

# 南京邮电大学

## 毕业设计（论文）

题    目                    定向覆盖模糊测试工具的设计与实现

专    业                    计算机科学与技术

学生姓名                    雷尚远

班级学号                    B190303    B19030334

指导老师                    王子元

指导单位                    计算机学院、软件学院、网络空间安全学院

日期：    2023 年 3 月 x 日至 2023 年 6 月 x 日

## 毕业设计（论文）原创性声明

本人郑重声明：所提交的毕业设计（论文），是本人在导师指导下，独立进行研究工作所取得的成果。除文中已注明引用的内容外，本毕业设计（论文）不包含任何其他个人或集体已经发表或撰写过的作品成果。对本研究做出过重要贡献的个人和集体，均已在文中以明确方式标明并表示了谢意。

论文作者签名：

日期：      年      月      日

## 摘 要

模糊测试（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安全漏洞的方法，是软件安全领域常用的方法之一。由于代码覆盖率与漏洞覆盖率密切相关，大多数模糊测试工具都是以代码覆盖率为导向。然而，由于大多数被覆盖测试的代码可能并不包含漏洞，这使得盲目地扩展代码覆盖率的方式在实际测试时效率较低。极端情况尤为如此。与盲目增加代码覆盖率的模糊测试不同，定向覆盖的灰盒模糊测试（DGF）将大部分时间用于检测特定目标区域（例如，易出错代码段）而不会浪费资源于不相关的部分。因此，DGF 特别适用于补丁测试、漏洞复现以及特殊漏洞检测等场景。目前，DGF 已成为一个快速发展的研究方向。基于一些先进的定向覆盖模糊测试工具的研究和相关调查，本文主要做了以下点工作：

- (1) 基于现有的模糊测试工具框架 AFL（American Fuzzy Lop）以及 AFLGo 做了定向覆盖策略的设计和集成；
- (2) 实现了简单的定向覆盖的模糊测试命令行工具；
- (3) 针对相应的公开通用漏洞集（CVE）做了复现及定向实验对比测试。

此外本文亦通过分析工具设计以及实现过程中的局限性与不足，对于未来该方向的研究发展做出了一些展望。

**关键词：** 模糊测试；定向覆盖模糊测试；灰盒测试；软件安全

## ABSTRACT

Fuzzing is a method of discovering software security vulnerabilities by providing unexpected inputs to a target system and monitoring for abnormal results. It is one of the commonly used methods in the field of software security. As code coverage is closely related to vulnerability coverage, most fuzz testing tools are guided by code coverage. However, blindly extending code coverage may be inefficient in practical testing since most of the covered code may not contain vulnerabilities, especially for corner cases. In contrast to blind code coverage-based fuzz testing, directed grey-box fuzzing (DGF) spends most of its time detecting specific target regions (such as error-prone code segments) rather than wasting resources on irrelevant parts. Thus, DGF is particularly suitable for scenarios such as patch testing, bug reproduction, and special bug detection. For now, DGF has become a fast-growing research area. Based on some advanced directed coverage fuzz testing tools and relevant investigations, this article mainly focuses on the following points of work:

- (1) Designed and integrated a directed coverage strategy based on the existing fuzzy testing tool framework AFL (American Fuzzy Lop) and AFLGo;
- (2) Implemented a simple command-line tool for directed fuzz testing;
- (3) conducted reproductions and directed experiments on corresponding public vulnerability databases (CVE) for comparative testing.

In addition, this article also provides some prospects for the future research and development of this direction by analyzing the limitations and deficiencies in the design and implementation process of the tool.

