CS639A: Program Analysis, Verification And Testing
Users Online : 14

Assignment 4 : Coverage Guided Fuzzing

Submitted on 2/10/2021 20:49

Instructions

- You are given an extra 10 minutes after due time to submit your assignment.
- However, please note that any submissions made after the due time are marked as late submissions.

Assignment 4 : Coverage Guided Fuzzing

Question:

Problem Statement

It was fun to learn about fuzzing. Clever mutation operators along with nifty coverage metrics makes a naive fuzzer great. In this assignment, you are to implement a ***custom mutation operator*** along with a ***coverage metric operator*** (tl operator will determines if there is a change/improvement in coverage metric when a turtle program is executed with mutated inputs.) for the fuzzer loop in ***submission.py*** file.

A basic coverage guided fuzzer loop has been implemented for you in **`fuzzer.py`** file.

Implement the marked functions and class interfaces in **`Submission/submission.py`** file for this assignment.

1. `def compareCoverage(curr_metric, total_metric)`
2. `def mutate(input_data)` # input_data type InputObject()
3. `def updateTotalCoverage(curr_metric, total_metric)`

Points to note.

1. Fuzzing needs initial seed values. **Specify using '-d' or '--params' flag.**

2. Refer to **fuzzer.py** (KachuaCore/fuzzer.py) and **fuzzerInterface.py** (KachuaCore/interfaces/fuzzerInterface.py) file for better understanding.

3. **The last entry in the Coverage List is to be ignored if the program terminated successfully on the input run.** (https://github.com/CS639A-PAVT/BugTracker/issues/6)

Running the Fuzzer Loop : (From KachuaCore folder)

```
$ ./kachua.py -t 100 --fuzz example/fuzz2.tl -d '{":x": 5, ":y": 100}'

t : Time budget for fuzzing in seconds. -d : Specify initial parameters.
```

Objective: Mutate the inputs in a way that it maximizes coverage within a small time budget. (Eg : Covers as many IF statements in the Turtle program as possible).

Version

CS639A: Program Analysis, Verification And Testing
Users Online : 14

A brief report (less than 5-pages) describing your implementation, assumptions and limitations. (Understand the difference between the tool's limitation and a bug: any error or missing feature that is caught during evaluation is a bu unless it is listed under the "limitations" section of your tool.)
A set of test cases (at least 5) with the expected output. **(tests folder, KachuaCore)**

The quality of all the above would affect your marks. The quality of all the above would affect your marks.

Submission Format

Your submission **MUST** be in the following format:

The submission should be a **zip** file.
The zip file should be named as **assignment_"number"_"Roll-of-student"**.
Zip the content of the **source** as is and submit. **Don't refactor the base code or move the files around.** (KachuaCore Folder, Submission Folder, README)

Please note that your submission will **NOT** be graded if you do not follow the format. Furthermore, we will use the **Readme** file provided by you to build and run your code. Therefore, please make sure that the Readme is clear. We cannot grade your submission if we cannot run it on our system.

Some important comments

Before doing anything "extra" (which might fetch bonus marks), first, complete the basic expectations from your implementation.
Program analysis tools are expected to display their results in a user-friendly manner; a user would never like to use a tool that simply spits out a bunch of numbers. So, display the results from your tool suitably.
Discussion is healthy, copying is not. You are encouraged to discuss the assignments, but you must implement the assignments individually. If any two students are found with "similar" pieces of code, both of them will be failed (with n concern as to who was the source).

Uploaded Files:

assignment_4_21111037.zip

Grades:

Marks: 100