定向覆盖模糊测试工具的设计与实现

毕业设计中期检查

雷尚远

南京邮电大学计算机学院

2023 年 4 月 17 日

1 Background

Pre-Knowledge

Motivation

Research Status

2 References

What Fuzzing is?

Defination[1]

- **Fuzzing** Fuzzing is the execution of the PUT using input(s) sampled from an input space (the “fuzz input space”) that protrudes the expected input space of the PUT.
  - PUT: Program Under Test
- **Fuzz testing** Fuzz testing is the use of fuzzing to test if a PUT violates a correctness policy.
- **Fuzzer** A fuzzer is a program that performs fuzz testing on a PUT.
- **Bug Oracle** A bug oracle is a program, perhaps as part of a fuzzer, that determines whether a given execution of the PUT violates a specific correctness policy.
- **Fuzz Configuration** A fuzz configuration of a fuzz algorithm comprises the parameter value(s) that control(s) the fuzz algorithm.
- **Seed** A seed is a (commonly well-structured) input to the PUT, used to generate test cases by modifying it.

What Fuzzing is?

Fuzz Testing

```
Input: $\mathbb{C}$, $t_{\text{limit}}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{\text{elapsed}} < t_{\text{limit}} \land$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{\text{bug}}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

Fuzzing Algorithm

```
Input: $\mathbb{C}$, $\mathsf{t}_{\text{limit}}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $\mathsf{t}_{\text{elapsed}} < \mathsf{t}_{\text{limit}} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $\mathsf{t}_{elapsed}$, $\mathsf{t}_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $\mathsf{O}_{\text{bug}}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $\mathsf{O}_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

- $\mathbb{C}$:a set of fuzz configurations
- $\mathsf{t}_{\text{limit}}$: timeout
- $\mathbb{B}$: a set of discovered bugs

Fuzzing Algorithm

```
Input: $\mathbb{C}$, $t_{limit}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{elapsed} < t_{limit} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{bug}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

$\textsc{Preprocess}\ (\mathbb{C}) \rightarrow \mathbb{C}$

- **Instrumentation**
  - grey-box and white-box fuzzers
  - static/dynamic(InputEval)
- **Seed Selection**
  - weed out potentially redundant configurations
- **Seed Trimming**
  - reduce the size of seeds
- **Preparing a Driver Application**
  - library Fuzzing, kernal Fuzzing

Fuzzing Algorithm

```
Input: $\mathbb{C}$, $t_{\text{limit}}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{\text{elapsed}} < t_{\text{limit}} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{\text{bug}}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

Stop Condition

- $t_{\text{elapsed}} < t_{\text{limit}}$
- CONTINUE ($\mathbb{C}$)$\longrightarrow$ $\{\texttt{True}, \texttt{False}\}$
  - Determine whether a new fuzz iteration should occur

Fuzzing Algorithm

```
Input: $\mathbb{C}$, $t_{limit}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{elapsed} < t_{limit} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{bug}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

SCHEDULE ($\mathbb{C}$, $t_{elapsed}$, t_{limit}) $\rightarrow$ conf

- **Function**
  - Pick important information(conf)
- **FCS Problem**
  - *exploration*:Spent time on gathering more accurate information on each configuration to inform future decisions
  - *exploitation*:Spent time on fuzzing the configurations that are currently believed to lead to more favorable outcomes

Fuzzing Algorithm

```
Input: $\mathbb{C}$, $t_{limit}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{elapsed} < t_{limit} \land$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{bug}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

INPUTGEN (conf)⟶ tcs

- **function**
  - Generate testcases
- **classification**
  - Generation-based(*Model-based*)
  - Mutation-based(*Model-less*)
  - White-box Fuzzers: symbolic execution

Fuzzing Algorithm

```
Input: $\mathbb{C}$, $t_{\text{limit}}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{\text{elapsed}} < t_{\text{limit}} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{\text{bug}}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

INPUTEVAL (conf, tcs, O_{bug})
→ $\mathbb{B}'$, execinfos

- **Fuzzing PUT**
  - tcs
  - $\mathbb{B}'$
- **Feedback Information**
  - conf, tcs
  - execinfos (tcs,crashes,stack backtrace hash,edge coverage,etc.)

Fuzzing Algorithm

```
Input: $\mathbb{C}$, $t_{limit}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{elapsed} < t_{limit} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{bug}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

- CONFUPDATE ($\mathbb{C}$, conf, execinfos) $\rightarrow \mathbb{C}$
  - Update Fuzz Configuration(distinguishablity)
  - Seed Pool Update
- $\mathbb{B} \cup \mathbb{B}' \rightarrow \mathbb{B}$
  - Update Bugs Set

Fuzzing Algorithm

```
Input: $\mathbb{C}$, $t_{\text{limit}}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{\text{elapsed}} < t_{\text{limit}} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{\text{bug}}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

stop condition

- $t_{\text{elapsed}} < t_{\text{limit}}$
- CONTINUE ($\mathbb{C}$)$\longrightarrow$ {True, False}
  - Determine whether a new fuzz iteration should occur

1 Background

Pre-Knowledge

Motivation

Research Status

2 References

Classification

The amount of collected information defines the color of a fuzzer[1].

