Compiler Fuzzing via Guided Value Mutation - Proposal

PIUS KRIEMLER* and JONAS DEGELO*, ETH Zürich, Switzerland

ACM Reference Format:
Pius Kriemler and Jonas Degelo. 2023. Compiler Fuzzing via Guided Value Mutation - Proposal. In . ACM, New York, NY, USA, 2 pages.

1 THE PROBLEM

The sheer complexity of compilers make their optimisations very efficient, but the bugs they introduce hard to catch. Despite the source code being correct, the compiled programs can be erroneous. This makes catching compiler bugs safety critical. Fuzzers have been successfully used to detect bugs in compilers [5]. As the semantics of such randomly "fuzzed" programs are difficult to control, recent research introduced the method of Equivalence Modulo Inputs [1]. With this approach, code is inserted to [4], or deleted from [2], a target program, whereas the semantics of a program is not changed. It not only allows to detect more elaborate compiler bugs, but also only requires one compiler to catch a bug. In this project, we are using a different semantic-changing approach, namely guided value mutation on the constant values of an input source code.

2 APPROACH

We mutate constant values of a seed program to create valid and runnable, but semantically different, mutants. The goal is to find large differences in number of assembly instructions produced by two different GCC compiler versions. We expect that newer compiler versions should not generate significantly more assembly instructions under the same compiler flags. Faster code not always uses less lines of assembly, see loop unrolling or inlining, but large line differences can give a hint that at least it's an interesting case to investigate.

In the course of this project, we will implement a pipeline in Python that parses a seed program and sequentially generates mutants, compiles them, and compares the generated assembly output. In a first phase, we mutate the constant values randomly. In a second phase, we mutate the values guided by discrete Bayesian optimisation [3]. We aim to efficiently find the mutant with the largest difference in assembly instructions, as the compilation time is costly.

3 WORK OUTLINE

- April 16th: Gather source code for seed programs and implement base pipeline
- April 23rd: Evaluate seed programs with random value mutation
- May 7th: Implement guiding algorithm
- May 14th: Evaluate seed programs with guided value mutation
- June 6th: Final report and buffer time

*Both authors contributed equally to this research.

REFERENCES

[1] Vu Le, Mehrdad Afshari, and Zhendong Su. 2014. Compiler validation via equivalence modulo inputs. *ACM SIGPLAN Notices* 49, 6 (June 2014), 216–226. https://doi.org/10.1145/2666356.2594334

[2] Vu Le, Chengnian Sun, and Zhendong Su. 2015. Finding deep compiler bugs via guided stochastic program mutation. *ACM SIGPLAN Notices* 50, 10 (Oct. 2015), 386–399. https://doi.org/10.1145/2858965.2814319

[3] Phuc Luong, Sunil Gupta, Dang Nguyen, Santu Rana, and Svetha Venkatesh. 2019. Bayesian Optimization with Discrete Variables. 473–484. https://doi.org/10.1007/978-3-030-35288-2_38

[4] Chengnian Sun, Vu Le, and Zhendong Su. 2016. Finding compiler bugs via live code mutation. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016)*. Association for Computing Machinery, New York, NY, USA, 849–863. https://doi.org/10.1145/2983990.2984038

[5] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. 2011. Finding and understanding bugs in C compilers. *ACM SIGPLAN Notices* 46, 6 (June 2011), 283–294. https://doi.org/10.1145/1993316.1993532