Directed Greybox Fuzzing

Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, Abhik Roychoudhury

Writer: Kai Li

1 Summary

This paper proposed a Directed Greybox Fuzzing (DGF) technique called **AFLGo** [1] which generates inputs with the objective of reaching a given set of target program locations efficiently. The motivation comes from that Directed symbolic exection is not efficient in generating valid input that can reach to the target location in a program due to the heavy-weight program analysis. **AFLGo** works by casting reachability as an optimization problem and employ a specific meta-heuristic to minimize the distance of the generated seeds to the targets. Specifically, at complilation time, it computes the *seed distance* by computing the distance of each basic block to the targets, while this measurement is inter-procedural, to reduce the complexity, it proposed a novel measurement startegy based on call-graph (CG) and control-flow-graph (CFG) that after computing the function-level target distance in CG, it computes basic block level target distance distance to the call sites, within the same intra-procedural CFG. Then at runtime, **AFLGo** aggregates the distance values of each exercised basic block to compute the seed distance as their mean. The meta-heuristic that DGF employs to minimize seed distance is called Simulated Annealing and is implemented as power schedule which controls the energy of all seeds. A seed's energy specifies the time spent fuzzing the seed. By moving the analysis to compile-time, overhead at runtime is minimized. **AFLGo** is implemented in four components, the Graph Extractor, the Distance Calculator, the Instrumentor, and the Fuzzer.

- The AFLGo Graph Extractor (GE) generates the call graph (CG) and the relevant control-flow graphs (CFGs) and is implemented as extension of the AFL LLVM pass.
- The AFLGo Distance Calculator (DC) takes the call graph and each intra-procedural control-flow to compute the inter-proce- dural distance for each basic block and is implemented as a Python script that uses the networkx package for parsing the graphs and for shortest distance com- putation according to Djikstra's algorithm.
- The AFLGo Instrumentor takes the Basic block distance file and instruments each basic block in the target binary and is implemented as an extension of the AFL LLVM pass.
- The AFLGo Fuzzer is implemented into AFL version 2.40b. It fuzzes the instrumented binary according to the annealing based power schedule.

The evaluation result on Patching testing shows that **AFLGo** covers 13% more previously uncovered changed basic blocks than Katch (directed symbolic exection based fuzzer) and **AFLGo** found 13 previously unreported bugs in addition to 4 of the 7 bugs that were found by Katch. On Continuous fuzzing, **AFLGo** discovered 26 distinct bugs in seven security-critical open-source projects compared to OSS-Fuzz. On Crash reproduction, **AFLGo** is 3 to 11 times faster than AFL.

2 Strengths and Weaknesses

2.1 Strengths

1. The noval measurement strategy of distance that is inter-procedural, accounts for multiple targets at once, can be effectively pre-computed at instrumentation-time, and is efficiently derived at runtime.
2. The implementation based on AFLGo is public available and is scalable to test different programs, in addition to integration of AFLGo as patch testing tool into the fully automated toolchain of OSS-Fuzz.
3. Large-scale evaluation of the efficacy and utility of directed greybox fuzzing as patch testing and crash reproduction tool.

2.2 Weakness

1. External validity and notably generality. The evaluation results may not hold for subjects that the paper did not test, and a comparison with a directed whitebox fuzzer other than Katch or BugRedux might turn out differently.
2. Internal validity. AFLGo may not faithfully implement the technique presented here.
3. Construct validity. An empirical evaluation is always comparing only the implementations of two concepts rather than the concepts them- selves. Improving the efficiency or extending the search space of a fuzzer may only be a question of "engineering effort" that is unre- lated to the concept

References

[1] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. Directed greybox fuzzing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 2329–2344, New York, NY, USA, 2017. Association for Computing Machinery.