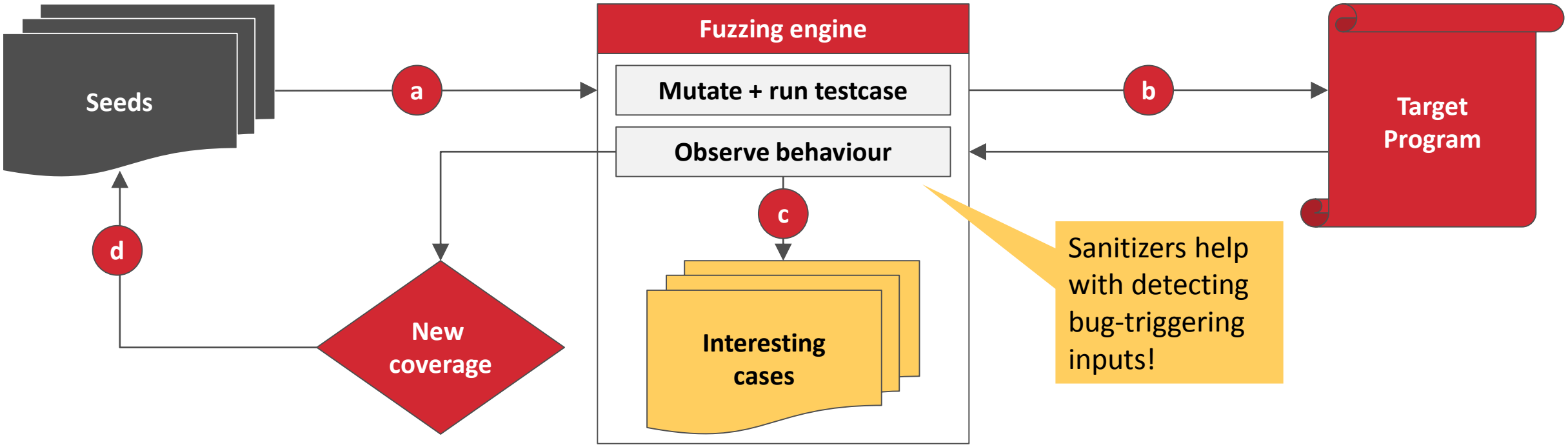
ASanity: On Bug Shadowing by Early ASan Exits

Vincent Ulitzsch vincent@sect.tu-berlin.de
Deniz Scholz deniz.scholz@t-online.de
Dominik Maier dmaier@sect.tu-berlin.de

Summary

1	Fuzzers add compile-time instrumentation *sanitizers* to enhance their bug capabilities.
2	AddressSanitizer identifies illegitimate memory accesses, but aborts program execution after the first bug.
3	ASan’s early exits can hide bugs, as we show through a large-scale study.

Fuzz-testing relies on detecting crashing test cases



a Seed the fuzzing engine with valid program input

b Fuzzing engine takes some program input, mutates it, runs it against the target

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

d Add inputs that yield new coverage to input queue

AddressSanitizer helps to analyze crashes

Vulnerable Program

```
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

```
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 aborts program execution here

And thus misses a more severe bug here

- **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.