**Keywords:** Fuzzing; Directed Greybox Fuzzing; Greybox test; Software Security

# 目 录

第一章 绪论.....	1
1.1 背景分析.....	1
1.2 国内外研究现状.....	1
1.3 研究内容.....	3
1.4 论文结构.....	3
第二章 相关技术研究.....	5
2.1 模糊测试技术.....	5
2.1.1 基本概念定义.....	5
2.1.2 基本架构.....	6
2.1.3 黑盒模糊测试技术.....	6
2.1.4 白盒模糊测试技术.....	6
2.1.5 灰盒模糊测试技术.....	6
2.2 定向模糊测试技术.....	6
2.2.1 白盒定向模糊测试技术.....	6
2.2.2 灰盒定向模糊测试技术.....	6
2.3 研究动机.....	6
2.4 本章小结.....	6
第三章 定向模糊测试策略设计.....	7
3.1 AFLGo 架构研究.....	7
3.1.1 距离计算机制.....	7
3.1.2 能量调度机制.....	7
3.2 定向适应度指标.....	7
3.2.1 距离定义.....	7
3.2.2 目标函数集覆盖率.....	7
3.2.3 能量调度机制.....	7
3.3 本章小结.....	7
第四章 基于 AFLGo 的定向模糊测试系统的实现.....	8
4.1 需求分析.....	8
4.2 架构设计.....	8
4.3 静态分析器的改进.....	8
4.4 定向模糊测试工具.....	8
4.5 本章小结.....	8
第五章 系统测试.....	9
5.1 系统测试概述.....	9
5.1.1 系统测试目标.....	9

5.1.2 系统测试环境.....	9
5.2 功能测试.....	9
5.3 实验评估.....	9
5.3.1 定向模糊测试工具.....	9
5.4 本章小结.....	9
第六章 总结与展望.....	10
6.1 总结.....	10
6.2 展望.....	10
结束语.....	11
致谢.....	12
参考文献.....	13

## 第一章 绪论

### 1.1 背景分析

“常用系统中可能会潜伏着严重的漏洞<sup>[1]</sup>。”这一论述源自于模糊测试首次面世的论文。其揭示了一个事实，即随着软件技术的不断发展，软件安全问题就将日益成为愈发重视的议题。在当前的信息化时代，软件已经成为了人们生活、工作和娱乐的重要组成部分，这也意味着我们将面临着越来越多的安全威胁。因此，确保软件安全已经成为了一项非常重要的任务。

软件安全（Software Security）就是使软件在受到恶意攻击的情形下依然能够继续正确运行及确保软件被在授权范围内合法使用的思想。在当今社会，软件越来越普及，并被广泛应用于各个领域，包括电商、金融、医疗等。但是，由于软件的复杂性和开发过程中的缺陷，软件本身也存在着各种安全问题。这些问题可能导致信息泄露、数据损坏、远程攻击等，对个人、企业甚至整个社会造成巨大的损失。因此，保障软件安全显得尤为重要。

近年来，因为软件漏洞造成的损失案例屡见不鲜。2017 年，全球范围内爆发了 WannaCry 勒索病毒攻击事件，该攻击利用了微软 Windows 操作系统中的漏洞，并导致了数十亿美元的经济损失；2019 年，美国资讯技术服务公司 SolarWinds 遭受了一次大规模的网络攻击，该攻击利用了 SolarWinds Orion 平台软件中的漏洞，影响了包括美国联邦政府在内的许多组织和机构；2021 年 2 月，法国 LCL 银行的客户登录自己的银行应用程序时，看到的是别人的银行账户信息。原因是由于备份超级计算机系统（日本惠普公司制造）的程序存在缺陷，超级计算机系统出现了意外，其中存储（/LARGE0）中的某些数据被误删除；2021 年 12 月，知名日志框架 Log4j2 被爆出远程代码执行漏洞，影响了大量使用该框架的中间件和应用，给企业和用户带来了巨大的安全风险。

以上诸多例子可以说明，大多数的安全事件都是攻击者利用软件系统中的漏洞从而进行攻击引发的。因而可以帮助发现和修复安全漏洞的软件测试技术（Software testing）一直以来都是软件安全领域的一个重要议题。

