Comparison of various testing tools on GNU Core Utils

Rohith Mukku (14402)
Advisor: Subhajit Roy

November 2017

Abstract

Software testing is an important and necessary task so as to avoid crashes and bugs. There are several methods and two of these are symbolic execution and fuzzing. In this UGP, we look at these two methods - how they work and are different from each other. We also compare them with the tests run on GNU Coreutils.

1 Introduction

Software testing plays a very important role when distributing software packages. They help to find the bugs, test the correctness of the programs. In this project, our target programs are GNU Core Utils, a collection of every day's basic tools like ls, cat, tac, rm, etc.

There are various techniques in software testing and two of them are: Symbolic Execution and Fuzzing. KLEE[1] uses symbolic execution[2] while American Fuzzy Lop (AFL) does fuzzing[3]. Symbolic execution explores conditional paths by creating a control flow graph. Fuzzing tests by feeding random inputs and is usually used for detecting deadlocks, memory leaks.

2 Background

2.1 Symbolic Execution

In symbolic execution, the analysis of the target program is done by getting the list of possible points (or the conditional statements) where control flow path can split into two branches. It denotes a symbol at those points, forms constraints and explores all possible conditions by solving the constraints.

2.2 Fuzzing

In fuzzing, a Black-Box software testing technique, the target program is given random inputs in-order to find bugs. Having new inputs comes under Fuzzer Generator, where there are various methods like mutative, generative, etc. It doesn't care about what is in the target program (implementation part of the program).

3 Experiment

We have completed testing the 89 GNU tools with KLEE.

3.1 Implementation & Analysis

KLEE

Command used to run:

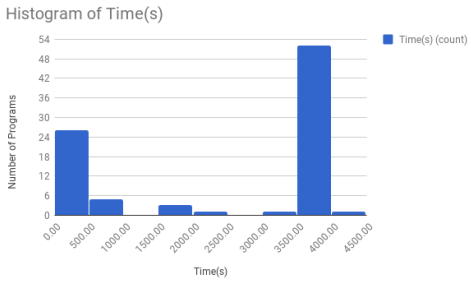
```
# klee <options> program.bc -sym-args 0 1 10 -sym-args 0 2 2 -sym-files 1 8 -sym-stdin 8 -sym-stdout
```

To collect statistics:

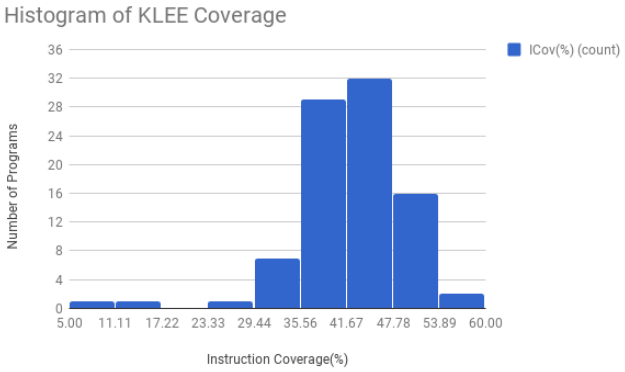
```
# klee-stats klee-output/
```

Time allotted to test each program was given as 1 hour.

(a) Number of instructions executed



(b) Time taken for all programs



(c) Instruction coverage

Figure 1: KLEE results

The case where klee exceeds 3600s can be due to no bound limit on the number of states to be explored[1].

[1] https://security.cse.iitk.ac.in/sites/default/files/11907299.pdf

AFL

The command used:

```
# afl-fuzz -i testcases/ -o output-dir/ path/to/program @@
```

AFL needs to be given some initial input and an output directory with some other options. It runs on the initial inputs and random mutation takes place on which the tests are run. The target program needs to be modified in a way to read inputs from file (input file in the input directory in this case). We couldn't test it correctly, as there was error in modifying the source of the GNU Core Utils. Below are some results from testing on some programs (these are not the final results).

	cat	**ls**	**mkdir**	**echo**	**mv**	**rm**
Cycles done	673899	46513	1885594	2879585	795885	2324702
Bitmap Coverage	0.11%	0.48%	0.20%	0.06%	0.14%	0.07%
Total paths	2	24	1	1	1	1
Unique Hangs	0	0	0	0	0	0
Unique Crashes	0	0	0	0	0	0

Table 1: AFL results

The above table is incomplete.

4 Future Work

The original experiment was to mainly compare KLEE, Crest (Concoling testing tool) which used symbolic execution with AFL, libFuzzer which used fuzzing techniques. We could extend to those tools and get a more detailed study.

Acknowledgements

I would like to express my thanks of gratitude to prof. Subhajit Roy who gave me the opportunity to do this project and helping me in learning new things. Secondly I would also like to thank Awanish Pandey for helping a lot in setting up tools for this project within the limited time frame.

Related Links

- `http://klee.github.io/docs/coreutils-experiments/`
- `http://lcamtuf.coredump.cx/afl/`

References

[1] C. Cadar, D. Dunbar, and D. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, (Berkeley, CA, USA), pp. 209–224, USENIX Association, 2008.

[2] J. C. King, "Symbolic execution and program testing," *Commun. ACM*, vol. 19, pp. 385–394, July 1976.

[3] R. McNally, K. Yiu, D. Grove, and D. Gerhardy, "Fuzzing: The State of the Art," Tech. Rep. DSTO–TN–1043, Defence Science and Technology Organisation, PO Box 1500 Edinburgh, South Australia 5111, Australia, Feb. 2012.