西北大学

信息科学与技术学院
本科生毕业设计/论文
文献综述报告

报告题目 嵌入式 JavaScript 引擎模糊测试方法研究

学　　号 2021117283

姓　　名 雷璟锟

导　　师 叶贵鑫

2025 年 01 月 07 日

一、研究背景及意义

随着物联网技术的发展，万物互联已经是必然的趋势。为了将物联网设备顺利接 入互联网，极具网络编程优势的 JavaScript 成为嵌入式开发的首选编程语言，因此也出现了许多优秀的嵌入式 JavaScript 引擎。JavaScript 引擎负责解释并执行 JavaScript 语言编写的程序，其功能的正确性决定了 JavaScript 程序是否能正确运行。JavaScript 引擎是执行 JavaScript 程序必不可少的软件，JavaScript 引擎的特殊性引起了学术界对其安全问题的广泛关注。 嵌入式 JavaScript 引擎除了需要关注功能正确性和安全性以外，还需要关注引擎性能的高效性。在计算能力强且电源供应便捷的服务器或者 PC 上，软件的性能缺陷可以用硬件优势予以弥补。然而，嵌入式 JavaScript 引擎运行在计算能力较弱且要求低功耗的设备上，使得嵌入式 JavaScript 引擎在实现时使用的算法需要具备高效的 特点。与桌面 JavaScript 引擎不同，嵌入式 JavaScript 引擎运行在嵌入式的设备上使 其无法使用即时编译技术，没有即时编译技术加持使得嵌入式 JavaScript 引擎的性能问题会更严重。嵌入式 JavaScript 引擎的性能缺陷会使程序的执行时间变长，增加嵌入式设备的能耗，使得嵌入式 JavaScript 引擎的性能缺陷检测变得更迫切。性能缺陷 通常具有隐蔽性强，影响时间长，修复缓慢的特点。其主要原因有两点：其一，引擎开发过程中程序员会更关注功能的正确性，性能问题通常没有得到足够的重视；其二，与功能测试不同，性能测试是通过判断其性能是否在可接受范围之内决定测试是否通 过，但是否可接受的定义具有较强的主观性。 差分模糊测试是一种软件缺陷自动检测技术，被广泛应用于编译器和解释器的自 动化测试。其主要用于检测 JavaScript 引擎的安全缺陷和功能性缺陷，目前为止尚未 有研究学者使用差分模糊测试对性能缺陷进行检测。经过对现有工作的分析发现，差分模糊测试难以检测性能缺陷的原因主要有三点。首先，测试性能缺陷的测试用例需要有具备语法正确 和 语义丰富两个条件。语法正确的测试用例是检测嵌入式 JavaScript 引擎性能缺陷的基本条件，语义丰富的测试用例才有可能触发嵌入式 JavaScript 引擎的性能缺陷。其次，测试用例中语句的执行时间与计时器精度不匹配， 语句执行次数太少未能暴露出引擎性能缺陷等问题，使得测试用例即使覆盖了存在性能问题的缺陷代码也难以使用自动化测试技术检测其性能缺陷。这是由于 JavaScript 除了诸如正则匹配之类的少数操作的执行时间可能会比较长以外，大部操作的执行时间都远小于 1ms，然而计时器的精度大于 1ms 使得计时器无法估量单个操作的执行时间。最后，如何自动客观地指定性能评价指标是具有挑战且有必要解决的难题。正因为这三个问题的存在，使得差分模糊测试没有被用于软件的性能缺陷检测。

二、国内外研究现状

模糊测试主要由四个阶段组成：测试用例生成阶段、程序执行阶段、程序运行状态监控阶段、崩溃分析阶段。测试用例生成阶段包括种子文件生成、变异、测试用例生成和测试用例筛选。种子文件是符合程序输入格式的原始样本，测试用例的产生就是通过选择不同的突变策略来在不同的位置突变种子文件，这意味着种子文件和突变策略的好坏在很大程度上决定了模糊测试的效率。然而，事实上，并不是所有的用例都是有效的，我们需要选择能够通过程序过滤器从而触发新路径或漏洞的测试用例。这个选择过程一般由定义的适应度函数(如代码覆盖率)指导。测试用例生成阶段可分为基于突变的策略和基于生成的策略以及两者混合的策略。基于突变的生成策略是在已知测试用例的基础上通过增删改等方法生成新的测试用例，它包含了上述四个过程。基于生成的生成策略是直接根据设定好的输入用例格式直接生成新的测试用例，可以不包含监控阶段。基于混合的生成策略是根据设定好的种子文件的格式生成种子，然后在这些种子文件上进行变异生成测试用例。程序运行状态监控阶段在程序运行时监控程序的状态，通过在程序运行的过程中收集相应的

