Assignment #1: Complete a fuzzing loop

Name: Yashvir Singh Nathawat

Roll No: 231110059

IMPLEMENTATION:

I have implemented following three functions with objective to get maximum coverage within a small-time budget:

1) compareCoverage(curr_metric, total_metric)

a) Function Description: This function compares the current coverage metric (curr_metric) with the total coverage metric (total_metric) and returns a boolean indicating whether the current metric is higher than the total metric.

b) Implementation: The function iterates over each **metric_value** in **curr_metric**. For each **metric_value**, it checks if it's not present in **total_metric**. If it finds a **metric_value** in **curr_metric** that is not in **total_metric**, it means there's an improvement in coverage, and the function returns **True**. Otherwise, if all **metric_value** from **curr_metric** are also in **total_metric**, it means there's no improvement in coverage, and the function returns **False**.

2) mutate(input_data)

a) Function Description: This function takes input_data as an argument and performs a mutation operation on it. It then returns the mutated data.

b) Implementation:
I have used three mutation functions which are :

i) mutation_function_one(input_data, min_range, max_range)
The **mutation_function_one** is designed to introduce random mutations to the data contained within the **input_data** object. This function takes three parameters: **input_data**, **min_range**, and **max_range**. Firstly, it generates a random integer, **ran_num**, within the specified range provided by **min_range** and **max_range**.

Additionally, it randomly selects a sign, either -1 or 1, which is stored in the variable **ran_sign**. The function then iterates through the keys of **input_data.data**. For each key, it randomly decides whether to perform a bitwise OR (|) operation or a bitwise XOR (^) operation on the corresponding value, using **ran_num** as the operand. This choice is determined by a random probability threshold set at 0.5. Following the bitwise operation, the result is then scaled by **ran_sign** and subsequently stored back into **input_data.data**. Overall, this function introduces a level of randomness and variability into the data by applying these bitwise operations with random parameters, potentially leading to significant modifications in the dataset.

ii) mutation_function_two(input_data, min_range, max_range)

The **mutation_function_two** is a Python function designed to introduce random mutations to the data stored within the **input_data** object. It takes three parameters: **input_data**, **min_range**, and **max_range**. Firstly, it generates a random integer, **ran_num**, within a fixed range of -100 to +100. Additionally, it randomly selects a sign, either -1 or 1, which is stored in the variable **ran_sign**. The function then iterates through the keys of **input_data.data**. For each key, it employs a random probability threshold set at 0.5 to determine whether to perform a bitwise AND (**&**) operation or a bitwise OR (|) operation on the corresponding value. Based on this decision, it applies the chosen bitwise operation between the value associated with the key and **ran_num**. Subsequently, the result is scaled by **ran_sign** and stored back into **input_data.data**. In essence, this function introduces variability into the data by applying these bitwise operations with a randomly generated operand and sign, potentially leading to significant alterations in the dataset. The randomness in operation choice and operand selection ensures diverse mutations each time the function is executed.

iii) mutation_function_three(input_data, min_range, max_range)

The **mutation_function_three** is a Python function designed to introduce random mutations to the data stored within the **input_data** object. It takes three parameters: **input_data**, **min_range**, and **max_range**. Firstly, it generates a random integer, **ran_num**, within the range of -20 to +20. Additionally, it randomly selects a sign, either -1 or 1, which is stored in the variable **ran_sign**. The function then iterates through the keys of **input_data.data**. For each key, it uses a

random probability threshold set at 0.5 to determine whether to perform a bitwise AND (**&**) operation or a bitwise XOR (^) operation on the corresponding value. Depending on this decision, it applies the chosen bitwise operation between the value associated with the key and **ran_num**. Subsequently, the result is scaled by **ran_sign** and stored back into **input_data.data**. In essence, this function introduces variability into the data by applying these bitwise operations with a randomly generated operand and sign, potentially leading to significant alterations in the dataset. The randomness in operation choice and operand selection ensures diverse mutations each time the function is executed. The range of **ran_num** is limited to -20 to +20, which restricts the magnitude of mutation, providing a controlled level of variation to the data.

iv) mutation_function_three(input_data, min_range, max_range)
The **mutation_function_four** is a Python function designed to introduce controlled variations to the data stored within the **input_data** object. It takes three parameters: **input_data**, **min_range**, and **max_range**. For each key in **input_data.data**, the function generates a random value between 0 and 1. Based on the generated value, it applies specific mutations. If the random value is less than 0.5, the value is set to 0. If it's between 0.5 and 0.6, the value is set to -1. For values between 0.6 and 0.7, the value is set to 1. For values between 0.7 and 0.8, the value is set to **min_range**. Finally, for values greater than or equal to 0.8, the value is set to **max_range**. This controlled approach allows for targeted adjustments to the data, providing specific variations within predefined ranges. The function then returns the modified **input_data**.

3) updateTotalCoverage(curr_metric, total_metric)

a) Function Description: This function updates the total coverage metric (total_metric) with the current coverage metric (curr_metric).

b) Implementation: I have taken the union operation to update Total Coverage.

ASSUMPTIONS:

1. **Range Limit Assumption**: The code assumes that the acceptable range for random values generated, particularly in functions like **mutation_ function_two**, **mutation_function_three**, and **mutation_function_four**, is between -500 and +500. This range limit ensures that the mutations applied to the data so that the time of per iteration is less.
2. **Bitwise Operation Assumption**: In functions like **mutation_function _two** and **mutation_function_three**, it is assumed that bitwise operations (AND, OR, XOR) are valid and applicable to the data in **input_data.data**.

LIMITATIONS:

The provided code has several limitations:

1. Limited Mutation Techniques: The code primarily relies on basic bitwise operations (AND, OR, XOR) and random value assignments. It doesn't incorporate more advanced or specialized mutation techniques that might be more effective in certain scenarios.
2. No Validation Checks: The code doesn't include validation checks for the input parameters. For instance, it assumes that **input_data** is always in the expected format with a **data** attribute, and that **min_range** and **max_range** are within valid bounds.