Fuzzing Programs with Structure Aware Fuzzers

Darshana Das K TCR21CS020
Krishnahari P TCR21CS032
Libna Kuriakose T TCR21CS033
Parvathy C M TCR21CS049

BTech in Computer Science and Engineering

Guide: Dr. Ezudheen P
Supervisor: Dr. Rahul Gopinath

A project report submitted in fulfilment of
the requirements for the Mini Project in Semester 6

Department of Computer Science and Engineering
Government Engineering College Thrissur

3 April 2024

Abstract

Fuzzing is a widely used technique for finding vulnerabilities in software. In recent years, structure aware fuzzers have gained attention due to their ability to better understand the input structure of programs and generate more effective test cases. Fuzzing programs that handle structured binary inputs, like multimedia files, presents a difficult challenge due to the specific input format these programs require. While current fuzzers are versatile in their approach, they often lack effectiveness when it comes to targeting a particular format. This work aims to develop a structure aware fuzzer to improve the efficacy of fuzzing techniques.

Our tool, Binary Fuzzer takes a binary template (using Kaitai Struct) as input and utilises the fuzzing logic of libFuzzer. Kaitai Struct's capability to define complex data structures in a declarative manner makes it well-suited for use as input to our tool. This fuzzer addresses the limitations of traditional fuzzing methods by leveraging knowledge of the program's structure to guide the generation of input data. The advantage of a structure aware fuzzer lies in its ability to explore the input space more intelligently, leading to a higher probability of discovering critical software flaws.

Contents

CHAPTER 1

Introduction

Fuzzing has emerged as a critical technique for uncovering vulnerabilities in software programs, serving as a proactive measure to identify potential weaknesses before they are exploited by malicious actors. Traditional fuzzers operate by generating random or semi-random inputs to stress-test the resilience of a program. However, these conventional fuzzing methods often lack the sophistication to comprehend the underlying structure of the data they manipulate. Consequently, they may overlook certain types of bugs and vulnerabilities that lie dormant within the program's codebase.

To address this limitation, structure-aware fuzzing has gained prominence. By incorporating an understanding of the data's structure into the fuzzing process, these advanced techniques can generate test cases that effectively exercise different segments of a program. This nuanced approach has the potential to uncover a broader spectrum of bugs and vulnerabilities. One such approach involves leveraging Kaitai Struct, a declarative language tailored for describing binary data formats in a structured manner. With Kaitai Struct, fuzzers can utilize input in the form of KSY files, which precisely define the layout and organization of the data to be fuzzed.

Our tool, BinaryFuzzer, harnesses the capabilities of structure-aware fuzzing by accepting input in the form of KSY files. This methodology facilitates more intelligent and targeted fuzzing, thereby enhancing the likelihood of identifying vulnerabilities within software programs. Kaitai Struct files offer a concise and coherent means of articulating the format of binary data, streamlining the creation of fuzzing test cases that comprehensively cover specific elements of the data structure. Moreover, Kaitai Struct boasts several advantages, including portability, reusability, and ease of maintenance, which further enhance its utility in the fuzzing ecosystem.

In addition to Kaitai Struct, another pivotal component in the realm of fuzzing is libFuzzer—a library tailored for in-process fuzzing. LibFuzzer is adept at conducting fuzz testing on individual functions or modules within a program, allowing for granular examination of code segments. By integrating libFuzzer with KSY files, we can construct a robust framework for structured fuzzing that amalgamates the proficiency in understanding data structures with the exhaustive testing capabilities offered by libFuzzer. This synergy between structured fuzzing and in-depth testing empowers security researchers and developers to uncover and remediate security vulnerabilities more effectively.

The integration of Kaitai Struct and libFuzzer introduces a paradigm shift in the approach to fuzz testing, offering a potent combination of precision and comprehensiveness. By leveraging the descriptive power of KSY files, BinaryFuzzer can tailor test cases to target specific areas of a program's data structure, thereby maximizing the likelihood of triggering latent vulnerabilities. Meanwhile, the utilization of libFuzzer enables rigorous and systematic testing of individual code segments, ensuring thorough coverage and robust bug detection.

Furthermore, the adoption of Kaitai Struct and libFuzzer fosters a collaborative ecosystem wherein security researchers and developers can share and reuse fuzzing definitions and test cases, thereby accelerating the identification and resolution of vulnerabilities across diverse software projects. This collaborative ethos aligns with the broader principles of open-source development and information security, fostering innovation and collective advancement in the field of software security.

CHAPTER 2

Literature review

In recent years, there have been notable advancements in the field of fuzzing programs with structure-aware techniques, aimed at enhancing vulnerability discovery in software systems. These developments have introduced various sophisticated methodologies, each contributing unique technical intricacies and capabilities.

One significant contribution is FormatFuzzer, developed by Rafael Dutra, Rahul Gopinath, and Andreas Zeller from CISPA Helmholtz Center for Information Security. FormatFuzzer introduces a format-specific approach to fuzzing, utilizing binary templates to generate C++ code for parsing, mutating, and generating inputs. It effectively handles complex formats such as MP4 or ZIP files and integrates with format-agnostic fuzzers like AFL for intelligent mutations and seed evolution, thereby enhancing fuzzing outcomes.

Another notable advancement is presented in the study "Superion: Grammar-Aware Greybox Fuzzing for Structured Inputs" by Junjie Wang, Bihuan Chen, Lei Wei, and Yang Liu. Superion enhances traditional coverage-based greybox fuzzing by incorporating grammar awareness, facilitating more effective trimming and mutation of test inputs while maintaining their validity and structure. Experimental studies on XML and JavaScript engines demonstrate Superion's improved code coverage and bug-finding capabilities compared to AFL, uncovering 31 new bugs, including 21 vulnerabilities with 16 CVEs assigned.

In their paper titled "Fast Format-Aware Fuzzing for Structured Input Applications," the authors propose an efficient approach for vulnerability discovery in applications with structured input. Their method, called FFAFuzz, leverages input-to-state (I2S) dependencies and indirect dependency analysis. FFAFuzz significantly reduces time overhead, outperforming existing techniques like Redqueen and WEIZZ, making it a valuable contribution to enhancing security testing for structured input applications.

The paper "GREYONE: Data Flow Sensitive Fuzzing with Fuzzing-driven Taint Inference and Constraint Conformance" introduces GREYONE, a novel fuzzing approach combining dynamic taint analysis with structure-aware techniques. It significantly enhances code coverage and vulnerability discovery by inferring taint during fuzzing and incorporating structure-awareness. GREYONE outperforms state-of-the-art fuzzers, identifying 105 new security bugs, 41 of which are confirmed by CVE, demonstrating its effectiveness in identifying software vulnerabilities.

The research paper titled "Structure-Aware Mutations for Library Fuzzing" by Valentin Metz, conducted under the supervision of Fabian Kilger at the Chair for IT Security, introduces a custom, structure-aware mutator for AFL++. This mutator is specifically designed to generate input structures aligned with the expected format of AutoDriver, an automatic generator for fuzz drivers. By integrating this mutator, the study achieves expedited initial coverage exploration, deliberate utilization of AutoDriver-specific functionalities, and reduced crashes due to structurally invalid input.

2.1 Section

More text.

2.1.1 Subsection

Even more text.