UTOPIA: Automatic Generation of Fuzz Driver using Unit Tests

Bokdeuk Jeong, Joonun Jang, Hayoon Yi, Jiin Moon, Junsik Kim, Intae Jeon, Taesoo Kim, WooChul Shim, Yong Ho Hwang

bd.jeong, joonun.jang, hayoon.yi, jiin.moon, junsik1.kim, intae.jeon, tsgates.kim, woochul.shim, yongh.hwang@samsung.com

Samsung Research Georgia Institute of Technology
Fuzz testing is an effective way to uncover bugs in libraries that may not be found through traditional testing techniques.

Hurdles in Library Fuzzing: Scalability

But, requires manually writing fuzz drivers.

- Time consuming and labor intensive
  - #APIs X #Libs = ?
  - Project even evolves! Not just a one-time task

Manual human efforts for library fuzzing does not scale well.

Taking the next step: OSS-Fuzz in 2023

February 1, 2023

Since launching in 2016, Google's free OSS-Fuzz code testing service has helped get over 8800 vulnerabilities and 28,000 bugs fixed across 850 projects. Today, we're happy.

Existing Work for Automatic Generation

Synthesize a sequence of APIs from consumer code by

1) inferring API dependences through the static analysis
2) or observing their uses at runtime

Semantically invalid states
Root Cause of the Limitation: API Misuse

Using APIs properly means
- Valid API call sequences
- Valid API call parameters

Adhering to valid API usage helps avoid
- exploring libraries in invalid and uninteresting states.
- producing spurious crashes.
  : unseen in end-to-end binary fuzzing


Issue 1722: Bug report on libvpx( AddressSanitizer: SEGV on unknown address)
Reported by afoss...@gmail.com on Thu, Mar 18, 2021, 1:17 PM GMT+9

Only an application using the API incorrectly would be at risk of a crash
Challenges & Approach Ideas

C1: Valid API sequence

Key Idea 1: No Inference. Convert UT to fuzz driver.

```c
bool unit_test() {
    struct A *a = CREATE(10);
    int sum = SUM(a, 1, 2);
}
```

```c
bool fuzz_test() {
    struct A *a = CREATE(fuzz1);
    int sum = SUM(a, fuzz2, fuzz3);
}
```

C2: Fuzz input as a semantically valid API argument

Key idea2: Parameter attribute analysis + root-definition analysis

C3: UT specific challenges

- How to handle the assertion checks?

Key idea3: Explore several basic strategies to make choices

```c
bool test() {
    struct A *a = CREATE(10);
    assert_neq(a, NULL); // if ignored, will crash on #4
    int sum = SUM(a, 1, 2);
    assert_eq(sum, 3); // if enforced, can it reach #6?
    do-something
}
```
Our Approach: Why UT instead of Consumer Code?

<table>
<thead>
<tr>
<th>Code Extraction</th>
<th>Consumer code</th>
<th>UT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program slicing: not practical</td>
<td><img src="image1.png" alt="Code Extraction Diagram" /></td>
<td><img src="image2.png" alt="UT Diagram" /></td>
</tr>
<tr>
<td>API uses</td>
<td>Too small or too large</td>
<td>TC1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>open()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>seek()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>read()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>close()</td>
</tr>
</tbody>
</table>

1. Simple: extract test functions only
2. Carefully crafted APIs only
3. Increase the fuzzing adoption using UT

<table>
<thead>
<tr>
<th>Opportunity</th>
<th>Category</th>
<th>Total #</th>
<th>Unit Tested # (rate)</th>
<th>Fuzz Tested # (rate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub C/C++ (Top 200)</td>
<td>200</td>
<td>143 (72%)</td>
<td>38 (19%)</td>
<td></td>
</tr>
<tr>
<td>Android External</td>
<td>316</td>
<td>224 (71%)</td>
<td>61 (19%)</td>
<td></td>
</tr>
<tr>
<td>OSSFuzz</td>
<td>450</td>
<td>347 (77%)</td>
<td>450 (100%)</td>
<td></td>
</tr>
</tbody>
</table>
Detail: Fuzz Input w/ Valid Parameter Semantic

