Title Text

Subtitle Text, if any

Jaeho Kim
KAIST
Daejeon, South Korea
oojahooo@kaist.ac.kr

Tae Eun Kim
KAIST
Daejeon, South Korea

Seunghyeon Jeong
KAIST
Daejeon, South Korea

Kihwan Kim
KAIST
Daejeon, South Korea

Abstract

This is the text of the abstract.

1. Introduction

Fuzzing is a technique that is widely applied in the real world with its simple intuition and excellent applicability. For example, Google provides a large platform named OSS-Fuzz for open source software. Developers can upload their open source project to this platform and get reports about bugs that founded by executing the fuzzer. Actually, OSS-Fuzz has helped identify and fix over 8,900 vulnerabilities and 28,000 bugs across 850 projects [1].

Greybox fuzzing is a generally used fuzzing technique that explores the program guided by code coverage. It mutates the seed inputs in a way that increases code coverage. The term 'Greybox' means that the fuzzer does not use whole information of program's code, but it uses only code coverage information.

Because of the aforementioned characteristic of greybox fuzzer, it struggle with the absence of code coverage. Especially, it is challenging for greybox fuzzer if there are conditions with no intermediate code coverage. Figure 1 shows the example of the condition with no intermediate code coverage. At the line 6, the input argument should be exactly "HELLO" in order to execute true branch and crash statement at line 8. So executing with input string "HELL" and "HEAVEN" respectively have same code coverage. It is fatal problem for mutator in greybox fuzzing. "HELL" is more similar (i.e. less edit distance) with "HELLO" than "HEAVEN", so it can make "HELLO" by adding only one character, but the fuzzer evaluates those two inputs are same effectiveness. Therefore the fuzzer makes more insignificant inputs, and it takes too much time.

To overcome this problem, we present a new task for encoding the value coverage in string domain, using finite state machine, and combine it with directed fuzzing technique. Directed fuzzing is a new technique

```
1 int main(int argc, char *argv[])
2 {
3     if (argc < 2)
4         return 1;
5
6     if (!strcmp(argv[1], "HELLO"))
7     {
8         Crash();
9         printf("Hello␣World!");
10        return 0;
11    }
12
13    OK();
14    return 0;
15 }
```

Figure 1: Condition Example of a Simple Program that Takes String Input

[1] https://github.com/google/oss-fuzz#trophies

of fuzzing that aims a specific location of the program. So it is more efficient than greybox fuzzing when we want to find an input that causes program to crash at specific location.

2. Overview

2.1 Finite State Machine

Finite state machine is a useful structured representation that represents a program's behavior. In general, a program is a transition system. Every program has states and transition functions. So we can express a program defining states and transition function. Actually, Depending on how the state and transition function are defined, an infinite number of states may be required to express a program. But in a specific domain, we can represent it using finite number of states. So we utilize the finie state machine for only representing conditional expression containing string library functions.

We define finite state machine like below for using our domain.

DEFINITION 2.1. *$(Q, \Sigma, \delta, q_0, F)$ is a finite state machine where Q is finite set of states, Σ is a set of input symbol, especially ascii characters in our domain, $\delta : (Q \times \Sigma) \rightarrow Q$ is a partial function that takes current state and current input symbol, q_0 is initial state (i.e. state that input symbol is empty string), and F is a set of accepted (final) states.*

2.2 Examples and Methods

We choose three scenarios that usually occurs in many programs that take string inputs, 1) using `strcmp` function in conditional expression, 2) using `strstr` function in conditional expression, 3) using `atoi` function in conditional expression. Figure 2 is the simple code examples that contains our scenarios.

Each scenario makes greybox fuzzer struggle with the absence of code coverage. In Figure 2a, we have to find exactly "123456" as an crashing input. In Figure 2b, we should find the string that contains "1234" as an crashing input. In Figure 2c, we have to find the string that represents the integer 123456.

To make new coverage method for these scenarios, we use the finite state machine. Finite state machine can express the sequence of transition from starting input symbol to final input symbol, and check whether the input string is accepted.

So we construct the finite state machines for each scenario, and add it as new conditional statement these

```c
int main()
{
    char num_str[500];
    scanf("%s", num_str);

    if (!strcmp(num_str, "123456"))
        printf("Crash!!");

    return 0;
}
```

(a) Scenario that uses strcmp function

```c
int main()
{
    char num_str[500];
    scanf("%s", num_str);

    if (strstr(num_str, "1234"))
        printf("Crash!!");

    return 0;
}
```

(b) Scenario that uses strstr function

```c
int main()
{
    char num_str[500];
    scanf("%s", num_str);

    if (atoi(num_str) == 123456)
        printf("Crash!!");

    return 0;
}
```

(c) Scenario that uses atoi function

Figure 2: Three scenarios that usually occurs in many programs

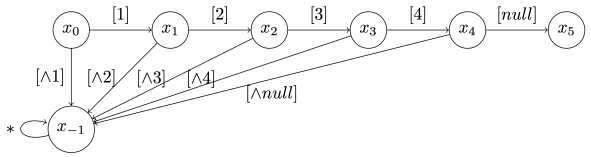
to original code. Figure 3 shows the FSMs constructed for each scenario. And we implement these FSMs in real program as adding new code.

You can see the detailed implementation in our repo [2].

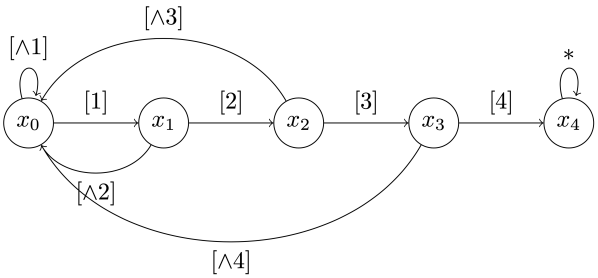
3. Evaluation

We have an experiment for comparing the efficiency of our technique with traditional fuzzers. So our baselines

[2] https://github.com/goodtaeeun/smAFL



(a) Contructed FSM for strcmp



(b) Contructed FSM for strstr

Figure 3: Constructed finite state machines for each scenario

are traditional greybox fuzzer, AFL [**?**], and traditional directed fuzzer, AFLGO [1].

3.1 Experimental Setup

3.2 Result

A. Appendix Title

This is the text of the appendix, if you need one.

References

[1] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. 2017. Directed Greybox Fuzzing. In *Proceedings of the ACM Conference on Computer and Communications Security*. 2329–2344.

[2]]aflfuzz Michal Zalewski. [n. d.]. American Fuzzy Lop. `http://lcamtuf.coredump.cx/afl/`.