Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

# Fuzzing JavaScript engines with a syntax-aware neural program model

Haoran Xu, Yongjun Wang [*], Zhiyuan Jiang, Shuhui Fan, Shaojing Fu, Peidai Xie

*College of Computer, National University of Defense Technology, China*

ARTICLE INFO

*Keywords:*
Fuzzing
JavaScript engines
Language model
Neural network
Grammar
Vocabulary

ABSTRACT

Neural network language modeling has become a remarkable approach in the generation of test cases for fuzzing JavaScript engines. Fuzzers built upon neural language models offer several advantages. They obviate the need for manually developing code generation rules, enable the extraction of patterns from high-quality seed sets, and exhibit commendable portability. Nevertheless, existing works confront challenges in three key aspects: diminished language modeling performance attributable to extensive vocabularies, potential semantic errors within generated test cases, and the limitation of black-box fuzzing, which fails to leverage the internal feedback from the target engine.

This paper proposes an innovative neural model-based grey-box fuzzing approach for JavaScript engines. We incorporate the context-free grammar of JavaScript into the neural language model to mitigate the challenges associated with extensive vocabularies, thereby enhancing the model's performance. Furthermore, to enhance the semantic validity of the generated test cases, we introduce semantic constraints into the mutation process. Notably, this work pioneers the integration of grey-box testing into a fuzzer built upon a neural language model, thereby enhancing the exploration of deep paths. Our prototype, PMFuzz, surpasses NNLM-based counterparts in both language modeling performance and test case generation capabilities. PMFuzz demonstrates a high level of competitiveness in exploring the software state space when compared to traditional coverage-guided grey-box fuzzers. In our evaluation, PMFuzz successfully identified 20 new defects within mainstream JS engines. Eight of them have been confirmed and fixed. Moreover, upon applying our method to C compilers, PMFuzz has revealed 11 new defects.

## 1. Introduction

JavaScript (JS) is the most popular programming language for implementing dynamic and interactive web sites. An astonishing 98.8% of websites execute JS on the client side (W3Techs, 2023). In addition to browsers, JS engines have also been migrated to other applications like server-side JS runtime environment (Dahl, 2009, 2018), PDF-related office software (Artifex Software, 2017), and IoT devices (Gavrin et al., 2015; cesanta, 2018) for running JS applications. As of 2022, JS has more code repositories than any other language on GitHub (Github, 2022) with no end in sight. However, a notable series of high-risk vulnerabilities (Kang, 2021; ProjectZero, 2022) have emerged in widely used JS engines, posing substantial security risks for billions of users worldwide. Adversaries chain successful attacks with an escape from the browser sandbox, gaining unauthorized privileges by crafting malicious web pages and enticing victims to access them (saelo, 2018). It is imperative to proactively uncover potential defects in JS engines to protect users against attacks.

Fuzzing is an effective automated bug discovery approach for JS engines (Mozilla, 2007; Holler et al., 2012; Veggalam et al., 2016; Han et al., 2019; Wang et al., 2019; Aschermann et al., 2019; Park et al., 2020; Wang et al., 2023; Groß et al., 2023). It generates diverse JS programs (test cases) to the target engine, with the aim of triggering unexpected behaviors such as assertion failures and crashes, and unexpected behavior usually means the existence of defect. Fuzzing has demonstrated its high efficiency in testing language processors, including JS engines and other compilers (Marcozzi et al., 2019). A large number of defects have been uncovered through fuzzing.

Recently, Neural Network Language Model (NNLM)-based approaches (Godefroid et al., 2017; Cummins et al., 2018; Liu et al., 2019) have been proposed for fuzzing input generation. They provide the following advantages: Firstly, the manual development of code for generating test programs is a time-consuming and labor-intensive endeavor (Cummins et al., 2018). Learning the language model of sample programs to generate new test cases effectively mitigates the development cost of the fuzzer. Secondly, as there is no requirement

* Corresponding author.
*E-mail addresses:* xuhaoran12@nudt.edu.cn (H. Xu), wangyongjun@nudt.edu.cn (Y. Wang), jzy@nudt.edu.cn (Z. Jiang), fanshuhui18@nudt.edu.cn (S. Fan), fushaojing@nudt.edu.cn (S. Fu), xpd2002@126.com (P. Xie).

https://doi.org/10.1016/j.cose.2024.103947
Received 3 January 2024; Received in revised form 20 May 2024; Accepted 4 June 2024
Available online 8 June 2024