ASanity: On Bug Shadowing by Early ASan Exits

Vincent Ulitzsch vincent@sect.tu-berlin.de
Deniz Scholz deniz.scholz@t-online.de
Dominik Maier dmaier@sect.tu-berlin.de
Fuzzers add compile-time instrumentation sanitizers to enhance their bug capabilities.

AddressSanitizer identifies illegitimate memory accesses, but aborts program execution after the first bug.

ASan’s early exits can hide bugs, as we show through a large-scale study.
Fuzz-testing relies on detecting crashing test cases

1. **Seed the fuzzing engine with valid program input**
   - **a** Seed the fuzzing engine with valid program input.

2. **Fuzzing engine takes some program input, mutates it, runs it against the target**
   - **b** Fuzzing engine takes some program input, mutates it, and runs it against the target.

3. **Fuzzing engine observes behavior and saves interesting testcases, e.g., crashing inputs**
   - **c** Fuzzing engine observes behavior and saves interesting testcases, e.g., crashing inputs.

4. **Add inputs that yield new coverage to input queue**
   - **d** Add inputs that yield new coverage to input queue.

**Sanitizers help with detecting bug-triggering inputs!**
AddressSanitizer helps to analyze crashes

Vulnerable Program

```c
void swap(char *left, char *right, int len) {
    // Call with len=size(right)
    char tmp[len];
    // Potential OOB read if len>size(left)
    memcpy(tmp, left, len);
    [...]  
}
```

AddressSanitizer output gives information about the cause of the bug

```
==3955==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x6100000001f5 at pc 0x5558ca920c3e bp 0x7ffd85b1b390 sp 0x7ffd85b1ab40 READ of size 16 at 0x6100000001f5 thread T0  
    #0 0x5558ca920c3d in __interceptor_memcpy.part.0  
    #1 0x5558ca966533 in swap  
    #2 0x5558ca96082a in __libc_start_main
```

ASan adds instrumentation during compile time to detect memory corruption errors during runtime. Gives information about crash-type, access type and byte-size of the violation.
ASan’s early exit behavior can hide bugs

Vulnerable Program

```c
void swap(char *left, char *right, int len)
{
    char tmp[len];
    // Potential OOB read
    memcpy(tmp, left, len);
    // OOB write shadowed by early exit
    memcpy(left, right, len);
    memcpy(right, tmp, len);
}
```

- **ASan** – by default – **aborts program execution early** (on the first bug).
- This **can hide bugs** later in the program flow.
- But: This behavior can be disabled via a compiler flag.
ASan’s early exits could lead to wrong bug prioritization

- ASan’s output is used to assign severities, and thus, priorities in large-scale fuzzing campaigns.
- An underestimated severity can lead to lower priority.
- Or worse: Once the out-of-bounds read is fixed, the testcase might not trigger the out-of-bounds write anymore – the bug will be missed!
Research Question

Do ASan early exits impact our bug-finding capabilities in practice?
Large Scale Study: Based On OSS-Fuzz

OSS-Fuzz

1. Submit project + harness to OSS-Fuzz repository

Pull project + fuzz

2.

Clusterfuzz

3.

Developer

Report bug + testcase to developer

4.

Monorail Issue Tracker

- OSS-Fuzz: Framework for continuously fuzzing open-source projects in ClusterFuzz, distributed fuzzing environment.
- Focus on heap buffer overflow out-of-bounds Read (OOB-R) issues:
  - RQ: Do the testcases also trigger an OOB-Write or use-after-free?
Monorail gives us detailed information about a bug

Monorail output gives crash type

And assign severities
Experiment Design

1. **Scrape Monorail bugtracker issues**
   - Received data for 44k issues, spanning around ~500 projects
   - Based on scraper by Ding. et. al. in 2021

2. **Filter for heap OOB-reads**
   - 1986 OOB-read issues, 1788 reproducible

3. **Recompile harness, early exists disabled**
   - Could recompile 814 examples, spanning 159 projects

4. **Re-execute triggering input**

5. **Collect results**
   - 5% of the issues trigger a more severe bug!

Based on scraper by Ding. et. al. in 2021.
Resulting Data

- For 23/159 projects: At least one testcase also triggers a use-after-free or heap OOB-W
  - 19/159 projects: At least one testcase additionally triggered an OOB-W
  - 8/159 projects: At least one testcase additionally triggered a use-after-free
- In total almost 5% (38/814) heap OOB-R issues also triggered an OOB-W or use-after-free
- Detailed listing also in the paper

<table>
<thead>
<tr>
<th>Projects</th>
<th>OOB Reads</th>
<th>OOB Writes</th>
<th>Use-After-Frees</th>
</tr>
</thead>
<tbody>
<tr>
<td>libdwarf</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>libsass</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ghostscript</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>botan</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>wasm3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>leptonica</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>muhy</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>inchi</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>58</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>openh264</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>net-snmp</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tdengine</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>muffparser</td>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>dav1d</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>libreoffice</td>
<td>20</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>libhttp</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>openjpeg</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>grok</td>
<td>15</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>suricata</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ndpi</td>
<td>59</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>php</td>
<td>9</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>libredwg</td>
<td>18</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>libheif</td>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
Case Study: Two bugs in the wasm3 interpreter (1/2)

```c
M3Result InitDataSegments (M3Memory * io_memory, IM3Module io_module) {
    [...]  
    i64 segmentOffset;
    //Read segmentOffset from wasm file
    if (((size_t)(segmentOffset) + segment->size <= io_memory->mallocated->length))
        u8 * dest = m3MemData (io_memory->mallocated) + segmentOffset;
        memcpy (dest, segment->data, segment->size); //OOB-R here
}
```

```c
M3Result ParseSection_Data (M3Module * io_module, [...]) {
    [...]  
    //Segment size is attacker controlled
    segment->data += segment->size;
    //Fix: _throwif("", segment->data > segment_end);
    [...]  
}
```

- We conducted case study on the wasm3 interpreter – issue reported as a heap-buffer OOB-R
- Fix will abort execution in case of OOB-R
Case Study: Two bugs in the wasm3 interpreter (2/2)

M3Result InitDataSegments (M3Memory * io_memory, IM3Module io_module) {
    [...] 
    i64 segmentOffset;
    //Read segmentoffset from wasm file
    if (((size_t)(segmentOffset) + segment->size <= io_memory->mallocated->length) 
        u8 * dest = m3MemData (io_memory->mallocated) + segmentOffset;
        memcpy (dest, segment->data, segment->size); //OOB-R and OOB-W here 
    }

The OOB-R shadowed an OOB-W in the InitDataSegments section
When fixed, our testcase will not trigger the OOB-W anymore: Bug could remain hidden!
Paper: We show how to exploit the OOB-W for code execution
Conclusion

1. ASan’s early-exits indeed shadow more severe bugs.

2. 5% of OSS-fuzz testcases also triggered more severe bug.

3. Further fuzzing campaigns should consider disabling ASan’s early-exits.

Thank you for your attention!

vincent@sect.tu-berlin.de

https://github.com/fgsect/asanity