Homework 4: Fuzz Testing

The final homework requires you to write a small text processing application and test it. It includes an adversarial component, in the sense that there is a game in the homework. You will be working “against” another team. There are two steps in the “game”, where the second step includes an auxiliary, “warmup” testing exercise, and an “adversarial” bit. You will work in teams of two.

For Part 1, you will need to turn in your C program by Monday, Dec 2nd, at noon. For Part 2, you will need to test the submission from another team (which we will assign by Monday evening) and complete your testing by Thursday Dec 5th, midnight. We will hold interactive grading sessions on Friday Dec 6th. Sign ups for these slots will be available on Canvas later this week after teams have been selected.

Part 1 : (the “Coding Team”) Write a small text processing application, in C. No other language is allowed. This application should consist of one .c file, namely maxTweeter.c that calculates the top 10 tweeters (by volume of tweets) in a given CSV file of tweets. Specifically, your program should take one command line argument, the path of the csv file. It should gracefully handle both valid and invalid inputs. It should never crash, regardless of input. For valid inputs it should produce 10 lines of output:

<tweeter>: <count of tweets>

For example, your first three lines might look like:
Prem: 96
Ji: 45
Casey: 32
...

Each of these lines are in decreasing order from the most frequent to the 10th most frequent tweeter. In the case of an invalid input the program should output “Invalid Input Format”. A valid input csv file has columns separated with “,”’s, but you cannot assume the location of the tweeter column will be fixed to a particular location (like 8 for instance). A valid csv file will have a header, and the tweeter column will be called “name”, and the items may or may not be surrounded by quotes (“/”). You may also assume that the maximum line length of any valid csv file will not be longer than 1024 characters, the length of the file will not exceed 20,000 lines. Finally, for this assignment, a valid file will not have additional commas inside the tweet. However, your program should not crash on any input, even invalid ones. **Make sure your program works within the provided Ubuntu AFL docker, because that’s where it will be fuzz tested by your assigned testing team, and also sanity tested by us.**

You will do this in teams of two, and your team should thoroughly test and review it however you see fit (including with AFL). You will receive 10 points for correctly passing all test cases on this (we will use a few tests on valid input files to sanity check your submission). This grading is non-adversarial. The code should be clean, well-written, well-commented, well-structured with good use of functions, and well-formatted. If your testing team complains that it is not, we will review it ourselves, and remove up to 5 points for poor coding practices.

For this part of the assignment, you will submit your C program to a public repository on GitHub. In addition to the C program, you should have a README.md file in the top level directory with both team members names at the top of the file. When you submit to canvas, you should submit the link to the GitHub repository with the specific SHA for the latest version of your program, which will be assigned to the Homework 4 group. The link will look something like this: https://github.com/<account_name>/<repo_name>/commit/<commit_sha> (A real world example with the rails project is: https://github.com/rails/rails/commit/9cc88043e70f927a3c8b151c862f6b3cb8b8a6f7)

Part 2:

This part will use the American Fuzzy Lop fuzz tester, and you will demonstrate your results in an interactive grading session with the Professor and the TAs. We have provided a docker to run AFL on DockerHub, along with a guide for getting this docker and running it (also included in the Homework 4 folder of the files). If you run into issues installing docker, uses docker, or running AFL in the docker, please let us know as soon as possible.

1. **Warmup:** You (the "testing team") will be given a program we wrote (to be provided after Part 1 is turned in), which will contain a certain number of defects which will result in crashes; you will receive (10 points, prorated, based on the number of distinct crash-causing defects you find with American Fuzzy Lop/AFL) . This part is non-adversarial.
2. **Adversarial:** You (the "testing team") will be given another team's (the "coding team") application source code, and you will be at a minimum using AFL to fuzz test it, but you may also test or review it in other ways. There are 10 points available here. You will gain 10 points for demonstrating that you have adequately tested the other team's code (at least using AFL and documenting any errors you find). However, for each distinct crash-causing defect found by the team testing your code will lose 2 points from this 10. A crash-causing defect is an input that crashes, and clear indication of where the defect is in the code i.e., a pair of a crashing input, and a line where the defect is. We will only count a defect if you can provide an input that causes the crash inside the docker. These defects should be distinct, which will be determined by the grading TA/Professor's discretion. This part is adversarial. You win by a) making your code as bullet proof as possible and b) testing and reviewing your opponents code as carefully.

For Part 2, you will submit one tar file with two directories, one each for Warmup and Adversarial, each directory should contain a README file, and set of crashing input files. The README file should contain this:

<Defect line number> <brief description of crash> <input filename causing crash>

More detailed instructions for Part 2 will be released after Part 1 is submitted.