

Abstract: Investigating the Efficacy of Fuzz Testing

Philipp Theodor Josef Görz

In this thesis, we explore different aspects of fuzzer efficacy. Fuzzers have become a popular and practical automated testing tool, especially for security vulnerabilities. Even with the popularity of fuzzing, we are lacking results as to the real bug-detection capability of fuzzers. Indeed, even though much research has been done on fuzzer benchmarks, we are still lacking a benchmark that is truly representative. We have not systematically compared fuzzers to static analysis tools. And, we have not studied the longevity of fuzzer integrations over time. This thesis aims to investigate these gaps.

In software testing, the gold standard for evaluating the quality of a test is to use mua, which evaluates a test's ability to detect synthetic bugs. These synthetic bugs are constructed in a way to create a large and diverse set of faults and if these are not detected, we can expect the test to fail to detect real bugs. However, the computational cost of traditional mua is prohibitive for benchmarking fuzzers. Thus, we implement optimizations that make benchmarking fuzzers feasible. Based on the resulting benchmark, we perform an initial evaluation and a follow-up fuzzer competition demonstrating the practicality of this benchmark.

Next, we compare the performance of fuzzers to static analysis tools based on over 100 known security vulnerabilities in C/C++ programs. We quantitatively compare the number of bugs detected as well as the number of inputs and reports required to assess to be able to act on these reported bugs. We find that there is little overlap in the bugs detected, while not entirely surprising, this provides strong justification for employing both approaches when testing. Furthermore, we find that one fuzzer and one static analysis tool already detect most bugs in their respective group. We can also confirm the common knowledge of an overwhelming amount of false positive reports by static analysis tools for existing projects, whereas fuzzer produce more actionable results.

Finally, we study the bug-detection and coverage performance of fuzzers as the projects they test age. To use a fuzzer, some glue code connecting the inputs generated by the fuzzer and the program under test is required—the so-called fuzz harness. If this harness is not maintained, it can get out of sync with the project, possibly causing a degradation of fuzzer performance. We study this effect based on historical data of a large scale fuzzing effort maintained by google. However, we find a surprising longevity of fuzzer performance, as long as the fuzz harness of the project still builds. Still, there are many cases of harness degradation, which we study and categorize. We conclude this effort by contributing metrics for detecting harness degradation.

Kurzfassung: Investigating the Efficacy of Fuzz Testing

Philipp Theodor Josef Görz

In dieser Dissertation werden verschiedene Aspekte der Effektivität von Fuzzern untersucht. Fuzzer haben sich als beliebtes und praktisches automatisiertes Software-Testwerkzeug etabliert, insbesondere zum Finden von Sicherheitslücken. Trotz ihrer Popularität fehlen uns jedoch Ergebnisse der tatsächlichen Fähigkeit von Fuzzern Sicherheitslücken zu finden. Als Teil dieser Dissertation untersuchen wir drei Bereiche. Erstens entwickeln wir einen Fuzzer-Benchmark mit minimaler Voreingenommenheit. Zweitens vergleichen wir Fuzzer mit statischen Analysetools. Drittens untersuchen wir die Langlebigkeit von Fuzzer-Integrationen.

Für den ersten Punkt entwickeln wir einen Fuzzer-Benchmark basierend auf “Mutation Analysis”. Dies ist der Goldstandard zur Bewertung der Qualität von Software-Tests. Tests werden darauf untersucht synthetische Fehler zu erkennen welche so konstruiert werden, dass sie eine große und vielfältige Art von Fehlern widerspiegeln. Falls diese nicht erkannt werden, können wir erwarten, dass die Tests auch keine echten Fehler erkennen. Allerdings sind die Kosten traditioneller “Mutation Analysis” jedoch prohibitiv hoch um naiv einen Fuzzer-Benchmark zu erstellen. Wir erforschen notwendige Optimierungen um diesen Ansatz praktikabel zu machen. Mit dem resultierenden Benchmark führen wir einen Fuzzer-Wettbewerb durch.

Für den zweiten Punkt vergleichen wir die Fuzzern mit statischen Analysetools anhand von über 100 bekannten Sicherheitslücken in C/C++-Programmen. Wir vergleichen quantitativ die Anzahl der erkannten Fehler sowie die Anzahl der Eingaben und Berichte, die erforderlich sind, um die gemeldeten Fehler praktisch verwenden zu können. Dabei stellen wir fest, dass es nur geringe Überschneidungen bei den erkannten Fehlern gibt. Obwohl dies nicht völlig überraschend ist, liefert es eine starke Begründung dafür, beide Ansätze für Software-Tests zu verwenden. Darüber hinaus stellen wir fest, dass ein Fuzzer und ein statisches Analysetool bereits die meisten Fehler in ihrer jeweiligen Gruppe erkennen. Wir können auch die allgemeine Erkenntnis bestätigen, dass statische Analysetools für bestehende Projekte eine überwältigende Anzahl von Fehlalarmen erzeugen, während Fuzzer handlungsrelevantere Ergebnisse liefern.

Abschließend untersuchen wir die Fehlererkennungs- und Abdeckungsleistung von Fuzzern, über einen langen Zeitraum von Projekten. Um einen Fuzzer zu verwenden, ist etwas Verbindungslogik erforderlich, die die vom Fuzzer generierten Eingaben mit dem zu testenden Programm verbindet – der sogenannte Fuzz-Harness. Wenn dieser Harness nicht gepflegt wird, kann es möglicherweise zu einer Verschlechterung der Fuzzer-Leistung führen. Wir untersuchen diesen Effekt anhand historischer Daten eines groß angelegten Fuzzing-Projekts, das von Google gepflegt wird. Überraschenderweise stellen wir jedoch eine erstaunliche Langlebigkeit der Fuzzer-Leistung fest. Dennoch gibt es viele Fälle von Harness-Verschlechterung, die wir untersuchen und kategorisieren. Wir schließen diese Untersuchung ab, indem wir Metriken zur Erkennung von Harness-Verschlechterung beitragen.