1. Give constraints to fuzz input.

```c
void API(int *P) {
    *P = 0;
}
```

- fuzzing an output param → ineffective

```c
void API(int P) {
    void buf = malloc(P);
}
```

- fuzzing malloc() → undesirable crash

---

< Data flow analysis on API parameters >

```
void API(x1) {
    x2 = x1 - 3;
    if (x2 < 3)
        y1 = x2 * 2;
    else
        y2 = x2 - 3;
    y3 = f(y1, y2);
    w2 = x2 - y3;
}
```

```
void API(int P) {
    *P = 0;
}
```

- Used as the operand of store instruction

```
void API(int P) {
    void *buf = malloc(P);
}
```

- Argument flows into malloc()

---

**Fuzz input constraint**

- Output: Don’t fuzz
- LoopCount: limit the maximum input
- AllocSize: limit the maximum input
- FilePath: Deliver fuzz input as the file content
- Array ↔ Len
  ```
  int *var1 = Fuzz.data();
  API(var1, Fuzz.size());
  ```
2. Assign fuzz input while maintaining valid parameter semantic.

Valid parameter relationships between APIs are already well expressed in UT.

**How to reflect the relationships to the fuzz input?**
2. Assign fuzz input while maintaining valid parameter semantic.

```cpp
bool fuzz_test() {
    int a = fuzz_input_A;
    API1(a);
    API2(a);
    int b = a = fuzz_input_B;
    int c = API3(b);
    API4(c);
}
```

Assign fuzz input to the root-definitions

```cpp
bool unit_test() {
    int a = 10;  // Root-definition of a
    API1(a);
    API2(a);
    int b = a = 20;  // Root-definition of c
    int c = API3(b);
    API4(c);
}
```

Following the use-def chain
Evaluation

Scalability of the generation of fuzz drivers:
5K working fuzz drivers generated from 55 OSS and Tizen open source projects
- 25 OSS projects: 2,715 fuzz drivers (1 fuzz driver in 15 per-core secs)
- 30 Tizen projects: 2,699 fuzz drivers

Fuzzing capability
123 bugs found
- 109 bugs in the OSS-Fuzz projects
- 14 in the Tizen projects

Fuzzing Evaluation Setup
- Libfuzzer w/ fork-mode + Asan
- Ran fuzz drivers with the seeds extracted from UT
- Results averaged over 10 fuzzing runs

