Fuzzer Improvement and Experimentation Report

Introduction

In this project, we aimed to enhance the bug-finding ability of a grey-box mutational fuzzer implemented in Python. The baseline fuzzer, derived from The Fuzzing Book, served as our starting point. The goal was to introduce novel concepts or implement prior research contributions to improve the fuzzer's effectiveness. We created a custom fuzzer, MyFuzzer, incorporating innovative coverage tracking and execution strategies.

Implementation

Fuzzer Architecture

Our custom fuzzer, MyFuzzer, extends the baseline grey-box mutational fuzzer. It includes three main components:

Coverage Tracking (MyCoverage): We implemented a custom coverage tracking mechanism to capture executed lines during the fuzzing process. The coverage method returns a representation of the coverage achieved during the execution, focusing on tracked lines.

Runner (MyRunner): The MyRunner class encapsulates the execution of the target program. It features a run_function method to execute the target function with a given input and a calculate_coverage method to track coverage.

Fuzzer Logic (MyFuzzer): The MyFuzzer class overrides the baseline fuzzer's methods, introducing our strategies for generating inputs and conducting the fuzzing process.

Fuzzer Execution

The fuzzing process involves repeatedly generating mutated inputs, executing them using the MyRunner, and updating coverage information. The fuzzer aims to discover bugs by identifying anomalous behaviors in the target program. The is_bug method checks if the result of the execution indicates a bug, and the handle_bug method defines how to handle bugs when found.

Example Program

To demonstrate the effectiveness of our fuzzer, we created a simple example Python program (target_program.py). The target program includes a function, target_program, which raises an exception if the input data contains the substring "bug." This scenario simulates a bug detection condition in a real-world application.

Experiments and Results

Experimental Setup

We designed experiments to compare the performance of our custom fuzzer (MyFuzzer) against the baseline fuzzer from The Fuzzing Book. Metrics included code coverage, bug discovery rate, and execution time. The target program (target_program.py) served as the test subject.

Results

Upon running the experiments, we observed the following:

Code Coverage: MyFuzzer consistently achieved higher code coverage compared to the baseline fuzzer. The custom coverage tracking logic contributed to a more thorough exploration of code paths.

Bug Discovery: Our fuzzer effectively identified bugs in the target program, demonstrating improved bug-finding capability compared to the baseline. Bugs were logged and inputs triggering bugs were saved for further analysis.

Execution Time: While MyFuzzer showed increased efficiency in code exploration, the execution time varied based on the complexity of the target program. Some trade-offs were identified, with MyFuzzer potentially requiring more time in certain scenarios.

Conclusion

The custom fuzzer, MyFuzzer, demonstrated improved bug-finding capabilities and enhanced code coverage compared to the baseline fuzzer. The experiments highlighted the effectiveness of our coverage tracking and execution strategies. The results suggest that our approach can be valuable for identifying bugs in a real-world context, but some trade-offs in execution time may be present in certain scenarios.

Overall, the enhancements introduced in MyFuzzer showcase the potential for innovative strategies to advance grey-box mutational fuzzing techniques.