```
Input: $\mathbb{C}$, $t_{limit}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{elapsed} < t_{limit} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{bug}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

- **program instrumentation**
  - static
  - dynamic
- processor traces
- system call usage
- etc.

Classification

```
Input: $\mathbb{C}$, $t_{limit}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{elapsed} < t_{limit} \land$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{bug}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

Program Instrumentation

- Static
  - source code
  - intermediate code
  - binary-level
- Dynamic

Classification

```
Input: $\mathbb{C}$, $t_{limit}$
Output: $\mathbb{B}$ // a finite set of bugs
$\mathbb{B} \leftarrow \varnothing$
$\mathbb{C} \leftarrow$ Preprocess($\mathbb{C}$)
while $t_{elapsed} < t_{limit} \wedge$ Continue($\mathbb{C}$) do
    conf $\leftarrow$ Schedule($\mathbb{C}$, $t_{elapsed}$, $t_{limit}$)
    tcs $\leftarrow$ InputGen($conf$)
    // $O_{bug}$ is embedded in a fuzzer
    $\mathbb{B}'$, execinfos $\leftarrow$ InputEval($conf$, $tcs$, $O_{bug}$)
    $\mathbb{C} \leftarrow$ ConfUpdate($\mathbb{C}$, $conf$, $execinfos$)
    $\mathbb{B} \leftarrow \mathbb{B} \cup \mathbb{B}'$
return $\mathbb{B}$
```

Program Instrumentation

- Static
- Dynamic
  - dynamically-linked libraries
  - execution feedback: branch coverage, new path, etc.
  - race condition bugs: thread scheduling

Classification

Classification of Fuzzing

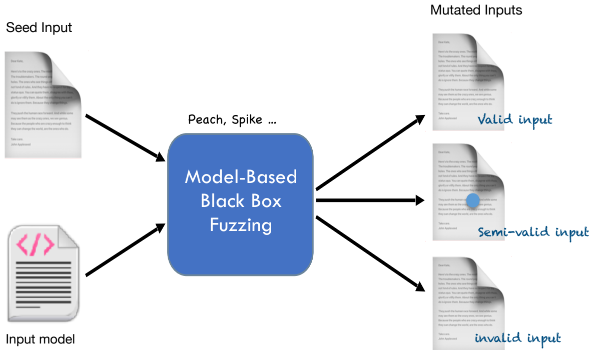
- **Black-box Fuzzing**
  - no program analysis, no feedback
- **White-box Fuzzing**
  - mostly program analysis
- **Grey-box Fuzzing**
  - no program analysis, but feedback

Why Grey-box Fuzzing ?

- Black-box Fuzzing

  Defination: techniques that do not see the internals of the PUT,and can observe only the input/output behavior of the PUT, treating it as a black-box[1].

  -No program analysis, no feedback

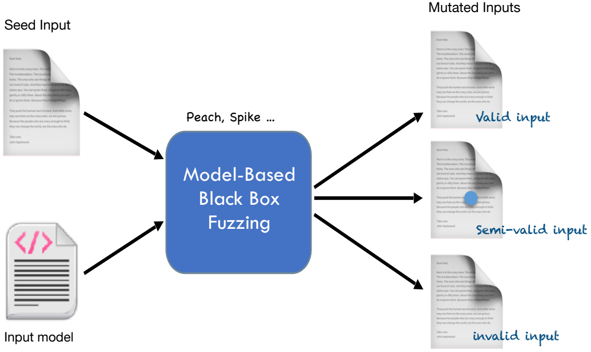


Why Grey-box Fuzzing ?

- Black-box Fuzzing

  Defination: techniques that do not see the internals of the PUT,and can observe only the input/output behavior of the PUT, treating it as a black-box[1].

  - No program analysis, no feedback



Seed Input

Input model

Peach, Spike ...

Model-Based Black Box Fuzzing

Mutated Inputs

Valid input

Semi-valid input

invalid input

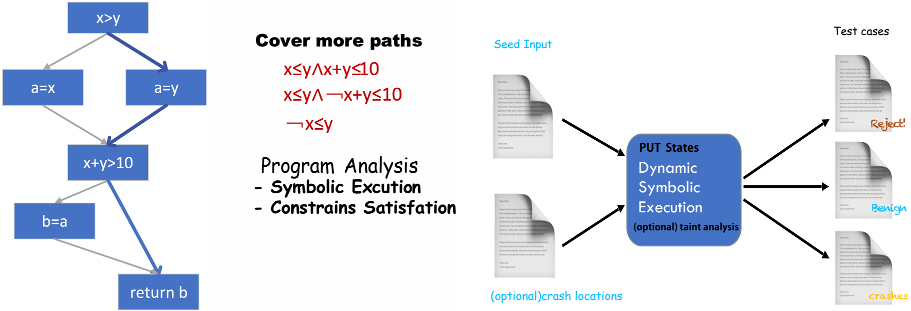
- You have no view of the PUT,but have some view of the input/output domain
- Fuzzing congfigurations are not changed according to some feedback
  - some fuzzer may add the testcases to seed pool
- Not effective

Why Grey-box Fuzzing ?

- White-box Fuzzing

  Defination: techniques that generates test cases by analyzing the internals of the PUT and the information gathered when executing the PUT[1].

  - Requires heavy-weight program analysis and constraint solving.



Cover more paths

$x \leq y \land x + y \leq 10$

$x \leq y \land \neg x + y \leq 10$

$\neg x \leq y$

Program Analysis