<table>
<thead>
<tr>
<th>Project</th>
<th>Bugs or CVEs</th>
<th>Category</th>
<th>Fuzzer</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCV</td>
<td>#21947</td>
<td>bof</td>
<td>readnetfromtensorflow_fuzzer.cc</td>
</tr>
<tr>
<td>OpenCV</td>
<td>#21852</td>
<td>bof</td>
<td>readnetfromtensorflow_fuzzer.cc</td>
</tr>
<tr>
<td>OpenCV</td>
<td>#21851</td>
<td>bof</td>
<td></td>
</tr>
<tr>
<td>libaom</td>
<td>CVE-2021-30473</td>
<td>free</td>
<td></td>
</tr>
<tr>
<td>libaom</td>
<td>CVE-2021-30474</td>
<td>bof</td>
<td></td>
</tr>
<tr>
<td>libaom</td>
<td>CVE-2021-30475</td>
<td>nullchk</td>
<td></td>
</tr>
<tr>
<td>uriparser</td>
<td>CVE-2021-46141</td>
<td>nullchk</td>
<td></td>
</tr>
<tr>
<td>uriparser</td>
<td>CVE-2021-46142</td>
<td>nullchk</td>
<td></td>
</tr>
<tr>
<td>libwebsocket</td>
<td>#2687</td>
<td>segfault</td>
<td>lws_upng_inflate_fuzzer.cpp</td>
</tr>
<tr>
<td>libhttp</td>
<td>#342</td>
<td>segfault</td>
<td></td>
</tr>
<tr>
<td>libhttp</td>
<td>#343</td>
<td>segfault</td>
<td></td>
</tr>
<tr>
<td>libphononumber</td>
<td>#20146814</td>
<td>nullchk</td>
<td></td>
</tr>
<tr>
<td>libphononumber</td>
<td>#20147539</td>
<td>segfault</td>
<td></td>
</tr>
<tr>
<td>libvpx</td>
<td>#1742</td>
<td>arith</td>
<td></td>
</tr>
<tr>
<td>libvpx</td>
<td>#1722</td>
<td>free</td>
<td></td>
</tr>
<tr>
<td>tesseract</td>
<td>#3583</td>
<td>bof</td>
<td></td>
</tr>
<tr>
<td>tesseract</td>
<td>#3584</td>
<td>bof</td>
<td></td>
</tr>
<tr>
<td>tesseract</td>
<td>#3586</td>
<td>segfault</td>
<td></td>
</tr>
<tr>
<td>tesseract</td>
<td>#3694</td>
<td>arith</td>
<td></td>
</tr>
<tr>
<td>wabt</td>
<td>#1793</td>
<td>segfault</td>
<td></td>
</tr>
<tr>
<td>wabt</td>
<td>#1794</td>
<td>oom</td>
<td></td>
</tr>
<tr>
<td>aosp_audio_util</td>
<td>#20667585</td>
<td>arith</td>
<td></td>
</tr>
<tr>
<td>tizen/bltbnm</td>
<td>#2543B2</td>
<td>nullchk</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation: Fuzzing Effectiveness

Comparison with the manual fuzz drivers from OSS-Fuzz

<table>
<thead>
<tr>
<th></th>
<th>libvpx</th>
<th>libaom</th>
<th>Node.js</th>
<th>openh264</th>
<th>jsonnet</th>
<th>Tesseract</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSS-Fuzz</td>
<td>1 (7)</td>
<td>1 (5)</td>
<td>2 (32)</td>
<td>1 (7)</td>
<td>1 (5)</td>
<td>1 (9)</td>
</tr>
<tr>
<td>UTopia</td>
<td>40 (43)</td>
<td>115 (109)</td>
<td>42 (60)</td>
<td>113 (132)</td>
<td>45 (6)</td>
<td>240 (356)</td>
</tr>
</tbody>
</table>

# Drivers (APIs tested)

* UTopia generated fuzz drivers from open264 and jsonnet test an internal function of API or an uninteresting API argument for logging.
Evaluation: Fuzzing Effectiveness

Unique coverage of UTopia compared with UT

<table>
<thead>
<tr>
<th></th>
<th>Libvpx</th>
<th>libaom</th>
<th>Node.js</th>
<th>Openh264</th>
<th>jsonnet</th>
<th>Tesseract</th>
</tr>
</thead>
<tbody>
<tr>
<td>max growth vs UT</td>
<td>x2.0</td>
<td><strong>x37.5</strong></td>
<td>x2.0</td>
<td>x1.3</td>
<td>x14.9</td>
<td>x3.2</td>
</tr>
</tbody>
</table>

Uncovered 3yrs old bugs in libaom with new exploration based on UT

<table>
<thead>
<tr>
<th>E001</th>
<th>Task</th>
<th>Assigned</th>
<th>Medium</th>
<th>----</th>
<th><a href="mailto:jz...@gmail.com">jz...@gmail.com</a></th>
<th>Please remove the <em>realtime-only</em> configuration from libaom_unit_tests-multi-2a</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>Defect</td>
<td>Assigned</td>
<td>Critical</td>
<td>----</td>
<td><a href="mailto:brinkbeck@google.com">brinkbeck@google.com</a></td>
<td>Bug report on libaom</td>
</tr>
<tr>
<td>2999</td>
<td>Defect</td>
<td>Assigned</td>
<td>Critical</td>
<td>----</td>
<td><a href="mailto:brinkbeck@google.com">brinkbeck@google.com</a></td>
<td>Bug report on libaom</td>
</tr>
<tr>
<td>2998</td>
<td>Defect</td>
<td>Assigned</td>
<td>Critical</td>
<td>----</td>
<td><a href="mailto:chyotsui@gmail.com">chyotsui@gmail.com</a></td>
<td>Bug report on libaom</td>
</tr>
</tbody>
</table>

