T-Fuzz: Fuzzing by Program Transformation

Hui Peng¹, Yan Shoshitaishvili², Mathias Payer¹
Fuzzing as a bug finding approach

- Fuzzing is finding more and more CVEs
- Vendors use it as proactive defense measure: OSS-Fuzz
- Hackers use it as first step in exploit development
Challenges for fuzzers

➢ Challenges
  ❑ Shallow coverage
  ❑ Hard to find “deep” bugs

➢ Root Cause
  ❑ Fuzzer-generated inputs cannot bypass complex sanity checks in the target program

5/22/2018
Existing approaches & their limitations

- Existing approaches focus on **input generation**
  - Driller (concolic execution)
  - VUzzer (taint analysis, data & control flow analysis)

- Limitations
  - High overhead
  - Not scalable
  - Not able to bypass “hard” checks
    - Checks on checksum, crypto-hash values
Insight: some checks are non-critical

- Some sanity checks are not intended to prevent bugs
  - **Non-Critical Checks (NCC)**
    - E.g., check on magic values, checksums, hashes
  - Removing NCCs won’t incur erroneous bugs
  - Removal of NCCs simplifies fuzzing

```c
void main() {
    int fd = open(...);
    char *hdr = read_header(fd);
    if (strncmp(hdr, "ELF", 3) == 0) {
        // main program logic
        // ...
    } else {
        error();
    }
}
```
T-Fuzz: fuzzing by program transformation

- Fuzzer generates inputs
- When Fuzzer gets stuck, Program Transformer:
  - Detects **NCC candidates**
  - Transforms program
- Crash Analyzer verifies crashes
- Repeat

Transformed Programs

Fuzzer (e.g. AFL)

Program Transformer

Crash Analyzer

Inputs

Crashing inputs

Bug Reports

False Positives

T-Fuzz design
Detecting NCC candidates

- Approximate NCCs as the edges connecting covered/uncovered nodes in the CFG
- Overapproximate, *may contain false positive*
- Lightweight and simple to implement
  - dynamic tracing
Program Transformation

- **Goal**: disable NCCs

- **Our approach**: negate NCCs
  - Easy to implement: static binary rewriting
  - Zero runtime overhead in target program
  - The CFG of the program stays the same
  - Traces of the transformed program map to the original one
  - Path constraints of the original program can be recovered
Filtering out false positives & reproducing bugs

Collect path constraints of the **original** program by symbolically tracing the **transformed** program

Path Constraints

Satisfiable?

No → False Positive

Yes → Generate input to reproduce the crash
Example 1

int main (){
    int x = read_input();
    int y = read_input();
    if (x > 0) {
        if (y == 0xdeadbeef)
            bug();
    }
}

int main (){
    int x = read_input();
    int y = read_input();
    if (x > 0) {
        if (y != 0xdeadbeef)
            bug();
    }
}

Collected path constraints:
{x > 0, y == 0xdeadbeef}

Un-negating

Negated check

Original Program

Transformed Program

SAT True BUG
Example 2

Original Program

```c
int main (){
    int i = read_input();

    if (i > 0) {
        func(i);
    }
}

void func(int i) {
   if (i <= 0) {
        bug();
    }
    //...
}
```

Transformed Program

```c
int main (){
    int i = read_input();

    if (i > 0) {
        func(i);
    }
}

void func(int i) {
   if (i <= 0) {
        bug();
    }
    //...
}
```

Path constraints:

\{i > 0; i <= 0\}

Negated check

Un-negating

UNSAT False BUG
Limitations of T-Fuzz (1)

➢ False crashes may hinder discovery of true bugs (L1)

Example: false crash hindering discovery of true bug

```c
FILE *fp = fopen(...);
if (fp != NULL) {
    // False crash
    fread(fp, ...);
    // ...
    // true bug
    bug();
}
```
Limitations of T-Fuzz (2)

➢ Transformation explosion (L2)
  ➢ Analogous to path explosion issue in symbolic execution

Original program

Transformed program

Transformed program

Transformed program

Transformed program

Transformed program

Transformed program

Transformed program

Transformed program

…….

…….

…….

…….

…….
Evaluation

➢ DARPA CGC dataset
➢ LAVA-M dataset
➢ 4 real-world programs
CGC dataset

- Improvement over Driller/AFL: **55 (45%)/61 (58%)**
- T-Fuzz is defeated by Driller in 10
  - due to false crashes (L1) in 3
  - due to transformation explosion (L2) in 7

### Table: Method Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th># of bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFL</td>
<td>105</td>
</tr>
<tr>
<td>Driller</td>
<td>121</td>
</tr>
<tr>
<td>T-Fuzz</td>
<td>166</td>
</tr>
<tr>
<td>Driller - AFL</td>
<td>16</td>
</tr>
<tr>
<td>T-Fuzz - AFL</td>
<td>61</td>
</tr>
<tr>
<td>T-Fuzz - Driller</td>
<td>55</td>
</tr>
<tr>
<td>Driller - T-Fuzz</td>
<td>10</td>
</tr>
</tbody>
</table>
LAVA-M dataset

- T-Fuzz performs well given conditions favorable for VUzzer and Steelix
- T-Fuzz outperforms VUzzer and Steelix for “hard” checks
- T-Fuzz was defeated by Steelix due to transformation explosion in who
- T-Fuzz found 1 unintended bug in who

<table>
<thead>
<tr>
<th>Program</th>
<th>Total # of bugs</th>
<th>VUzzer</th>
<th>Steelix</th>
<th>T-Fuzz</th>
</tr>
</thead>
<tbody>
<tr>
<td>base64</td>
<td>44</td>
<td>17</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>unique</td>
<td>28</td>
<td>27</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>md5sum</td>
<td>57</td>
<td>1</td>
<td>28</td>
<td>49</td>
</tr>
<tr>
<td>who</td>
<td>2136</td>
<td>50</td>
<td>194</td>
<td>95*</td>
</tr>
</tbody>
</table>
Real-world programs

➢ Widely evaluated in related work
➢ T-Fuzz detected far more (verified) crashes than AFL
➢ T-Fuzz found 3 new bugs

<table>
<thead>
<tr>
<th>Program + library</th>
<th>AFL</th>
<th>T-Fuzz</th>
</tr>
</thead>
<tbody>
<tr>
<td>pngfix + libpng (1.7.0)</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>tiffinfo + libtiff (3.8.2)</td>
<td>53</td>
<td>124</td>
</tr>
<tr>
<td>magick + ImageMagicK (7.0.7)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>pdftohtml + libpoppler (0.62.0)</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusion & future work

➢ Fuzzers are limited by coverage and unable to find “deep” bugs
➢ T-Fuzz extend fuzzing by “mutating” the target program as well
➢ **Experimental results show that T-Fuzz is more effective than state-of-art fuzzers**
   - T-Fuzz has improvement over Driller/AFL by 45%/58%
   - T-Fuzz was able to trigger bugs guarded by “hard” checks
   - T-Fuzz found new bugs: 1 in LAVA-M dataset and 3 in real world programs

➢ Future work
   - Improve transformation strategies
   - Improve filtering of false positives

https://github.com/HexHive/T-Fuzz