信息反馈给测试用例生成阶段来指导测试样本的生成。崩溃分析阶段收集目标程序崩溃或报告错误时的相关信息，以供以后重播和分析。目前针对 JavaScript 引擎的模糊测试研究主要基于以下两个问题：（1）测试用例的有效性。JavaScript 引擎在解析 JavaScript 源码时，会严格检查代码是否符合语法语义的规范，当遇到不符合规范的代码时，解析将会终止，这也就使得模糊测试通过随机变异产生的测试用例无法触及 JavaScript 引擎的核心功能，也就难以达到理想的测试效果，而想要产生大量符合 JavaScript 引擎规范的源码，又将花费大量的时间，导致在一定程度上限制了缺陷检测的效率。（2）代码覆盖率。代码覆盖率是生成 JavaScript 代码质量的主要表现形式之一。JavaScript 引擎模糊测试中代码覆盖率有两种，一种是指引擎中执行的代码占引擎所有代码的比值，另一种是指测试用例中被执行的代码占测试用例所有代码的比值，一定程度上决定着前者。代码覆盖率一直是衡量缺陷检测能力的重要指标之一，只有当产生的测试用例覆盖足够多的代码之后，才有可能挖掘出软件中存在的缺陷，而 JavaScript 引擎为了实现多种 Web 的动态交互功能，保证浏览器的可靠运行，其结构变得十分复杂，目前主流的 JavaScript 引擎都有大量的代码，像 Firefox 的 SpiderMonkey 引擎中就存在 100 多万行代码，如何生成大量高质量的 JavaScript 代码是目前基于模糊测试方法的主要问题之一。针对上述两个问题，目前的研究工作主要分为俩个方面，（1）基于生成的模糊测试。其主要思想是通过人工或机器学习构建的输入语法模型来生成有效的测试用例，从而挖掘更深层次的漏洞。早期的 Peach2 根据 JavaScript 引擎的特征手工构建输入语法模型，生成的测试用例数量有限，有效性也较差，同时消耗大量的人工成本；JSfunfunzz3 通过引入 JavaScript 语法模块，能够生成大量随机且语法正确的测试用例额，并且发现出大量的 JavaScript 引擎漏洞，但研究人员仍需花费大量精力来构建语法模块才能保证测试用例的有效性，且其自动化程度也相对较低。CodeAIchemist 将 JavaScript 代码拆分为代码块的组合，同时为每段代码块设立独特的组合约束条件，再将代码块组合成新的 JavaScript 代码，从而生成符合语法的 JavaScript 代码，但也需要对 JavaScript 代码有较多的先验知识，代码质量在很大程度上依赖与组合约束条件的完备性。Comfort 则利用更为先进的自然语言模型 GPT-2，通过差分测试和语言规范来辅助测试用例的生成，进一步提高了测试用例的有效性和质量，同时也因为 GPT-2 庞大的参数导致生成速度缓慢。（2）基于变异的模糊测试。其主要思想是通过随机或者启发性策略改变初始种子输入来构造新的测试用例，不依赖生成模型，与基于生成的模糊测试相比，执行速度更快，并且目前的基于突变的模糊测试都引入了负反馈机制，进一步提高了代码覆盖率。AFL（AmericanFuzzyLop）1 是近几年来最具代表性的基于突变的模糊测试工具，如何利用 AFL 对 JavaScript 引擎进行深层次的漏洞挖掘，研究者们做出了许多努力。Superion 基于 AFL 引入一种语法感知的修剪策略，通过将 JavaScript 源码解析成 AST 后在树级进行修剪，此外还提出了基于字典的变异和基于树的变异两种编译策略，在一定程度上提高了代码覆盖率和漏洞挖掘能力；Deity 假设那些能引发漏洞的输入重组之后有更多的机会发现同一片代码区域的错误，将已经发现的漏洞及其 POC 作为初始语料库，在树和字典的层级进行变异，从而生成符合语法规则的代码来进一步找到更深层次的漏洞，这也就导致了在代码覆盖率上难以达到很好的效果。Safuzzer 提出了一种基于语义的模糊测试方式，通过基于语法规则的语法树变异及语义修复，保证了样本的执行成功率，实现了更高的代码覆盖率，但其样本集均来自与开发人员手工编写的，在覆盖程序路径上存在一定的局限性。Fuzzilli 将 JavaScript 代码转为一种中间表示，通过对中间变量的输入、输出、操作码进行变异产生测试用例，使生成的 JavaScript 代码能继承原有的数据流和控制流特征，降低代码错误率，也因此受限于语料库的质量。

