1. Unleashing the hidden power of compiler optimization on binary code difference: an empirical study

Xiaolei Ren (1); Ho, M. (1); Jiang Ming (1); Yu Lei (1); Li Li (2)
Source: *PLDI 2021: Proceedings of the 42nd SIGPLAN International Conference on Programming Language Design and Implementation*, p 142-57, 20 June 2021; **ISBN-13:** 978-1-4503-8391-2; **DOI:** 10.1145/3453483.3454035;
Conference: PLDI '21: 42nd SIGPLAN International Conference on Programming Language Design and Implementation, 20-25 June 2021, Virtual Event, Canada; **Sponsor:** SIGPLAN; **Publisher:** ACM, New York, NY, USA
Author affiliation: (1) University of Texas at Arlington, Arlington, TX, United States (2) Monash University, Melbourne, VIC, Australia
Abstract: Hunting binary code difference without source code (i.e., binary diffing) has compelling applications in software security. Due to the high variability of binary code, existing solutions have been driven towards measuring semantic similarities from syntactically different code. Since compiler optimization is the most common source contributing to binary code differences in syntax, testing the resilience against the changes caused by different compiler optimization settings has become a standard evaluation step for most binary diffing approaches. For example, 47 top-venue papers in the last 12 years compared different program versions compiled by default optimization levels (e.g., -Ox in GCC and LLVM). Although many of them claim they are immune to compiler transformations, it is yet unclear about their resistance to non-default optimization settings. Especially, we have observed that adversaries explored non-default compiler settings to amplify malware differences. This paper takes the first step to systematically studying the effectiveness of compiler optimization on binary code differences. We tailor search-based iterative compilation for the auto-tuning of binary code differences. We develop BinTuner to search near-optimal optimization sequences that can maximize the amount of binary code differences. We run BinTuner with GCC 10.2 and LLVM 11.0 on SPEC benchmarks (CPU2006 & CPU2017), Coreutils, and OpenSSL. Our experiments show that at the cost of 279 to 1,881 compilation iterations, BinTuner can find custom optimization sequences that are substantially better than the general -Ox settings. BinTuner's outputs seriously undermine prominent binary diffing tools' comparisons. In addition, the detection rate of the IoT malware variants tuned by BinTuner falls by more than 50%. Our findings paint a cautionary tale for security analysts that attackers have a new way to mutate malware code cost-effectively, and the research community needs to step back to reassess optimization-resistance evaluations. (0 refs)
Inspec controlled terms: invasive software - optimisation - program compilers - program diagnostics
Uncontrolled terms: binary code difference - syntactically different code - compiler optimization - compiler transformation - SPEC benchmark - compilation iteration - BinTuner - IoT malware variants - optimization-resistance evaluation
Classification Code: C6150C Compilers, interpreters and other processors - C6150G Diagnostic, testing, debugging and evaluating systems - C6130S Data security
IPC Code: G06F11/36 - G06F21/00 - G06F8/41
Treatment: Practical (PRA)
Database: Inspec
Data Provider: Engineering Village