Melvin Moreno
COMP6710 – Dr. Akond Rahman
17 April 2023

Literature Review

Compiler Fuzzing: How Much Does It Matter?"
Michael Marcozzi, Qiyi Tang, Alastair F. Donaldson, and Cristian Cadar

Summary: This paper presents the first quantitative and qualitative study of the tangible impact of miscompilation bugs in a mature compiler. The authors emphasize the significance of compiler fuzzing in identifying and fixing critical bugs and vulnerabilities in compiler implementations, and the potential impact of compiler bugs on software security.

Applied Techniques: The paper evaluates the effectiveness of fuzzing. Fuzzing is defined as a technique that generates random inputs for finding bugs in compilers. The study uses a large-scale empirical evaluation of different fuzzing techniques applied to multiple compilers, including GCC and LLVM, to assess their bug-finding capabilities. Some of the fuzzing techniques used include grammar-based fuzzing, mutation-based fuzzing, and coverage-guided fuzzing.

Defects Identified: Bugs in compilers, including memory safety issues, undefined behavior, crashes, and other vulnerabilities.

A Comprehensive Study of Deep Learning Compiler Bugs"
Haoyan Ma, Junjie Chen, Yongqiang Tian, Shing-Chi Cheung, and Xiang Chen:

Summary: This paper presents the first systematic study of DL compiler bugs by analyzing 603 bugs arising in three popular DL compilers (i.e., TVM from Apache, Glow from Facebook, and nGraph from Intel). The authors analyzed the bugs according to their root causes, symptoms, and stages where they occurred during compilation and provided a series of valuable guidelines for future work on DL compiler bug detection and debugging.

Applied Techniques: The paper presents a comprehensive study of deep learning compiler bugs, focusing on popular deep learning frameworks TensorFlow and PyTorch. The study involves analyzing and categorizing bugs from real-world bug reports, evaluating their root causes, and identifying potential fixes.

Defects Identified: Bugs in deep learning compilers, including incorrect optimizations, incorrect code generation, memory management issues, and other programming errors. There are also high frequency occurrences of memory-related bugs, high prevalence of correctness bugs, and performance bugs.

An Empirical Study of Optimization of Bugs in GCC and LLVM
Zhide Zhou, Zhilei Ren, Guojun Gao, and He Jiang

Summary: This paper conducts an empirical study to investigate the characteristics of optimization bugs in two mainstream compilers, GCC and LLVM. The authors conduct an extensive analysis of bugs reports and code changes in the repositories of these compilers to investigate the characteristics, causes, and impacts of optimization-related bugs, and prpose recommendations for improving the quality and reliability of compiler optimizations.

Applied Techniques: The paper presents an empirical study of optimization-related bugs in two widely used compilers, GCC and LLVM. The study involves analyzing bug reports, categorizing the bugs based on their root causes and impacts, and identifying common patterns and trends in the bug.

Defects Identified: Bugs related to compiler optimizations, including incorrect code transformations, incorrect results crashes, and other opitimziation-related issues.

Well-Typed Programs Can Go Wrong: A Study of Typing-Related Bugs in JVM Compilers
Stefanos Chaliasos, Thodoris Sotiropoulos, Georgios-Petros Drosos, Charalambos Mitropoulos, Dimitris Mitropoulos, and Diomidis-Spinellis

Summary: This paper presents the first empirical study for understanding and characterizing typing-related compiler bugs. The authors investigate the occurrence, characteristics, and impacts of bugs related to type checking and type inference in JVM compilers, and propose recommendations for improving the quality of reliability of Java compilers.

Applied Techniques: The paper studies typing-related bugs in Java Virtual Machine (JVM) compilers by analyzing real-world bug reports and examining the root causes of the bugs. The study focuses on identifying bugs that result from type-related issues in JVM compliers.

Defects Identified: Typing-related bugs in JVM compilers, including issues with type inference, type checking, and other type-related errors.

Compiler Bug Isolation via Effective Witness Test Program Generation
Junjie Chen, Jiaqi Han, Peiyi Sun, Lingmin Zhang, Dan Hao, and Lu Zhang

Summary: This paper presents a new approach for isolating compiler bugs using effective witness program generation. The authors propose a technique that automatically generates test programs that serve as witnesses for compiler bugs, enabling effective bug isolation.

Applied Techniques: The paper presents a technique for isolating compiler bugs using effective witness test program generation. The approach involves automatically generating test programs that can trigger suspected compiler bugs, based on the bug reports and their associated witness programs. The generated test programs are used to reproduce and isolate the bugs for further analysis.

Defects Identified: Bugs in compilers, including crashes, incorrect code generation, incorrect optimizations, and other compiler-related issues