T-Fuzz: fuzzing by program transformation

Hui Peng
Purdue University

Yan Shoshitaishvili
Arizona State University

Mathias Payer
Purdue University

Presented by:

Brian Kilcullen

Daniel Park
Forecast

- Overview of fuzzers
- Intuition of T-Fuzz Design
- T-Fuzz Algorithm Design & Implementation
- Testing Results
- Critique
Fuzzing as a bug finding approach

- What is fuzzing?
- Useful for hackers and software vendors.
- Two main types
  - Generational
  - Mutational

Firefox Logo
Download Chrome
OpenSSL
Wireshark
Challenges for fuzzers

- Challenges
  - Shallow coverage
  - Difficult to find “deep” bugs

- Why?
  - Cannot bypass complex sanity checks

<table>
<thead>
<tr>
<th>Header(4)</th>
<th>Keys(95)</th>
<th>Len(4)</th>
<th>Data(Len)</th>
<th>CRC(4)</th>
</tr>
</thead>
</table>

C1: header

C2: key

C3: CRC

BUG
Existing approaches & limitations

- **AFL**
  - Makes use of heuristics
  - Corpus generation

- **Driller**
  - Selective concolic execution when stuck
  - Mix of CONcrete and symbOLIC execution

- **VUzzer**
  - Taint analysis
  - Data control flow analysis
Existing approaches & limitations

● Steelix
  ○ Uses comparison progress

● Learn&Fuzz
  ○ Learns distribution of valid inputs

● Flayer
  ○ Program transformation with manual effort
Existing approaches & limitations

- Existing approaches focus on input generation.
- Limitations
  - A lot of overhead
  - Not scalable
  - Cannot reliably bypass hard checks
Intuition: Non-Critical Checks

- Sanity checks in context of fuzzing
  - Non-critical checks
  - Critical check
- Removing NCCs
  - Does not trigger spurious bugs
  - Simplifies fuzzing
  - Found bugs are reproducible
  - May accidentally remove CC

```c
void main() {
    int fd = open(...);
    char *hdr = read_header(fd);
    if (strcmp(hdr, "ELF", 3) == 0) {
        // main program logic
        // ...
    } else {
        error();
    }
}
```
Intuition: Program Transformation

- Used when fuzzer gets “stuck”
- Steps
  - Detect possible NCCs
  - Transforms programs to remove NCCs
- Crash Analyzer verifies crashes
T-Fuzz: Detection of Non-Critical Checks

- Over-approximation: checks that fuzzer’s inputs can’t bypass
  - Found via dynamic tracing
  - False positives (critical checks) likely included
- **Candidates** found as boundary edges in the CFG
- Some candidates are filtered out
  - Those not in the code of interest
  - Those causing immediate termination
T-Fuzz: Transformation of the Program

- Flip/negate the conditional check of an NCC candidate
  - Less overhead than DBI
  - Less complex than static binary translation
- Simple mapping between original and transformed program
T-Fuzz: Verification of Detected Bugs

- **Goal:** Prove the bug was in the original program
- **Constraint-based; “transformation-aware”**
  - CO: original program constraints
  - CT: transformed program constraints
- **Attempts to satisfy:**
  - Crashing input (CT)
  - Path constraints leading to crash address
  - Cause of Crash (CO)
T-Fuzz: Implementation Details

- Written in Python
- Fuzzer component built on AFL
- Transformer uses angr tracer and radare2
- Crash analyzer implemented with angr
Testing: DARPA CGC

- 296 vulnerable binaries analyzed
- Time constraint: 24 hours
- All fuzzers given seed “fuzz”
- Driller tested with double resources
  - AFL and T-Fuzz used one CPU core
  - Driller used two CPU cores

![Venn diagram of bug finding results](image)

**TABLE I: Details of experimental results**

<table>
<thead>
<tr>
<th>Method</th>
<th>Number of Binaries</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>Driller \ T-Fuzz</td>
<td>10</td>
</tr>
<tr>
<td>T-Fuzz \ Driller</td>
<td>45</td>
</tr>
</tbody>
</table>
Testing: LAVA-M

- Uses coreutils binaries with injected bugs
- Competitors excel only with hardcoded tests
- T-Fuzz generally finds more bugs (time permitting)

<table>
<thead>
<tr>
<th>program</th>
<th>Total # of bugs</th>
<th>FUZZER</th>
<th>SES</th>
<th>VUzzer</th>
<th>Steelix</th>
<th>T-Fuzz</th>
</tr>
</thead>
<tbody>
<tr>
<td>base64</td>
<td>44</td>
<td>7</td>
<td>9</td>
<td>17</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>md5sum</td>
<td>57</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>28</td>
<td>49</td>
</tr>
<tr>
<td>uniq</td>
<td>28</td>
<td>7</td>
<td>0</td>
<td>27</td>
<td>24</td>
<td>26</td>
</tr>
<tr>
<td>who</td>
<td>2136</td>
<td>0</td>
<td>18</td>
<td>50</td>
<td>194</td>
<td>63</td>
</tr>
</tbody>
</table>
Testing: Real Programs

- Program+library pairs fuzzed for 24 hours
- Used random 32-byte seeds
- Only compared to base AFL
- (*) denotes a newly-discovered bug

<table>
<thead>
<tr>
<th>Program</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>
Open Issues

- False positives obscuring real bug paths
- "Transformation explosion" / "transformal explosion"
- Scalability issues angr’s symbolic execution

Fig. 6: The transformed binaries T-Fuzz generates and fuzzes
Critique: Positives

● Generally thorough testing and evaluation
  ○ Took care to acknowledge shortcomings
  ○ Provided sensible reasons behind result differences
● Clear explanations and diagrams throughout
● Open source: https://github.com/HexHive/T-Fuzz
● Justified design decisions (e.g. allowing false negatives)
Critique: Negatives

- Needs more in-depth analysis on getting “stuck”
- Difficult to translate experimental set-up to usability
- Dismissive towards related program transformation approaches
Critique: Possible Ideas

- **Use machine learning**
  - Transformed program queue
  - Detecting candidate NCC’s and filtering CC’s

- **Improve flexibility of reporting**
  - Options for showing eliminated false positives
  - Operators’ usability was already an emphasized goal
Questions?