软件测试可以通过模拟攻击者的行为来发现这些安全漏洞，并提供关于如何修复这些漏洞的信息。例如，黑盒测试可以探测应用程序中的安全问题，白盒测试可以评估应用程序的源代码中是否有漏洞，静态分析可以扫描源代码以发现潜在的安全问题，动态分析可以模拟攻击场景并检查应用程序的反应。

此外，软件测试还可以帮助确保应用程序在面对各种攻击时具有足够的鲁棒性和可靠性。它可以测试应用程序的身份验证和授权机制、加密技术、网络协议、输入输出数据验证等方面的功能，以确保应用程序满足安全需求。

### 1.2 国内外研究现状

软件安全信息系统和软件安全代码的有效安全项目往往依靠两种自动的安全测试：静态安全扫描测试和动态安全扫描测试。

在软件的开发期间，为了保证软件的安全性，通常会进行软件安全静态扫描。

这个过程是通过威胁建模和分析来完成的，其目的是对静态代码进行全面地扫描，以便及早地发现任何可能存在的安全漏洞。其是在不运行程序的情况下对软件进行测试和评估。静态分析可以检查代码、设计和文档等，以发现潜在的问题和错误，并确保软件符合某些标准或规范。由于本文主要探讨针对代码的漏洞审查，关于软件工程部分的文档、标准以及接口设计的测试技术在此不再赘述。利用数据流分析，符号执行以及污点分析等静态软件分析技术可以检查源代码中的错误和缺陷，包括语法错误、类型错误、内存泄漏、空指针引用等。与传统的动态测试相比，静态扫描可以更早地发现安全问题，因为它可以在代码尚未被编译或执行之前就进行检测。此外，静态扫描还可以减少测试成本，提高测试效率，并帮助开发团队更好地理解代码中的潜在安全风险。

软件安全动态扫描是一种对工作环境中实际运行的代码进行扫描的技术，它能够在代码运行时检测和分析可能存在的漏洞、缺陷和错误。与静态代码分析不同，动态扫描具有更强的准确性和实时性，因为它是在真实的环境中对代码进行测试和评估。通过使用动态扫描技术，开发人员和安全专家可以有效地识别并修复潜在的安全漏洞，从而保护软件系统免受攻击和破坏。此外，动态扫描还可以帮助企业遵循各种合规性标准和法规要求，确保其软件应用程序的安全性和稳定性。

模糊测试技术（Fuzzing Test）是动态安全扫描测试中重要的一种方式。而自从 1988 年模糊测试这一概念被提出后，这一方法一直在软件安全测试领域保持着较高的活跃度和关注度。在提出伊始，其主要用于测试操作系统。之后，随着软件技术的发展，模糊测试技术不断得到改进和推广，并应用于网络、移动设备等领域。目前，模糊测试技术已成为一种成熟的自动化测试技术，可以有效地检测软件中存在的漏洞和安全隐患。并且迄今为止其社区依然十分活跃，在 GitHub 上有超过 1000 个与模糊测试相关的仓库<sup>[2]</sup>。为了防止被恶意攻击，许多商业软件公司，例如 Adobe, Cisco, Google, 和 Microsoft 都将模糊测试作为其雇员软件开发安全测试的必要环节。可以说，模糊测试是软件安全领域中一个经久不衰的热门议题。

而定向模糊测试（Directed Fuzzing）作为模糊测试的一个研究方向，主要关注重点区域（例如，易出错区域）并且将大部分的时间用于到达测试这些位置而不浪费资源在无关部分<sup>[3]</sup>。从本源上讲，定向模糊测试工具早期的解决思路主要是基于利用程序分析和约束求解来生成检测不同程序执行路径输入符号的执行技术<sup>[4-9]</sup>。然而，由于定向符号执行技术（DSE）依赖于大量的程序分析和约束求解，其受限于兼容性和可扩展性的限制。