当前，JavaScript 语言的快速发展为多种环境下的软件应用开发提供了许多便利，越来越多的 JavaScript 引擎如雨后春笋般出现。但与此同时，各 JavaScript 引擎内部 隐含的软件缺陷需要更加高效的方法予以揭示。

1. 参考文献

[1]姚厚友.面向嵌入式 JavaScript 引擎的差分模糊测试方法研究[D].2021.
[2]田洋.基于标准文档分析的 JavaScript 引擎缺陷检测方法研究[D].2021.
[3]孙力立,武成岗,许佳丽,等.脚本语言执行引擎的模糊测试技术综述[J].高技术通讯.2022,32(12).DOI:10.3772/j.issn.1002-0470.2022.12.002 .
[4]曹帅.基于类型推断的 JavaScript 引擎模糊测试方法研究[D].2020.
[5]卢凌. 面向 JavaScript 引擎报错机制的类别导向模糊测试方法[D]. 辽宁:大连理工大学,2023.
[6] 周阳. 基于模糊测试的 JavaScript 引擎缺陷检测方法[D]. 湖北:华中科技大学,2022.
[7]西北大学.一种基于类型推断的具有引导性的测试用例变异方法 :CN202010200651.0[P].2020-03-20.
[8] 程勇,秦丹,杨光. 针对 JavaScript 浏览器兼容性的变异测试方法[J]. 计算机应用 ,2017,37(4):1143-1148,1173. DOI:10.11772/j.issn.1001-9081.2017.04.1143.
[9] 刘艺玮. 基于深度学习的 JavaScript 引擎模糊测试方法研究[D]. 四川:电子科技大学,2024.
[10]余启洋, 桑楠, 郭文生. 嵌入式浏览器 JavaScript 引擎的研究与设计[J]. 计算机应用与软件, 2014, 5.
[11]王聪冲. 面向 JavaScript 解析引擎的模糊测试技术研究[D]. 江南大学, 2021.
[12]王允超, 王清贤, 丁文博. 语义感知的 JavaScript 引擎模糊测试技术研究[J]. 信息工程大学学报, 2020.
[13] 吴泽君,武泽慧,王允超,等. 基于自然语言处理的 JavaScript 引擎定向模糊测试技术[J]. 信息工程大学学报,2022,23(6):737-745. DOI:10.3969/j.issn.1671-0673.2022.06.014.
[14] SOYEON PARK, WEN XU, INSU YUN, et al. Fuzzing JavaScript Engines with Aspect-preserving Mutation[C]//2020 IEEE Symposium on Security and Privacy: IEEE Symposium on Security and Privacy (SP 2020), 18-21 May 2020, San Francisco, CA, USA.:Institute of Electrical and Electronics Engineers, 2020:1629-1642.
[15]Dinh S T, Cho H, Martin K, et al. Favocado: Fuzzing the Binding Code of JavaScript Engines Using Semantically Correct Test Cases[C]//NDSS. 2021.
[16]Xu H, Jiang Z, Wang Y, et al. Fuzzing JavaScript Engines with a Graph-based IR[C]//Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. 2024: 3734-3748.
[17]Groß S, Koch S, Bernhard L, et al. FUZZILLI: Fuzzing for JavaScript JIT Compiler Vulnerabilities[C]//NDSS. 2023.
[18]Wang J, Zhang Z, Liu S, et al. {FuzzJIT}:{Oracle-Enhanced} Fuzzing for {JavaScript} Engine {JIT} Compiler[C]//32nd USENIX Security Symposium (USENIX Security 23). 2023: 1865-1882.
[19]Bin, Zhang,Jiaxi, Ye,Xing, Bi,等.Ffuzz: Towards full system high cove

rage fuzz testing on binary executables.[J].PLoS ONE.2018,13(5).e0196733.DOI:10.1371/journal.pone.0196733 .
Klaus Greff,Rupesh K. Srivastava,Jan Koutník,等.LSTM: A Search Space Odyssey[J].IEEE Transactions on Neural Networks and Learning Systems",pubMed Id":"27411231.2017,28(10).2222-2232.DOI:10.1109/TNNLS.2016.2582924 .