Homework #1
COSC 6351: Software Engineering

Group Members:

Soodeh Atefi
`PeopleSoft ID: 1565274`
`Email: satefi@central.uh.edu`

Md Rafiqul Islam Rabin
`PeopleSoft ID: 1797648`
`Email: mrabin@central.uh.edu`

Date: February 08 , 2020.

Hypothesis:

- We will create a mutation fuzzer with 7 different mutation techniques: 3 from random mutation-based fuzzing technique and 4 from AFL mutation-based fuzzer.
- We will create more input candidates by applying those 7 mutation-based fuzzing technique on the seed inputs of bc tool.
- We will run those newly generated input candidates against bc tool to detect number of valid inputs, bugs and coverage information.
- We will compare the performance of random mutation-based fuzzing operators with AFL mutation-based fuzzing operators.
- We will apply multiple mutations to compare against the performance of single mutation.
- We will compare the performance of mutation on arithmetic operators against the mutation on numbers.

Steps for evaluation:

- We will implement the mutation fuzzer in Python with 7 mutation operators. The three random mutation operators are (M1) Deleting random character, (M2) Inserting random character, and (M3) Flipping random character. And four AFL mutation operators are AFL techniques: (M4) Simple arithmetics, (M5) Walking bit flips, (M6) Known integers, and (M7) Walking byte flips.
- Generating random inputs is not good starting points for target based fuzzing like fuzzing bc tool as doing these will create maximum invalid inputs. Therefore, we will select a set of valid inputs from bc tool to start with. After that we will run our fuzzer against the seed inputs of bc tool to have more newly generated mutated test inputs.
- An input is valid if it passes by bc tool and the input is buggy if bc tool fails (i.e. crash immediately, or hang for long time, or give different output for different run) on it. Using gcov tool we can also have the information of coverage (i.e. statement, function, branch). We will generate 1000 new inputs for each different mutation and will compute these metrics.
- We will analyze the metrics (i.e. valid inputs, buggy inputs, coverage info) of random mutation and AFL mutation of previous step 3, separately. Say we can also compute the average cosine distance of generated test inputs.
- Instead of applying a single mutation at a time, we can perform multiple mutation (i.e. apply all 7 mutations) for excessive mutated inputs. We

will generate 1000 new inputs with multiple mutation and will compute the metrics.

- As bc tool is a calculator program, we can mutate arithmetic operators and numbers separately for another comparison. We will generate 1000 new inputs with mutating arithmetic operators and numbers separately and will compute the metrics.