以在发现漏洞方面取得更好的结果。选择哪种方法通常取决于测试目标、测试环境和可用资源。

0x05 工具推荐

这里有几种与模糊测试相关的工具和技术，包括AFL-Unicorn、Qiling、SlowFuzz、PerfFuzz、以及AFL++的QEMU和Unicorn模式。让我们来看看它们的优点、缺点以及双重作用：

1. **AFL-Unicorn**:
   - **优点**:
     - 允许将American Fuzzy Lop（AFL）与Unicorn引擎结合使用，从而能够对更广泛的目标进行模糊测试。
     - 具有高度可定制性，用户可以为目标二进制文件创建自定义的Unicorn插件。
   - **缺点**:
     - 复杂性较高，需要熟悉AFL和Unicorn。
     - 需要较多的配置和调试。
2. **Qiling**:
   - **优点**:
     - 可以模拟多种操作系统，包括Linux、Windows等，使其具有广泛的应用。
     - 允许用户在用户模式和内核模式下进行模糊测试。
   - **缺点**:
     - 需要深入了解操作系统的内部工作原理，以进行高效的模糊测试。
     - 需要额外的配置和学习曲线。
3. **SlowFuzz**:
   - **优点**:
     - 采用基于符号执行的方法，具有较高的漏洞发现潜力。
     - 能够在更广泛的输入空间中搜索漏洞。
   - **缺点**:
     - 符号执行速度较慢，可能需要更多的时间来执行测试。
     - 可能需要更多的计算资源。
4. **PerfFuzz**:
   - **优点**:
     - 利用性能计数器（Performance Counters）来导向模糊测试，从而提高了测试效率。
     - 可以更快地发现性能敏感漏洞。
   - **缺点**:
     - 对于一些非性能相关的漏洞，可能不够敏感。
     - 对于某些平台，可能需要额外的硬件支持。
5. **AFL++的QEMU和Unicorn模式**:
   - **优点**:
     - 可以使用AFL++与QEMU或Unicorn引擎结合，提供了广泛的目标支持。

    - AFL++改进了原始AFL的性能和功能，提供了更好的模糊测试体验。
- **缺点**:
    - 需要一定的配置和学习曲线。
    - 在某些情况下，可能需要更多的计算资源。

这些工具和技术各有优点和缺点，选择哪一个取决于具体需求和目标。例如，如果需要测试多个操作系统，Qiling可能是一个不错的选择。如果关注性能敏感漏洞，PerfFuzz可能更合适。另外，一些工具可以结合使用，以充分发挥各自的优势。最终，选择合适的工具取决于您的具体测试场景和资源可用性。

0x06 尾声

阅读到这里我们简单了解了模糊测试（Fuzz Testing）的各个方面，包括其基本概念、方法和相关技术。模糊测试作为一种强大的测试方法，具有广泛的应用领域，可用于发现软件漏洞、提高安全性和质量。

我们深入了解了模糊测试的基本要素，包括测试用例、异常测试用例和等价测试用例，以及数据集合和程序执行路径。这些概念构成了模糊测试的基础。探索了模糊测试的不同阶段，包括测试数据生成、数据交互与控制、测试结果反馈，以及这些阶段的关联模块和技术。这些阶段共同构建了模糊测试的全貌。

此外，我们介绍了一些高级模糊测试技术，如灰盒模糊测试、白盒模糊测试和基于反馈的模糊测试，以及它们的应用和优势。并且最后，推荐了一些相关工具及其优劣。

接下来，让我们把主动权交回你的手里，去探索更广阔的网络安全世界吧！

posted @ 2023-10-08 20:53 .N1nEmAn 阅读(149) 评论(0)