Assignment-1 (Complete a Fuzzing loop)

Name – Kartik Jain

Roll no – 231110023

Implementation:

Mainly I have implemented three functions which are given below with explanation:

Compare Coverage Function

The purpose of the Compare Coverage function is to compare the coverage achieved by the current execution with the total coverage achieved so far. It operates as follows:

Input Parameters: curr_metric (current coverage metric), total_metric (total coverage metric).

Functionality: The function iterates through the elements of curr_metric and checks if each element is present in total_metric. If any element from curr_metric is not found in total_metric, the function returns True, indicating that the coverage has improved. Otherwise, it returns False.

Significance: This function is crucial for determining whether a mutation has led to improved code coverage. It helps the fuzzing framework focus on inputs that explore unvisited code paths, increasing the chances of finding vulnerabilities.

Update Total Coverage Function

The Update Total Coverage function is responsible for updating the total coverage metric with the coverage achieved in the current execution. Its functionality is as follows:

Input Parameters: curr_metric (current coverage metric), total_metric (total coverage metric).

Functionality: The function combines the elements of curr_metric and total_metric while eliminating duplicates, resulting in an updated total_metric. This ensures that the total coverage metric always represents the cumulative coverage achieved throughout the fuzzing process.

Significance: The updateTotalCoverage function is essential for maintaining an accurate record of coverage. It helps track which parts of the code have been explored and provides insights into the effectiveness of the fuzzing process.

Mutate Function

The mutate function first generates a random number between 1 and 50 (inclusive) and stores it in the random_number variable.

It then checks if random_number is less than 30. This check introduces a 30% probability that the mutate_add method will be called, and a 70% probability that the mutate_subtract method will be called.

If random_number is less than 30 (i.e., with a 30% probability), the mutate_add method is invoked with the input_data as an argument. This method is supposed to add random values to the variables in input_data.

If random_number is greater than or equal to 30 (i.e., with a 70% probability), the mutate_subtract method is invoked with the input_data as an argument. This method is supposed to subtract random values from the variables in input_data.

The mutate_add and mutate_subtract methods are expected to make changes to the input_data dictionary.

Return Value:

After either the mutate_add or mutate_subtract method is called and modifies input_data, the function returns the modified input_data object.

Assumptions

1. Integration with a Fuzzing Framework**:** The code assumes integration within a larger fuzzing framework, where it relies on receiving input data, coverage information, and a list of intermediate representation (IR) statements. The code's effectiveness is contingent upon the quality and accuracy of these provided inputs.
2. Random Mutation Strategy**:** The code employs a randomization approach to select mutation strategies. It assumes that randomly choosing between two mutation strategies (addition and subtraction) is a suitable method for diversifying input data. The effectiveness of this randomization may vary depending on the specific application.

Limitations

1. Deterministic Mutations**:** The employed mutation strategies are deterministic, involving fixed additions or subtractions from variables. This limitation can lead to repetitive mutations and potentially overlook complex program states that require more varied mutation approaches.
2. Absence of Domain Knowledge**:** The code operates without specific domain knowledge about the program being fuzzed. This absence of domain expertise may hinder its ability to effectively target inputs likely to trigger particular code paths or vulnerabilities.