Symbolic Fuzzer

Done by:

Melika Zare, 9632456

Negin Khalifat, 9632461

Fateme Masoudi, 9632440

Fateme Ebrahimi, 9632439

:Teacher

Dr. Mohammad Amin Alipour

Spring 1399-1400

Table of Contents

Introduction

Fuzzing is the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

So far, Fuzzing has gone through two stages of development: concolic and symbolic fuzzing.

There are two problems with concolic approach. The first is that, concolic tracing relies on the existence of sample inputs. Secondly, direct information flows could be unreliable if the program has indirect information flows such as those based on control flow. In such cases, static analysis becomes necessary to bridge the gap. Symbolic execution is one of the ways that we can reason about the behavior of a program without executing it. A program is a computation that can be treated as a system of equations that obtains the output values from the given inputs. Executing the program symbolically -- that is, solving these mathematically -- along with any specified objective such as covering a particular branch or obtaining a particular output will get us inputs that can accomplish this task.

This project aims to develop a symbolic fuzzer to generate all possible paths to a given depth, collect the information about the constraint and possible input for the given example with variables which annotated with the type information.

Fuzzing

- Breaking it down into simpler terms, fuzzing is a testing technique for applications in which we pass random, invalid input to the target application. The application is then monitored for *unexpected behavior*. The unexpected behavior could be the application crashing, memory leakage, etc. that occur for previously unknown niche test cases that go beyond the scope of manual testing.
- One thing to keep in mind is that invalid inputs are supposed to be *valid enough* that they are accepted by the target application for processing and don't make the application crash right away. Their task is to help us find the hidden exceptions that are yet to be found within the application.

The project source code is uploaded in a GitHub repository.

The link to the source code is:

https://github.com/SotwareTesting-Project/Symbolic-Fuzzer

Fuzzing Features

- It is supported for self-contained function call
- It is supported for list
- The methods are not limited to void function
- Reassignment is supported by this project
- The output report can be generated for Unix

Setup Project

in windows:

```
git clone https://github.com/SotwareTesting-Project/Symbolic-Fuzzer.git
python3 -m venv environment_name
.\environment_name\Scripts\activate.bat
cd Symbolic-Fuzzer
pip install -r requirements.txt
```

Usage:

- `python3 src\main.py -i Examples\simpleIfElse.py –d 10`

Shortcomings and Future work

- It is not supported for recursive functions
- It is not supported for loop expressions
- For while expression this project raised an exception due to maximum depth exceed
- It is not supported for iterator and next in python
- It is not supported for functions with list as an argument
- Z3 is not compatible for all combinations of expressions
- Only used Z3 as a constraints solver

Conclusion

We conclude that, symbolic execution could not be suitable for all programs and the output depends on MAX_DEPTH. The illustration of that is that it could not determine and detect all the possible failures under some circumstances when execution was stopped at a lower depth. However, it has some benefits that brought in understanding all characteristic of the program and generating test cases for all satisfied paths.