在 2017 年，Böhme 等人提出的 AFLGo<sup>[10]</sup>引入了定向灰盒模糊测试（DGF）的概念。这是定向模糊测试的又一重要工作。其开创式地将位置的可达性问题转化为生成种子和其目标集之间距离的最小值问题。通过给更靠近目标集的种子更多的变异机会，它可以逐渐引导灰盒测试接近程序目标位置。与定向符号执行技术相比，DGF 有更好的可拓展性，并且在测试效率上有几个数量级的提升。例如，Böhme 等人可以在 20 分钟内重现 Heartbleed<sup>[11]</sup>（CVE-2014-0160）漏洞，而定向符号执行工具 KATCH<sup>[12]</sup> 需要 24 小时以上<sup>[10]</sup>。

此后，许多基于 AFLGo 的定向性改进工作被相继提出。在 2018 年，Hongxu



Chen 等人提出的 Hawkeye<sup>[13]</sup> 指出了 AFLGo 的距离计算方式导致的能量分配偏向导致的在测试的路径选择上偏向于距离目标更近的路径（或最短路径），而这有可能漏掉一些存在于较长路径上潜在的漏洞，于是其使用了新的距离度量来辅助达到更精确的距离导引；在 2020 年 Wang 等人提出的 UAFL<sup>[14]</sup> 开创了针对于特定漏洞行为的定向性导引，其利用目标行为次序而不是目标位置来查找释放后使用的漏洞，这种漏洞的内存操作一定要按照一定次序执行（例如，分配，使用，然后释放内存）才能触发；在 2021 年 Gwangmu Lee 等人提出的 CDGF<sup>[15]</sup> 则指出 AFLGo 在路径选择中的不考虑执行路径导致的目标位置的顺序影响从而忽略满足特定行为引起崩溃条件的种子的缺陷，他们的解决思路不再是利用 AFLGo 的种子分配能量的方式来指引种子变异从而检测指定目标位置，而改为将其视作约束求解问题，通过设置一系列的约束来优先考虑选择符合要求的种子从而达到检测指定目标位置的目标；而在 2022 年由 Heqing Huang 等人提出的 BEACON<sup>[16]</sup> 则从另一种方式进一步尝试提高定向模糊测试的速率：通过“剪枝”，即修建无效路径的方式。其结合轻量级的静态程序分析，来计算到达目标位置的抽象前提条件，并在运行时剪除那些不满足条件的路径，从而可以有效地提高效率，避免在无效或不可达的路径上浪费时间和资源。

### 1.3 研究内容

首先，针对于模糊测试技术，本文做了一个简单的梳理，对于目前模糊测试的基本主流技术框架做了分析和总结。自 1988 年这一技术的提出以来，相关技术研究百花齐放，技术快速迭代发展，直到近些年才有相关的总结研究和谱系分析<sup>[2,17]</sup>，建立系统性的分析。本文在参考相关文献的基础上对模糊测试的技术框架做了一个梳理。

其次，针对于定向灰盒模糊测试的开篇之作 AFLGo，本文对其的技术架构做了一个详细的总结梳理。对于 AFLGo 的一些实现细节做了详细的分析和总结。除此之外，本文还探讨了 AFLGo 在实际应用中的优缺点以及相比于其他模糊测试工具所具备的特点。针对 AFLGo 的优点，我们详细阐述了其灵活性、高效性和可扩展性，同时也分析了其在某些情况下可能会出现的一些问题。此外，本文还介绍了 AFLGo 在不同场景下的应用，帮助读者更好地理解如何使用 AFLGo 进行测试。

最后，参考以 AFLGo, Hawkeye 为主的定向性模糊测试工具，针对于 AFLGo 的架构进行了改进和集成。本文主要从距离定义和目标函数集合覆盖率两个指标针对于定向策略做了设计和修改，并结合定向模糊测试工具 AFLGo 和 LLVM 的 Pass<sup>[18]</sup> 技术做的静态分析的实现，实现了指标在 AFLGo 的集成。最终，本文在实现后通过实验成功复现 libxm2<sup>[19]</sup> 的 CVE-2017-{9047,9048}，证实了集成的可行性和可靠性。

