AI for Software Testing and Reverse Engineering Lab 2

INTRODUCTION

There are two main approaches for automated generation of software inputs or tests: symbolic and population based.

Symbolic generation of software tests relies on the ability of modern SMT solvers to quickly solve satisfiability problems for propositional logic with arithmetic constraints. By constructing such formulas for code-paths, symbolic execution engines construct the inputs required to reach a new piece of code deterministically. They then use some form of backtracking to try to reach full branch coverage.

Population-based methods use a guided random process for constructing new inputs/tests that try to trigger new code branches. They recombine previously tried inputs/tests guided by a heuristic that measures the proximity to conditions that trigger new branches.

In this Lab, you will construct and experiment the symbolic approach, understand how it works, learn how to use them, and discover its strengths and weaknesses.

LEARNING OUTCOMES

After completing this assignment, you will be able to:

- *Build path constraints and forward these to an SMT solver (such as Z3 https://github.com/Z3Prover/z3)*
- *Build search procedures that use symbolic execution to trigger new code branches*
- *Use and understand modern symbolic execution engines such as KLEE (https://klee.github.io/)*
- *Symbolically execute programs, analyze the generated paths and performance.*

This task is technically challenging. It is highly recommended to start early.

INSTRUCTIONS

For this assignment, we will again use the RERS Reachability problems. Similar to the previous assignment, we have provided you with the instrumentation and you should only need modify *SymbolicExecutionLab.java* for this assignment. You might also need the code that you have written for your fuzzer in the previous assignment. Before you begin with this assignment, make sure you have pulled the latest changes from the *JavaInstrumenation* repository:

- First navigate to the *JavaInstrumentation* folder.
- Then pull the latest changes by running the following command: `git pull`

Task 1: symbolic/concolic execution

The goal of this task is to give you in-depth experience with the inner workings of symbolic and/or concolic execution. Symbolic execution works by monitoring for branches, constructing a path constraint, and forwarding this to an SMT solver. For this task, you will need to expand the *SymbolicExecutionLab.java* file. It contains some basic functionality, i.e., how to call the Z3 SMT solver, how to add constraints to the model (constraint set), but it is incomplete. You have to add code for the missing operators, variable assignments (remember to use single static assignment) and decide how to use the information in the *encounteredNewBranch* method.

The *encounteredNewBranch* method is called every time a *myIf* is called and should be used to call the Z3 solver. You will have to write the logic that calls the solver together with the path constraint that you have constructed. In your implementation you should also handle the solutions that have been found by the solver, i.e., what can you do with the solutions that the solver has found for the path constraint that you have constructed? Before beginning with this assignment, make sure you have read the instructions on how to instrument the RERS problem for this lab assignment. The instructions can be found on the README of the GitHub repository:

- https://github.com/apanichella/JavaInstrumentation/blob/main/README.md

As we can use the solutions of a path constraint to check whether we can cover new branches, we can also use it to implement a new search strategy that uses symbolic execution to guide its input generation. Implement a new search strategy for your fuzzer that uses symbolic execution by tracking which unique branches are satisfiable and which ones are not satisfiable, together with the corresponding input traces that were found by the SMT solver. Use the traces that were found by the SMT solver to guide your fuzzer to generate new inputs that can be used to trigger new branches. Just like in the first lab assignment, plot the convergence graph (#unique error codes vs time) for this new search strategy and compare it to the results that you have achieved in the previous assignment. Which strategy do you think is better? Write down all your findings in your report.

Task 2: KLEE vs AFL

KLEE is a state-of-the-art symbolic execution engine based on LLVM. We have already installed KLEE for you on the docker container.

For instructions on how to get KLEE working on the RERS problems, please have a look at this document in the repository:

- https://github.com/apanichella/JavaInstrumentation/blob/main/docs/symbolic_rers.md

Your task is to compare the performance of KLEE and AFL by running them on the Reachability problems (11-19). Give them the same amount of runtime and analyze the obtained reachability targets. Your focus is to highlight and explain the differences between fuzzing and concolic execution. Address the following elements in your report:

1. Descriptions of AFL and KLEE. How do they work? What are they good at?
2. What are the results that you got for each state-of-the-art tool on the RERS 2020 Reachability problems?
3. An analysis of the results.

The Driller paper (see Resources) contains good examples of result presentations. Use these for inspiration, although some will be too detailed to include in your report, be selective!

RESOURCES

Slides from Lectures 3, 4

Study:

1. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. Cristian Cadar, Daniel Dunbar, Dawson Engler
2. SAGE: whitebox fuzzing for security testing. Godefroid, Patrice, Michael Y. Levin, and David Molnar.
3. Driller: Augmenting Fuzzing Through Selective Symbolic Execution. Stephens, Nick and Grosen, John and Salls, Christopher and Dutcher, Andrew and Wang, Ruoyu and Corbetta, Jacopo and Shoshitaishvili, Yan and Kruegel, Christopher and Vigna, Giovanni
4. Symbolic execution for software testing: three decades later. Cadar, Cristian, and Koushik Sen.

Concolic Fuzzer Chapter from the FuzzingBook https://www.fuzzingbook.org/html/ConcolicFuzzer.html

Instructions on how to setup KLEE for RERS from Github https://github.com/tudelft-cs4110-2019/symbolic_rers

The blog by Jonathan Salwan on concolic execution: http://shell-storm.org/blog/Binary-analysis-Concolic-execution-with-Pin-and-z3/

PRODUCTS

A small report of max two A4 pages (excluding visualizations) answering the questions from the 2 tasks. The report can be extended with at most two A4 pages if visualizations are used.

An archive (tar/zip) containing the code for computing the results.

Make sure that you have also provided some instructions on how to run your code.

ASSESSMENT CRITERIA

You can either pass or fail this assignment. We expect everyone to pass. You have to complete all lab assignments. Submissions of which the report text does not fit into two A4s (excluding visualizations) will not be evaluated; the submission will automatically receive a fail.

Code that does not compile/run will not be evaluated; the submission will automatically receive a fail.

Your work will be evaluated based on its completeness (having done all tasks), the correctness of the implementation, and demonstration that you understand the results in the analysis.

When deemed insufficient, you will receive feedback and will be given a one-week grace period to fix any shortcomings.

SUPERVISION AND HELP

There will be lab sessions every Wednesday, where the teachers and TAs will be available to answer any questions you may have. The preferred way to ask questions is through Mattermost.

SUBMISSION AND FEEDBACK

The submission is through Brightspace. You will receive feedback within one week after the deadline.