Fuzzing Through the Ages

Valentin Huber

November 6, 2023

Contents

1 Random notes

- "Today, testing is the primary way to check the correctness of software. Billions of dollars are spent on testing in the software industry, as testing usually accounts for about 50% of the cost of software development. It was recently estimated that software failures currently cost the US economy alone about $60 billion every year, and that improvements in software testing infrastructure might save one-third of this cost."[1]

2 Fuzzing types

2.1 Random input generation

2.1.1 An Empirical Study of the Reliability of UNIX Utilities (1990)

- [2]
- OG Fuzzing paper
- Started because in a stormy night, electrical interference on a dial-up connection
- Authors were surprised by amount of crashes, and artificially produced those.
- Generates random data (all chars/only printable chars, with or without NULL), throws them against a program
- Were able to crash or hang between 24 and 33% of programs on different UNIX systems
- Different error categories: pointer and array errors, unchecked return codes, input functions, sub-processes, interaction effects, bad error handling, signed characters, race conditions and undetermined.

2.2 Symbolic Execution

2.2.1 DART (2005)

- [1]
- Automated extraction of interface and env based on static source-code parsing
- Starts with random input, then uses symbex (without calling it symbex) to choose a different path
- Introduces a lot of concepts that I understand to be base level for symbex
- Has a unclear distinction to symbex, argues that symbex is stuck at expressions that aren't an issue with the symbex I know
- Concolic execution, fallback on concrete value whenever stuck

- Works on C code
- Positioned against static code analysis, which produces a lot of false positives while errors reported by DART are "trivially sound"[1]
- Run on a Pentium III 800MHz
- "As illustrated by the examples in Section 2, DART is able to alleviate some of the limitations of symbolic execution by exploiting dynamic information obtained from a concrete execution matching the symbolic constraints, by using dynamic test generation, and by instrumenting the program to check whether the input values generated next have the expected effect on the program."[1]

2.2.2 SAGE (2008)

- [3]
- First Whitebox Fuzzing paper so far.
- Developed at Microsoft.
- Does minor optimization to be able to perform partial symbex
- New invention: "Generational Search" — flips every branching condition after a symbex run to test in the next run, thus requiring fewer symbex runs overall.
- Uses concolic symbex whenever it gets too complex (i.e. interaction with the environment). It then checks whether the expected execution path is actually chosen and if not recovers (so-called "divergence").
- Runs on x86, Windows, file-reading applications.
- Found some vulnerabilities in media parsing engines and Office 2007.
- Further findings: symbex is slow (duh), at least two orders of magnitude compared to concrete execution.
- Divergences are common (60% of runs). This is because a lot of instructions were concretized to help with performance.
- No clear correlation between coverage and crashes, only weak effect when using a block coverage based heuristic to choose next execution.

References

[1] P. Godefroid, N. Klarlund, and K. Sen, "Dart: Directed automated random testing," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, (New York, NY, USA), p. 213–223, Association for Computing Machinery, 2005.

[2] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Commun. ACM*, vol. 33, p. 32–44, dec 1990.

[3] P. Godefroid, M. Y. Levin, and D. A. Molnar, "Automated whitebox fuzz testing," in *Network and Distributed System Security Symposium*, 2008.