### 1.4 论文结构

本文共分六章，各个章节的内容如下：

第一章：主要介绍本文的研究背景、国内外研究现状、研究内容以及本文的

论文结构。

第二章：主要介绍本文涉及技术的概况，包括模糊测试技术和定向模糊测试技术，介绍利用这些技术的典型架构，最后阐述本文的研究动机。

第三章：介绍 AFLGo 的主要架构和适应度指标，以及针对于其不足设计出的定向模糊测试适应度指标。详细阐述相应指标的核心设计, 包括插桩，目标函数集覆盖率的计算。

第四章：结合目标场景对实现的定向模糊测试工具做了需求分析，详细介绍了基于 LLVM 的 Pass 技术实现的插桩以及静态分析过程。对于结构集成以及实现方式的缘由做了详细分析。

第五章：对于实现集成的工具进行了实验测试，并做了相应的测试评估，包括工具测试的可行性和性能评估。

第六章：总结了本文的工作并对于未来定向模糊测试技术的发展及研究方向做了展望。

## 第二章 相关技术研究

### 2.1 模糊测试技术

模糊测试（Fuzz Testing）是一种软件测试技术，其主要思想是通过向输入参数、文件、网络请求等随机或半随机注入无效、异常、边界数据，来检测目标系统在处理异常情况时的鲁棒性。模糊测试可以帮助发现那些未经预料的漏洞和错误，这些问题可能会导致应用程序崩溃、停止响应或者执行意外的操作。

在进行模糊测试时，测试人员通常需要编写一个模糊测试工具，该工具可以生成大量的随机测试用例，并将这些测试用例输入到应用程序中进行测试。模糊测试通常被认为是一种高效的测试技术，因为它可以在较短的时间内检测应用程序中的许多潜在问题。此外，由于测试用例是随机生成的，模糊测试可以找出一些没有被其他测试方法发现的漏洞和错误。

模糊测试的过程通常分为以下几个步骤：

- (1) 选择目标：选择需要测试的软件目标，如应用程序、库、操作系统等。
- (2) 寻找输入：确定需要对目标注入的输入类型和数据源，如输入参数、文件、网络请求等。
- (3) 创建模糊数据：使用随机或半随机的方式生成模糊数据，并将其注入到目标中。
- (4) 监控程序行为：监控被测试程序的运行行为，如崩溃、错误输出等。
- (5) 分析结果：对测试结果进行分析和报告，识别潜在的漏洞和安全问题，并反馈给开发人员进行修复。

模糊测试是一种简单有效的测试方法，可以在较短时间内发现大量的异常情况，并帮助开发人员提高软件质量和安全性。

#### 2.1.1 基本概念定义

为了准确地讨论学界关注的模糊测试方向的问题以及梳理架构，我选择了采用 Manès 等人于论文<sup>[2]</sup>中提出的基本概念的定义。

- **Fuzzing**: 模糊测试技术是指使用从超出被测试程序（PUT）的预期输入空间的输入空间（“模糊输入空间”）采样的输入来执行被测试程序。  
事实上
- **Fuzz Testing**: 模糊测试是指使用模糊测试技术（Fuzzing）来测试被测试程序是否违反正确性策略。
- **Fuzzer**: 模糊测试工具是对被测试程序执行模糊测试的程序。
- **Bug Oracle**: 错误检测器是一个程序，可能是模糊测试工具的一部分，它确定被测试程序的给定执行是否违反了特定的正确性策略。
- **Fuzz Configuration**: 模糊算法的配置是指所有能控制模糊算法执行的参数配置。
- **Seed**: 种子是被测试程序的（通常结构良好的）输入，一般用于通过修改它来生成测试用例。