UTs are designed to check what developers expected to be correct, but fuzzers focus on what they didn’t expect. ← Nicely bridged by UTopia.
Evaluation : Exploration of the Assertion

No silver bullet for all types of assertions

A single fuzz driver per project was selected and ran 10 times for 12 per-core hours

```
struct A *a = CREATE(fuzz);
assert_neq(a, NULL);
```

```
INIT(&obj, fuzz);
assert_eq(obj.member, 3);
```

A single fuzz driver per project was selected and ran 10 times for 12 per-core hours.
Limitations

Some UTs test internal function instead of APIs.

```c
TEST_F(URLTest, Base6) {
  // libnode TC
  char* input = autofuzz276;  // autofuzz276 = "/\nStrace:\n"
  char* base = autofuzz277;  // autofuzz277 = "file://h\333\207eam`,ar"
  URL simple(input, strlen(input), base, strlen(base));  // crashes with SEGV
}
```

Insufficient error checks in UT make sense, but not in fuzzing

```c
pkt = aom_codec_get_cx_data(&enc_, &iter);
// no check on pkt
if (pkt->kind == AOM_CODEC_CX_FRAME_PKT) {  // if pkt is NULL, it will crash
```

Non-conventional relations of parameters

```c
printf("Hello %s", input);  // replace the first parameter with fuzz input?
```
How Is UTopia Being Utilized Now?

UTopia is open at https://github.com/Samsung/UTopia

Bug report with generated fuzz drivers

Fix vector access in TF::sortByExecutionOrder #22006
Add assert to address tf simplifier security concerns #21861

UTopia-generated fuzz drivers are merged to oss-fuzz with modification.

Initial submission

Merged after review with simplification
Q&A
APPENDIX
Evaluation: Mitigation of Spurious Crashes

Mitigation of undesirable crashes through static analysis

Crash Count

<table>
<thead>
<tr>
<th></th>
<th>assimp</th>
<th>libhtp</th>
<th>leveldb</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>114,790</td>
<td>641,105</td>
<td>462,948</td>
</tr>
<tr>
<td>w/o Loop</td>
<td>141,859</td>
<td>2,298</td>
<td>3,181</td>
</tr>
<tr>
<td>w/o Alloc</td>
<td>955,991</td>
<td>413,048</td>
<td>375,085</td>
</tr>
<tr>
<td>w/o Arr</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

1,167 unique crashes from the generated fuzz drivers using 25 OSS projects are collected. Each fuzz driver ran for 1 per-core hr.

Remaining crashes

- Timeout (429)
- oom, abort (143)
- Bugs (109)
- Not covered by UTopia (209)
- Intended to cover, but missed (160)
- etc (117)

Due to the analysis failure with loop count attribute: nested loop, loop count multiplication, loop count identification failure in C++ object

Due to the analysis failure with array/len in complex loop or multi-dimension

A single fuzz driver per project was selected and ran 10 times for 12 per-core hours.
### Evaluation: UTOpia-generated fuzz drivers

