Fuzz Them All

Student Name: Abdulaziz Alqadhi

Email: amalqadh@asu.edu

Submission Date: 29/11/2023

Class Name and Term: CSE543 Fall 2023

Fuzzer Overview and Usage Guide

1. Introduction:

The provided Java program is a simple a dumb mutative fuzzer designed to generate mutated inputs for testing purposes. Fuzz testing involves providing unexpected or random data as input to a program to discover vulnerabilities, bugs, or unexpected behaviors. This fuzzer utilizes a pseudo-random number generator (PRNG) to create variations in the input data.

2. Dependencies:

The fuzzer has no external dependencies. It relies on the standard Java libraries for file handling, string manipulation, and random number generation.

3. Building and Running:

To build and run the fuzzer, follow these steps:

- Compilation:
  - Ensure you have Java Development Kit (JDK) installed.
  - Save the provided code in a file named **“Fuzzer.java”**.
  - Open a terminal and navigate to the directory containing the **“Fuzzer.java”** file.
  - Compile the code using the command: **“javac Fuzzer.java”**

- Execution:
  - After successful compilation, run the fuzzer using the command:
    " java Fuzzer (-s seed | <prng_seed>) <num_iterations> [-o <output_file>]"
    Replace **“<prng_seed>“** with a seed string or use the **“-s”** flag to pass a seed file name that is on the same directory as compiled Fuzzer.
  - **“<num_iterations>“** specifies the number of iterations for the fuzzer.
  - Optional, use **“-o”** followed by the desired **“<output_file>“** to save the output to a file.

4. Input Generation Strategy:

- The fuzzer starts with an initial seed provided as a string or read from a file.
- It converts the seed to a byte array and initializes a pseudo-random number generator (PRNG) using the seed's hash code.
- The program iteratively mutates the input by randomly changing individual bytes based on a predefined probability (“RANDOM_BYTE_PROBABILITY”).
- Every “EXTEND_INPUT_INTERVAL” iterations, the input is extended by appending random bytes (“EXTEND_INPUT_LENGTH”) to increase diversity.
- The final mutated input is either written to an output file (if specified) or printed to the standard output.

5. Command-Line Arguments:

- **“-s seed”**: Specifies a seed file name (the seed file most be on the same directory as the Fuzzer).
- **“<prng_seed>“**: Alternative way to provide the PRNG instead of using the **“-s”** flag.
- **“<num_iterations>“**: Specifies the number of iterations for input mutation.
- **“-o <output_file>“**: Optional flag to specify an output file for the final mutated input.

6. Automation Script:

To ease the fuzzing process, an automation script named "automate_fuzzer.sh" is provided. It facilitates iterative execution of the fuzzer, directing output to corresponding crash files, and testing the fuzzer output on the programs.

7. Error Handling:

The program includes basic error handling for incorrect command-line arguments or issues with reading the seed file.

8. Example Usage:

- Using seed string: “java Fuzzer seedString 1000 -o output.crash”
- Using seed file: “java Fuzzer -s seedfile.txt 500”
- Using seed string: “java Fuzzer 'AAAADDD' 500 | /challenge/prog_0”
- Using seed file: “java Fuzzer -s seedfile.txt 500 -o out.crash; /challenge/prog_0 < out.crash”

9. Conclusion:

The fuzzer provides a simple yet effective way to generate diverse inputs for testing applications. Users can customize the fuzzer behavior by adjusting parameters such as the seed, number of iterations, and output file.