---

## 2.1.2 基本架构

---

### 算法 2-1: 模糊测试算法

---

```

Input:  $\mathbb{C}$ ,  $t_{\text{limit}}$ 
Output:  $\mathbb{B}$  // a finite set of bugs
1  $\mathbb{B} \leftarrow \emptyset$ ;
2  $\mathbb{C} \leftarrow \text{Preprocess}(\mathbb{C})$ ;
3 while  $t_{\text{elapsed}} < t_{\text{limit}} \wedge \text{Continue}(\mathbb{C})$  do
4    $\text{conf} \leftarrow \text{Schedule}(\mathbb{C}, t_{\text{elapsed}}, t_{\text{limit}})$ ;
5    $\text{tcs} \leftarrow \text{InputGen}(\text{conf})$ ;
   //  $O_{\text{bug}}$  is embedded in a fuzzer
6    $\mathbb{B}', \text{execinfos} \leftarrow \text{InputEval}(\text{conf}, \text{tcs}, O_{\text{bug}})$ ;
7    $\mathbb{C} \leftarrow \text{ConfUpdate}(\mathbb{C}, \text{conf}, \text{execinfos})$ ;
8    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$ ;
9 return  $\mathbb{B}$ ;

```

---

算法2-1是

## 2.1.3 黑盒模糊测试技术

## 2.1.4 白盒模糊测试技术

## 2.1.5 灰盒模糊测试技术

## 2.2 定向模糊测试技术

### 2.2.1 白盒定向模糊测试技术

### 2.2.2 灰盒定向模糊测试技术

## 2.3 研究动机

## 2.4 本章小结

## 第三章 定向模糊测试策略设计

### 算法 3-1: 定向灰盒模糊测试算法

---

```

Input:  $\mathbb{S}$  // a finite set of seeds
Input:  $\mathbb{T}$  // a finite set of targer sits
Output:  $\mathbb{S}'$  // a finite set of buggy seeds
1  $\mathbb{S}' \leftarrow \emptyset$ 
2  $SeedQueue \leftarrow \mathbb{S}$ 
3  $Graphs \leftarrow GraphExt(Code)$ 
4  $BBdistance \leftarrow DisCalcu(\mathbb{T}, Graphs)$ 
5 while  $!signal \wedge t_{elapsed} < t_{limit}$  do
6    $s \leftarrow Dequeue(SeedQueue)$ 
7    $trace \leftarrow Execution(s)$ 
8    $distance \leftarrow SeedDis(trace, BBdistance)$ 
9    $e \leftarrow AssinEnergy(s, t_{elapsed}, distance)$ 
10  for  $i \leftarrow 1$  to  $e$  do
11     $s' \leftarrow Mutation(s)$ 
12    if  $s'$  crashes then  $\mathbb{S}' \leftarrow \mathbb{S}' \cup s'$ 
13    if  $IsIntersting(s')$  then  $Enqueue(s', SeedQueue)$ 
14 return  $\mathbb{S}'$ 

```

---

### 3.1 AFLGo 架构研究

#### 3.1.1 距离计算机制

#### 3.1.2 能量调度机制

### 3.2 定向适应度指标

#### 3.2.1 距离定义

#### 3.2.2 目标函数集覆盖率

#### 3.2.3 能量调度机制

### 3.3 本章小结

## 第四章 基于 AFLGo 的定向模糊测试系统的实现

### 4.1 需求分析

### 4.2 架构设计

### 4.3 静态分析器的改进

### 4.4 定向模糊测试工具

### 4.5 本章小结

## 第五章 系统测试

### 5.1 系统测试概述

#### 5.1.1 系统测试目标

#### 5.1.2 系统测试环境

### 5.2 功能测试

### 5.3 实验评估

#### 5.3.1 定向模糊测试工具

### 5.4 本章小结

## 第六章 总结与展望



### 6.1 总结

### 6.2 展望



## 结束语

## 致 谢

本论文采用  $\text{\LaTeX}$  模版编写的，是基于南京邮电大学 2021 年理工艺教类的 Word 模板进行严格迁移编写的。本模板地址<https://github.com/dhiyu/NJUPT-Bachelor>感谢imguozr(<https://github.com/imguozr/NJUPThesis-Bachelor>)和lemoxiao(<https://github.com/lemoxiao/NJUPThesis-Scholar>)的工作，为本模板的形成奠定了大量的基础。

