In-Class Group Fuzzing Assignment

Drew Springall & Daniel Tauritz

February 20, 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 in-class assignment 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.