Evaluation of New Advanced Fuzzers

Table of Contents

Overview of your Internship Assignment

Project objective

For my 3-month internship, I am tasked to evaluate a new library API fuzzer and develop additional functionalities to include more test cases and simulate real-life usage of the target library.

The fuzzer of interest, GraphFuzz[1], is built upon LibFuzzer and presents a novel idea of modelling the sequence of API calls through a dataflow graph. Unfortunately, this tool is relatively new -- developed in 2022, and fails to capture many characteristics of the C++ language. This limits its ability to generate test cases and emulate the various ways complex libraries are used in real life. Thus, we look to analyse the specific shortcomings of GraphFuzz, evaluate what capabilities could be appended to better fuzz libraries, and develop some of these features. From a technical perspective, we hope these enhancements allow higher code coverage and shorter time to spot known bugs in libraries.

Fuzzing

To start, I spent roughly 2 weeks learning the basics of fuzzing. Although my project focuses on library fuzzing, which is rather distinct from conventional fuzzing of standalone programs, this was needed to introduce me to fuzzing in general because I had no prior experience in this field. This entailed reading some articles that described what fuzzing is, how it works, and also what are the different fuzzing methods. I also went through several guided fuzzing exercises that were based on AFL++, a common fuzzer, to familiarise myself with the steps involved in fuzzing a program.

[1] Harrison Green and Thanassis Avgerinos. 2022. GraphFuzz: library API fuzzing with lifetime-aware dataflow graphs. In Proceedings of the 44th International Conference on Software Engineering (ICSE '22). Association for Computing Machinery, New York, NY, USA, 1070–1081. https://doi.org/10.1145/3510003.3510228

With some fundamental knowledge, I ventured into library API fuzzing and LibFuzzer. Due to the differences between how a library is used as opposed to a standalone program, the entire fuzzing process and the tools used also vary. In essence, fuzzing a library involves fuzzing multiple APIs, which means more test cases are required, and this is further complicated by dependencies between APIs, a common characteristic of complex libraries.

GraphFuzz Basics

GraphFuzz, the tool of interest in the project, is a relatively new library fuzzer that models the sequences of the APIs called. This makes it an intriguing subject because it drastically reduces the amount of effort required to fuzz a library. When using traditional library fuzzers like LibFuzzer, multiple harnesses have to be manually written to cover every API, incurring significant time and effort. Despite that, it is almost impossible to accommodate the dependencies amongst APIs.

Meanwhile, to use GraphFuzz, all the user needs to provide it with is the target library's source code, to be instrumented with code coverage capabilities. From that, GraphFuzz would try to deduce a schema of all the custom classes of the libraries and their methods. Albeit inaccurate and hence requiring manual revision, this is much more convenient and easier. GraphFuzz then builds a graph, which represents a series of API calls and can be thought of as the equivalent of a harness, passes that graph to a driver program, and collects coverage results and crashes if any. The process then repeats with thousands of different graphs, obtained by mutating the existing graphs. GraphFuzz currently offers 10 mutations, which could be applied to vary what APIs are called and the order in which they are called.

Evaluating GraphFuzz

After conducting some analysis of GraphFuzz mainly through its documentation and manual code review, I observed that it seemed to be lacking some utility that, if incorporated, could allow more flexibility in how the APIs are called and ultimately cover more test cases. To verify these speculations, I ran GraphFuzz on some toy programs and the results substantiated these hypotheses.

Specifically,

1. GraphFuzz does not capture class hierarchy information within the library source code. Consequently, it does not allow subclass objects to access inherited methods nor allow polymorphism in the graphs it generates. However, actual usage of the library allows for such cases and hence a need for GraphFuzz to accommodate class hierarchy and its implications.
2. GraphFuzz's policy of identifying object constructors and destructors are too rigid. Existing implementations only pinpoint APIs with identical names as their enclosing classes as constructors or destructors, excluding other functions that have the same effect as a constructor or destructor but are named differently.
3. GraphFuzz's proposed feature to curb crashes originating from improper usage of the library limits flexibility. The motivation is that simultaneous access and modification to objects dependent on one another could easily lead to crashes which terminate the execution of the test case pre-maturely and hinder fuzzing of the internals of the library. GraphFuzz suggests bundling up such objects into a single bundled object so that these objects cannot be used simultaneously. However, this is a hard restriction on how these objects can be used and hinders the discovery of bugs through inappropriate use of APIs but which does not immediately cause a crash.

Work in progress

As of the time of writing, I am working on implementing some functionality to tackle the class hierarchy problem detailed above. I am also trying to gain a better understanding of the C++ language and what other features can be introduced to GraphFuzz to better fuzz C++ libraries.

How the Project/ Research can be Applied to DSO/Division's Mission and Vision

The motivation of this project is to further enhance the fuzzing of libraries as a means to detect flaws in (open-source) libraries employed. This strongly aligns with DSO's mission to develop and provide technological capabilities to sharpen Singapore's national security. The results of this project can also be applied to any C/C++ libraries, and perhaps extended to other library fuzzers based on LibFuzzer.

Challenges Faced During the Internship

While I have had prior experience working on C++ codebases, this is my first experience with fuzzing and examining in detail the mechanisms of a library fuzzer. Hence, I took quite a significant amount of time familiarising myself with fuzzing concepts and frameworks, and brainstorming meaningful additions to GraphFuzz.

Furthermore, the lack of clear documentation on the experimental features in GraphFuzz, which were plenty, meant that some time was spent on manual code review to ascertain how those features work behind the scenes to achieve their intended function. Ultimately, I built my version of GraphFuzz to display some of the intermediate data referenced in the fuzzing process.

Lessons Learnt From Tasks Assigned

I have been exposed to many new concepts thus far. Apart from the obvious -- fuzzing, I have gained a better understanding of code instrumentation and how it varies depending on the specific use cases of the built program. Although the ultimate goal of this project is to allow easier detection of cyber security flaws in libraries, the technical knowledge and skills picked up are easily applicable to other domains, such as writing more efficient and secure code in future projects.

This project allows for much autonomy in that I could decide for myself what subtasks I would like to focus on and how to approach each of them as long as the overarching objective has been achieved. In the process, I have gained greater insights into the brainstorming and planning process behind research and development work, such as how various subtasks can be prioritised.

Positive Experiences in DSO

I have enjoyed my time at DSO. The tasks introduce new topics and pose sufficient challenges for me to think critically and to continuously learn new content, yet not too overwhelming to the extent that I am stuck and cannot contribute to the project. The culture and environment at DSO have been very conducive to both working and learning. My mentor and fellow interns have also been very approachable and willing to share their tips and experiences with me.

Changes Recommended

1. Air-conditioner
   It would be great if the aircon temperature could be raised slightly.

Meeting Initial Expectations

This internship has met my expectations of learning as much as possible and applying them to the work done. I have been exposed to new topics and the relevant tools, and given enough freedom to do my own exploration, decision-making, and milestone planning, but also been offered sufficient resources and guidance by my supervisor when needed.

Thoughts on Internship Supervisor

My supervisor has been very friendly and approachable. When asking for advice on problems that I am stuck on, he is always able to leverage his technical expertise to guide me through. That said, he also allowed me to explore different methods and test out my hypotheses before giving me more hints, encouraging me to do some independent thinking and learning which I thought facilitated my learning throughout the internship.

Thoughts on Future Career

This internship serves as another experience in the wide range of cybersecurity work done at DSO. It has been an enriching experience thus far, giving me insights into the day-to-day work of cybersecurity researchers from a development-focused perspective.