## 参考文献

- [1] Miller B P, Fredriksen L, So B. An empirical study of the reliability of unix utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.
- [2] Manès V J, Han H, Han C, et al. The art, science, and engineering of fuzzing: A survey[J]. IEEE Transactions on Software Engineering, 2019, 47(11): 2312-2331.
- [3] Wang P, Zhou X, Lu K, et al. The progress, challenges, and perspectives of directed greybox fuzzing: arXiv:2005.11907[EB/OL]. arXiv, 2022.
- [4] Ganesh V, Leek T, Rinard M. Taint-based directed whitebox fuzzing[C]//2009 IEEE 31st International Conference on Software Engineering. IEEE, 2009: 474-484.
- [5] Ma K K, Yit Phang K, Foster J S, et al. Directed symbolic execution[C]//Static Analysis: 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings 18. Springer, 2011: 95-111.
- [6] Person S, Yang G, Rungta N, et al. Directed incremental symbolic execution[J]. Acm Sigplan Notices, 2011, 46(6): 504-515.
- [7] Do T, Fong A C M, Pears R. Dynamic symbolic execution guided by data dependency analysis for high structural coverage[C]//Evaluation of Novel Approaches to Software Engineering: 7th International Conference, ENASE 2012, Warsaw, Poland, June 29-30, 2012, Revised Selected Papers 7. Springer, 2013: 3-15.
- [8] Ge X, Taneja K, Xie T, et al. Dyta: dynamic symbolic execution guided with static verification results[C]//Proceedings of the 33rd International Conference on Software Engineering. 2011: 992-994.
- [9] Li H, Kim T, Bat-Erdene M, et al. Software vulnerability detection using backward trace analysis and symbolic execution[C]//2013 International Conference on Availability, Reliability and Security. IEEE, 2013: 446-454.
- [10] Böhme M, Pham V T, Nguyen M D, et al. Directed greybox fuzzing[C/OL]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas Texas USA: ACM, 2017: 2329-2344. DOI: 10.1145/3133956.3134020.
- [11] Heartbleed - a vulnerability in openssl[EB/OL]. [2023-05-13]. <https://heartbleed.com/>.
- [12] Marinescu P D, Cadar C. Katch: High-coverage testing of software patches[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. 2013: 235-245.
- [13] Chen H, Xue Y, Li Y, et al. Hawkeye: Towards a desired directed grey-box fuzzer[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018: 2095-2108.
- [14] Wang H, Xie X, Li Y, et al. Typestate-guided fuzzer for discovering use-after-free vulnerabilities [C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020: 999-1010.
- [15] Lee G, Shim W, Lee B. Constraint-guided directed greybox fuzzing.[C]//USENIX Security Symposium. 2021: 3559-3576.
- [16] Huang H, Guo Y, Shi Q, et al. Beacon: Directed grey-box fuzzing with provable path pruning [C]//2022 IEEE Symposium on Security and Privacy (SP). IEEE, 2022: 36-50.
- [17] Zhu X, Wen S, Camtepe S, et al. Fuzzing: a survey for roadmap[J]. ACM Computing Surveys (CSUR), 2022, 54(11s): 1-36.
- [18] Writing an llvm pass[EB/OL]. [2023-05-13]. <https://llvm.org/docs/WritingAnLLVMPass.html/>.
- [19] Libxml2 is the xml c parser and toolkit developed for the gnome project[EB/OL]. [2023-05-13]. <https://gitlab.gnome.org/GNOME/libxml2/-/wikis/home>.