Assignment Series 2: Fuzzing

Drew Springall & Daniel Tauritz

April 4, 2024

1 Introduction

Fuzzing is a general approach which seeks to identify implementation errors in code that parses untrusted input through automated evaluation of inputs called $fuzzing$. Though many approaches exist, three are: blackbox random fuzzing, whitebox constraint-based fuzzing, and grammar-based fuzzing ["Learn&Fuzz: Machine Learning for Input Fuzzing" by Microsoft Research & The Technion]. AI holds the promise of improving fuzz-based approaches by intelligently generating test case inputs based on previously generated inputs and the implementations' behavior when handling those inputs. This assignment series has you work-through the process of applying AI techniques to fuzzing approaches with your group but with a twist. Instead of searching for a single program crash, your goal will be to find the best performing set of fuzzing inputs (i.e., input strings) based on a set of pre-existing implementations. Your collaborative implementation should optimize for:

1. Maximize the number of available implementations that crash/error due to one or more of your chosen inputs

2. Maximize the number of different crash/error types raised across all implementations and all inputs

3. Minimize the number of inputs required to trigger the above crashes/errors as well as the number of characters in each of those inputs

To be clear, finding bugs/misbehaviors in the various implementations **is not the primary goal**. You are explicitly optimizing for the three elements above in order to find the "*best*" set of inputs for fuzzing that data format. Imagine there is a singular, never-before-seen deserialization implementation which you wish to identify bugs in. If you find a bug, you are given a *large* cash prize but every input you test requires that you pay $1,500 in order for it to be evaluated. Instead of spending many millions of dollars to fuzz-test this implementation via the traditional approach, you want to find the inputs with the highest likelihood of identifying a bug based on functionally identical implementations that require you to pay $0 in order to evaluate an input.

2 Submission Instructions

For each of the assignments described in the following sections, each group's coordinator should email Drs. Springall and Tauritz, with CC to the other members of the group, a zip file containing their Python code and report, by the deadlines as described next.

The deadline for Assignment 2c is 10:00 PM on Friday April 26, 2024. The deadline for Assignment 2b must be at least four days before Assignment 2c. The deadline for Assignment 2a must be at least four days before Assignment 2b. Each group's coordinator must email Drs. Springall and Tauritz as soon as possible, but no later than 1:00 PM on Monday April 8, with their group's deadlines for Assignment 2a and Assignment 2b.

3 Assignment 2a: Random Search

Tasks:

- Design and implement the software pipeline in order to efficiently evaluate an input and observe each deserialization implementation's behavior.
- Implement a random search which generates and evaluates random fuzzing sets consisting of fifty inputs for a user-specified number of times, and logs sufficient data to feed the fitness function and plot to show the performance
- Write a *short* report which describes your design, implementation, and results; includes an iteration versus average fitness stair step graph; and lists the best performing set found according to your fitness function.

4 Assignment 2b: Hill Climbing Search

Tasks:

- Improve your fitness function to better capture the three stated objectives and allow the user to indicate their relative weights.
- Design and implement a mutation operator to impose a neighborhood structure on the search space and perform local search.
- Implement a hill climbing search appropriate for your neighborhood structure to search the space of fuzzing sets with user-defined upper bounds on the number/size of inputs. For your report, use a user-defined value of up to fifty (50) inputs in a set and no more than 150 characters per input string. This is in order to improve coverage of your matrix in response to results of Assignment 2a.
- Run experiments to investigate your design and compare it to random search with the new fuzzing set constraints (i.e., comparing to Assignment 2a).

- Write a report which describes your design, implementation, experimentation, results, statistical analysis, and discusses your findings. Include appropriate graphs and tables. Also include the top fuzzing input sets you found and discuss them.

5 Assignment 2c: Evolutionary Computation

Tasks:

- Improve as appropriate your fitness function to better capture the two stated objectives and allow the user to indicate the relative weights indicating the trade-offs between those objectives.

- Reuse from Assignment 2b with improvements as appropriate, your mutation operator to induce a useful neighborhood structure on the search space for performing evolutionary computation.

- Design a recombination operator which takes as input two fuzzing input sets and produces as output one fuzzing input set which inherits genes (and hence traits) of the two input sets (i.e., combines two sets into one set).

- Implement an appropriate evolutionary algorithm (EA) to search the space of fuzzing sets with user-defined upper bounds on the number of inputs in each set and the total size in characters. For your report, use a user-defined value of up to fifty (50) inputs in each set and no more than 150 characters per input string. This is in order to facilitate comparison with your previous Hill Climbing Search. You can either implement your EA from scratch, or employ an existing implementation, such as for instance provided by the following two libraries:

  – DEAP [https://github.com/DEAP/deap]

  – jMetalPy [https://github.com/jMetal/jMetalPy]

- Run experiments to investigate your design and compare it to both Random Search and Hill Climbing Search (i.e., comparing to the results of both those searches as performed for Assignment 2b).

- Remember the $1,500 per fuzzing input in the real-world: $75,000 significantly cuts into your large-cash prize (potentially exceeding it). Figure out how to configure your EA to bias optimization towards small input sizes (e.g., five).

- Write a report which describes your design, implementation, experimentation, results, statistical analysis, and discusses your findings. Include appropriate graphs and tables. Also include the top fuzzing input sets you found, as well as the small biased input size, and discuss them.