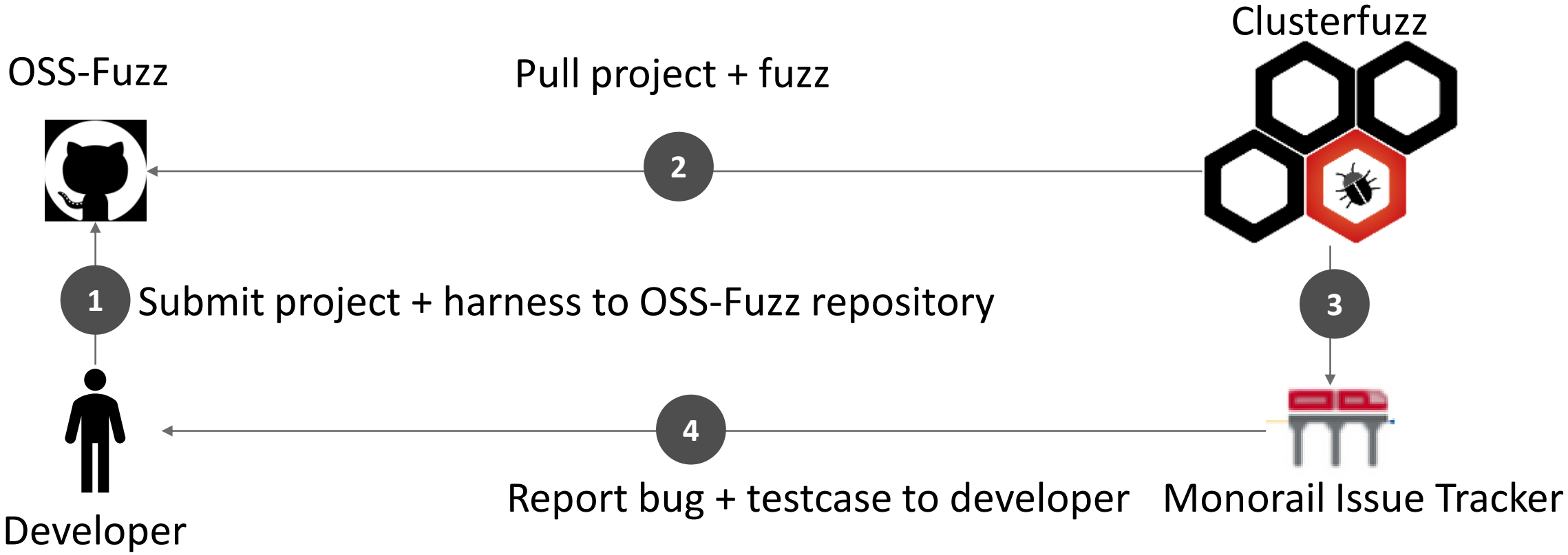
ASan’s early exits could lead to wrong bug prioritization

- 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: 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

oss-fuzz oss-fuzz ▾ | New issue | Open issues ▾ | Search oss-fuzz issues…

☆ Starred by 1 user

Owner: ----

CC: a...@adalogics.com
vshym...@gmail.com

Status: Verified *(Closed)*

Components: ----

Modified: Apr 15, 2021

Type: Bug-Security

ClusterFuzz
Stability-Memory-AddressSanitizer
Reproducible
ClusterFuzz-Verified
Engine-libfuzzer
OS-Linux
Security_Severity-Medium
Proj-wasm3
Reported-2021-04-14
Disclosure-2021-07-13

Issue 33237: wasm3:fuzzer: Heap-buffer-overflow in m3_LoadModule

Reported by ClusterFuzz-External on Wed, Apr 14, 2021, 1:54 AM PDT **Project Member**

Detailed Report: https://oss-fuzz.com/testcase?key=6745255706755072

Project: wasm3
Fuzzing Engine: libFuzzer
Fuzz Target: fuzzer
Job Type: libfuzzer_asan_wasm3
Platform Id: linux

Crash Type: Heap-buffer-overflow READ {*}
Crash Address: 0x6070000000db
Crash State:
  m3_LoadModule
  fuzzer.c

Monorail output gives crash type

Sanitizer: address (ASAN)

Recommended Security Severity: Medium

And assign severities

Crash Revision: https://oss-fuzz.com/revisions?job=libfuzzer_asan_wasm3&revision=202104140601

Reproducer Testcase: https://oss-fuzz.com/download?testcase_id=6745255706755072

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!

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

Projects	OOB Reads	OOB Writes	Use-After-Frees
libdwarf	1	1	0
libsass	1	1	0
ghostscript	9	1	0
botan	2	1	0
wasm3	1	1	0
leptonica	16	1	1
mruby	5	1	0
inchi	3	0	1
ffmpeg	58	4	1
openh264	4	2	0
net-snmp	6	1	0
tdengine	3	2	0
muparser	7	3	0
dav1d	2	1	0
libreoffice	20	1	1
libhtp	1	0	1
openjpeg	3	1	0
grok	15	2	0
suricata	4	0	1
ndpi	59	0	2
php	9	2	1
libredwg	18	1	0
libheif	7	3	0

Case Study: Two bugs in the wasm3 interpreter (1/2)

```
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 here
    }
}
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
    }
}
```

Integer overflow allows us to write into the dest pointer

The OOB-W is shadowed by the OOB-R in the same line.

- 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.

https://github.com/fgsect/asanity

Thank you for your attention!

vincent@sect.tu-berlin.de