Evaluating Synthetic Bugs

Joshua Bundt
Northeastern University
bundt.j@northeastern.edu

Andrew Fasano
Northeastern University
MIT Lincoln Laboratory
fasano@mit.edu

Brendan Dolan-Gavitt
New York University
brendandg@nyu.edu

William Robertson
Northeastern University
w.robertson@northeastern.edu

Tim Leek
MIT Lincoln Laboratory
tleek@ll.mit.edu

ABSTRACT

Fuzz testing has been used to find bugs in programs since the 1990s, but despite decades of dedicated research, there is still no consensus on which fuzzing techniques work best. One reason for this is the paucity of ground truth: bugs in real programs with known root causes and triggering inputs are difficult to collect at a meaningful scale. Bug injection technologies that add synthetic bugs into real programs seem to offer a solution, but the differences in finding these synthetic bugs versus organic bugs have not previously been explored at a large scale. Using over 80 years of CPU time, we ran eight fuzzers across 20 targets from the Rode0day bug-finding competition and the LAVA-M corpus. Experiments were standardized with respect to compute resources and metrics gathered. These experiments show differences in fuzzer performance as well as the impact of various configuration options. For instance, it is clear that integrating symbolic execution with mutational fuzzing is very effective and that using dictionaries improves performance. Other conclusions are less clear-cut; for example, no one fuzzer beat all others on all tests. It is noteworthy that no fuzzer found any organic bugs (i.e., one reported in a CVE), despite 50 such bugs being available for discovery in the fuzzing corpus. A close analysis of results revealed a possible explanation: a dramatic difference between where synthetic and organic bugs live with respect to the "main path" discovered by fuzzers. We find that recent updates to bug injection systems have made synthetic bugs more difficult to discover, but they are still significantly easier to find than organic bugs in our target programs. Finally, this study identifies flaws in bug injection techniques and suggests a number of axes along which synthetic bugs should be improved.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Department of Defense under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense. Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

ASIA CCS '21, June 7–11, 2021, Hong Kong, Hong Kong.

ACM ISBN 978-1-4503-8287-8/21/06.
https://doi.org/10.1145/3433210.3453096

CCS CONCEPTS

• **Security and privacy** → **Software security engineering**; • **Software and its engineering** → *Software defect analysis.*

KEYWORDS

Fuzzing; synthetic bugs; evaluation

ACM Reference Format:
Joshua Bundt, Andrew Fasano, Brendan Dolan-Gavitt, William Robertson, and Tim Leek. 2021. Evaluating Synthetic Bugs. In *2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3433210.3453096

1 INTRODUCTION

Fuzz testing, or fuzzing, is currently one of the best techniques for vulnerability discovery. Since its introduction in 1990 [31], a variety of fuzzing techniques have been explored including grammar-based fuzzing [17]; fuzzing combined with symbolic execution and SMT solvers [42]; taint-based fuzzing [15]; and fuzzing with neural networks [39]. Today, fuzzing is used widely and at scale in industry (e.g., Sage [18], ClusterFuzz [19], and OSS-Fuzz [22]). We refer interested readers to a recent survey on fuzzing techniques for a comprehensive overview of the field [29].

Automated vulnerability discovery is essential to both software developers interested in deploying secure software as well as hackers interested in exploiting software. As such, there is a clear need to understand what vulnerability discovery techniques actually work in practice and in which contexts. Furthermore, accurately quantifying and describing the performance of novel approaches to vulnerability discovery tools and techniques is necessary to advance the state of the art.

Fortunately, significant prior work has laid out guidelines and exposed pitfalls concerning how fuzzers should be evaluated [26]. A critical component of fuzzer evaluations is the "bug corpus," which has historically been a combination of previously-discovered bugs and new (0-day) discoveries attributed to the technique under evaluation. In 2016, LAVA introduced synthetic bug generation, in part, to overcome the limitations of relying on known vulnerabilities for fuzzer evaluation [11].

In the years since the release of LAVA and its most well-known benchmark corpus, LAVA-M, a large body of work has used synthetic bug injection in their evaluations. However, it is unclear that LAVA bugs are representative of organic (i.e., non-synthetic) bugs. Since LAVA bugs are commonly found by modern fuzzers while