Range Checker : To improve the fuzz efficiency

Wooseok Kang
kangwoosuk1@gmail.com
KAIST
Daejeon, South Korea

Abstract

In fuzzing, input generation is one of the most important features which can dominate performance of fuzzer. In this project, I will implement the range analysis tools for general C program based on data flow analysis to improve the code coverage and find specific domain bugs. The result will be compared other well known fuzzers like AFL.

Keywords: static analysis, fuzzing, data flow analysis

ACM Reference Format:
Wooseok Kang. 2020. Range Checker : To improve the fuzz efficiency. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Fuzzing is one of the most commonly used method to find software bugs. It causes the program crash with generated random inputs. While fuzz testing for large program, the execution time will take long, so the careful input generation is important for effective fuzzing. However, since there are too many possibilities in program, it is not easy to predict the path of input before we run it.

In this project, we analysis the range of variables in the program for each code coverage based on data flow analysis as much as precisely. Also we inspect the range of input values which can reach this coverage. Then we can use these summaries to generate more desirable inputs for fuzzing. For example, if we somehow know the bugs position in the source code, we can reject the inputs which don't pass it. Also, we can make inputs to explore more diverse code coverage by modulating the distribution. Our experiment target will be general C program.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Approach

Our program has two stages. The first step is to analyze the range of the each variable of the program and possible input values based on data flow analysis, and the next step is running fuzz testing with this information.

2.1 Value Range Analysis

In value range analysis, we analyze the range of each values in the program as possible as precisely. Also, it collects the information for possible input value ranges which can reach in this code coverage. All this processes are done with data flow analysis using appropriate control-flow graph and equation rule.

In Figure 1, it shows the input range analysis result that can reach for each code coverage. If we assume that the bug is in the location that x has the range [1, 100), we can rejects randomly generated inputs which are out of bound. It will help the finding input which can cause actual bug effectively.

```
int main(void) {
    int x;                          x : (-∞, +∞)
    scanf("%d", &x);

    if (x >= 1) {
        ...                         x : [1, +∞)
        if (x < 100) {
            ...                     x : [1, 100)
        }
    } else {
        ...
                                    x : (-∞, 1)
    }
}
```

Figure 1. Example C source code which contains range information.

2.2 Fuzz Testing

In fuzzing step, we will generate random inputs for program using input generation methods for other well known fuzzers.

However, our fuzzer has two advantage features compare them. It can modulate the inputs to explore the large coverage using information which is collected in previous stage. Also, if user gives the specific location of source code, our fuzzer can generate input to fit that specification..

3 Expected Result

I will be experiment our program for real C program. Actually, there are some limitations for data flow analysis and C program has so many different actions, it may need some assumption or approximation while implementing. However, I will try to cover as much as space I can.

If the result will be as I expected, our fuzzer can find more coverage than not using the analysis information. Also, it can generate the inputs which pass through specific code path effectively. Especially, it will act well for slightly large programs. I will compare the performance of fuzzers which use the analysis information and which not, then finally it will be compared with other well known fuzzers like AFL.