<table>
<thead>
<tr>
<th>Target Library</th>
<th>Unit Tests</th>
<th>Analyzed Test Cases</th>
<th>UTOpia-Generated Fuzz Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SR LoC BS</td>
<td>#eFn TF Cov TeCov</td>
<td>#TC Functions Time(sec) Cov/U Cov AG/MG</td>
</tr>
<tr>
<td>nodejs</td>
<td>O 3M gn</td>
<td>3,065</td>
<td>11.8% 11.5% 124</td>
</tr>
<tr>
<td>libamo</td>
<td>O 363K cm</td>
<td>5,065</td>
<td>51.5% 48.4% 682</td>
</tr>
<tr>
<td>asimp</td>
<td>O 356K gn</td>
<td>5,055</td>
<td>45.5% 23.9% 449</td>
</tr>
<tr>
<td>libvpx</td>
<td>O 248K gn</td>
<td>1,446</td>
<td>47.6% 27.7% 373</td>
</tr>
<tr>
<td>tesseract-ocr</td>
<td>O 158K cm</td>
<td>3,650</td>
<td>62.4% 57.1% 477</td>
</tr>
<tr>
<td>openh264</td>
<td>O 92K cm</td>
<td>1,523</td>
<td>61.3% 33.6% 320</td>
</tr>
<tr>
<td>libphonenumber</td>
<td>O 55K cm</td>
<td>510</td>
<td>65.2% 63.7% 324</td>
</tr>
<tr>
<td>wabt</td>
<td>O 47K cm</td>
<td>1,034</td>
<td>24.9% 24.6% 190</td>
</tr>
<tr>
<td>leveldb</td>
<td>O 21K cm</td>
<td>397</td>
<td>87.1% 86.0% 218</td>
</tr>
<tr>
<td>libhtp</td>
<td>O 20K gn</td>
<td>386</td>
<td>73.6% 73.3% 339</td>
</tr>
<tr>
<td>jsonnet</td>
<td>O 13K cm</td>
<td>98</td>
<td>35.9% 35.9% 45</td>
</tr>
<tr>
<td>uriparser</td>
<td>G 8K cm</td>
<td>42</td>
<td>90.5% 88.7% 92</td>
</tr>
<tr>
<td>mediapipe</td>
<td>G 225K bz</td>
<td>2,337</td>
<td>47.0% 37.5% 524</td>
</tr>
<tr>
<td>filament</td>
<td>G 64K nj</td>
<td>5,948</td>
<td>32.8% 25.0% 292</td>
</tr>
<tr>
<td>nuduo</td>
<td>G 16K cm</td>
<td>359</td>
<td>15.3% 19.9% 30</td>
</tr>
<tr>
<td>vowpal_wabbit</td>
<td>G 81K cm</td>
<td>1,383</td>
<td>20.3% 14.8% 224</td>
</tr>
<tr>
<td>ledger</td>
<td>G 51K cm</td>
<td>32</td>
<td>9.3% 9.3% 17</td>
</tr>
<tr>
<td>cpuinfo</td>
<td>A 423K cm</td>
<td>66</td>
<td>54.2% 15.6% 142</td>
</tr>
<tr>
<td>minijail</td>
<td>A 16K gn</td>
<td>162</td>
<td>54.1% 47.9% 189</td>
</tr>
<tr>
<td>phtreadpool</td>
<td>A 12K cm</td>
<td>54</td>
<td>69.3% 69.3% 291</td>
</tr>
<tr>
<td>cpu_features</td>
<td>A 6K cm</td>
<td>36</td>
<td>51.5% 9.4% 31</td>
</tr>
<tr>
<td>puflin</td>
<td>A 5K cm</td>
<td>92</td>
<td>83.6% 66.3% 44</td>
</tr>
<tr>
<td>bsdiff</td>
<td>A 4K cm</td>
<td>137</td>
<td>57.2% 43.4% 66</td>
</tr>
<tr>
<td>sfmtly</td>
<td>A 23K cm</td>
<td>897</td>
<td>48.7% 48.2% 23</td>
</tr>
<tr>
<td>snappy</td>
<td>T 6K gn</td>
<td>46</td>
<td>75.9% 75.9% 17</td>
</tr>
</tbody>
</table>

Total: - 53M - 33,720 - - - 5,523 1,039 1,769 2,715 (100%) 2,292 1,277 15.6hr 6min - -