First, Fuzz the Mutants

Michael Shell
Georgia Institute of Technology
someemail@somedomain.com

Homer Simpson
Twentieth Century Fox
homer@thesimpsons.com

James Kirk
and Montgomery Scott
Starfleet Academy
someemail@somedomain.com

***Abstract*—The abstract goes here.**

I. Introduction

A. Mutation Testing

Mutation testing [5], [3], [4] is an approach to evaluating and improving tests. Mutation testing introduces small syntactic changes into a program, under the assumption that if the original program was correct, then a program with slightly different semantics will be incorrect, and should be detected by effective tests. Mutation testing is used in software engineering research, occasionally in industry at-scale, and in some critical open-source work [1], [7], [2].

A mutation testing approach is defined by a set of mutation operators. Such operators vary widely in the literature, though a few, such as deleting a small portion of code (such as a statement), negating a conditonal, or replacing arithmetic and relational operations (e.g., changing + to − or == to <=), are very widely used.

II. Preliminary Experiments

Coverage differences were not statistically significant by Mann Whitney U test, but bug count differences between both methods and AFL without mutants were significant with p-value < 0.006.

III. Related Work

The most closely related work is the the T-Fuzz approach [6], which focused specifically on removing sanity checks in programs in order to fuzz more deeply. Our approach is motivated in part by the desire to remove sanity checks, but uses a more general and lightweight approach. T-Fuzz used dynamic analsyis to identify sanity checks, while we simply trust that program mutants will include many (or most) sanity checks. Moreover, when a sanity check is hard to identify, but implemented by a function call, statement deletion mutants may in effect remove it where T-Fuzz will not. Our approach also introduces changes that are not within the domain of T-Fuzz, e.g., changing conditions to include one-off values. Finally, T-Fuzz worked around the fact that inputs for the modified program are not inputs for the real program under test using a symbolic exeuction step, while we simply hand the inputs generated for mutants to a fuzzer and trust a good fuzzer to make use of these "hints" to find inputs for the real program, if they are close enough to be useful.

IV. Conclusions

References

[1] I. Ahmed, C. Jensen, A. Groce, and P. E. McKenney, "Applying mutation analysis on kernel test suites: an experience report," in *International Workshop on Mutation Analysis*, March 2017, pp. 110–115.

[2] M. Beller, C.-P. Wong, J. Bader, A. Scott, M. Machalica, S. Chandra, and E. Meijer, "What it would take to use mutation testing in industry—a study at facebook," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021, pp. 268–277.

[3] T. Budd, R. J. Lipton, R. A. DeMillo, and F. G. Sayward, *Mutation analysis*. Yale University, Department of Computer Science, 1979.

[4] R. J. Lipton, R. A. DeMillo, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[5] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, "Mutation testing advances: an analysis and survey," in *Advances in Computers*. Elsevier, 2019, vol. 112, pp. 275–378.

[6] H. Peng, Y. Shoshitaishvili, and M. Payer, "T-Fuzz: Fuzzing by program transformation," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 697–710.

[7] G. Petrović and M. Ivanković, "State of mutation testing at google," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 163–171. [Online]. Available: https://doi.org/10.1145/3183519.3183521

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.23xxx
www.ndss-symposium.org

	Bugs			Stmt Cov			Branch Cov		
Method	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
AFL on program only	3	5	4.2	79.86%	84.37%	81.73%	78.36%	81.35%	80.40%
AFL on random mutants, non-cumulative	6	7	6.4	80.04%	84.90%	81.70%	79.85%	82.58%	80.70%
AFL on random mutants, cumulative/sequential	6	7	6.2	80.21%	84.90%	81.77%	80.10%	82.34%	80.90%

TABLE I. RESULTS FOR PRELIMINARY EXPERIMENTS