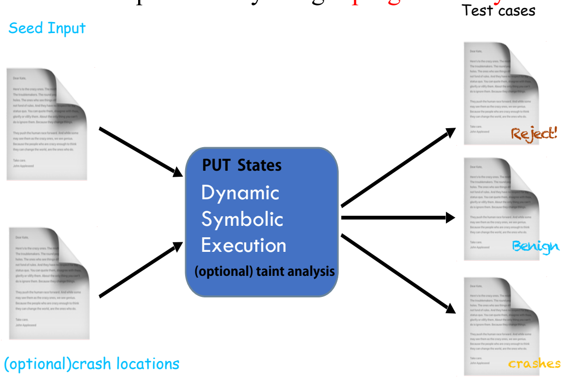
- **Symbolic Excution**
- **Constrains Satisfation**

Why Grey-box Fuzzing ?

- White-box Fuzzing

  Defination: techniques that generates test cases by analyzing the internals of the PUT and the information gathered when executing the PUT[1].

  - Requires heavy-weight program analysis and constraint solving.



- You have the view of the PUT state(CFG,CG)
- Program analysis (effective but not efficient!)

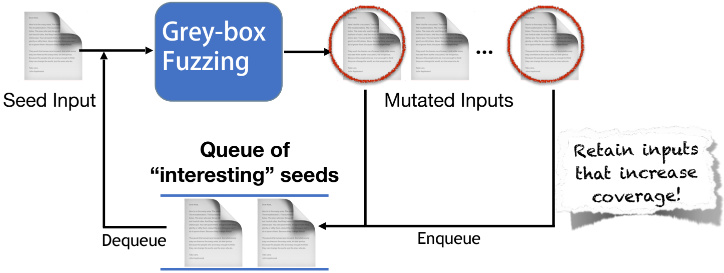
Why Grey-box Fuzzing ?

- Grey-box Fuzzing

  Defination: techniques that can obtain *some* information internal to the PUT and/or its executions to generates test cases[1].

  - Uses only lightweight instrumentation to glean some program structure

  - And coverage feedback


Seed Input
Grey-box Fuzzing
...
Mutated Inputs
Queue of "interesting" seeds
Dequeue
Enqueue
Retain inputs that increase coverage!

Why Grey-box Fuzzing ?

Grey-box Fuzzing is frequently used

- State-of-the-art in automated vulnerability detection
- Extremely efficient coverage-based input generation
  - All program analysis before/at instrumentation time.
  - Start with a seed corpus, choose a seed file, fuzz it.
  - Add to corpus only if new input increases coverage.

Why Directed Grey-box Fuzzing ?

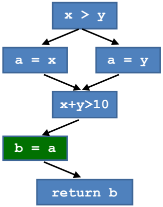
Directed Fuzzing has many applications

- Patch Testing: reach changed statements
- Crash Reproduction: exercise stack trace
- SA Report Verification: reach “dangerous” location
- Information Flow Detection: exercise source-sink pairs

Why Directed Grey-box Fuzzing ?

Directed Fuzzing

- **Goal:reach a specific target**
  - **Target Locations**: the line number in the source code or the virtual memory address at the binary level[2].
  - **Target Bugs**: use-after-free vulnerabilities, etc.
- **DSE:classical constraint satisfaction problem**
  - uses program analysis and constraint solving to generate inputs that systematically and effectively explore the state space of feasible paths[3].
  - **Program analysis** to identify **program paths** that reach given program locations.
  - **Symbolic Execution** to derive **path conditions** for any of the identified paths.
  - **Constraint Solving** to find an input



$$\varphi_1 = (x>y) \land (x+y>10)$$
$$\varphi_2 = \neg(x>y) \land (x+y>10)$$

1 Background

Pre-Knowledge

Motivation

Research Status

2 References

Why Directed Grey-Box Fuzz?

- 大家都会 LaTeX，好多学校都有自己的 Beamer 主题

Why Directed Grey-Box Fuzz?

- 大家都会 LaTeX，好多学校都有自己的 Beamer 主题
- 中文支持请选择 XeLaTeX 编译选项

1 Background

2 References

[1] MANÈS V J, HAN H, HAN C, et al. The art, science, and engineering of fuzzing: A survey[J]. IEEE Transactions on Software Engineering, 2019, 47(11) : 2312 – 2331.

[2] WANG P, ZHOU X, LU K, et al. The Progress, Challenges, and Perspectives of Directed Greybox Fuzzing[EB]. arXiv, 2022.

[3] MA K-K, YIT PHANG K, FOSTER J S, et al. Directed symbolic execution[C] // Static Analysis: 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings 18. 2011 : 95 – 111.

Thanks!