Fuzzing Binaries for Memory Safety Errors with QASan

Andrea Fioraldi, Daniele Cono D’Elia, Leonardo Querzoni
Fuzz Testing or Fuzzing

- A very effective random testing technique that discovered thousands of bugs
Challenges

- Trigger as many faults as possible in a given time window
Coverage-guided Fuzzing

Corpus → Input Mutation → Program Under Test

Coverage → Crashes
Challenges

● Trigger as many faults as possible in a given time window
  ○ Coverage-guided Fuzzing
Challenges

- Trigger as many faults as possible in a given time window
  - Coverage-guided Fuzzing

- Observe the failure to know if a fault is triggered
Sanitization

Add tripwires to expose silent faults at runtime

- AddressSanitizer
- MemorySanitizer
- UndefinedBehaviourSanitizer
- ThreadSanitizer
- ...

Fuzzing Binaries for Memory Safety Errors with QASan
What about closed-source binaries?

- Get coverage with
  - Dynamic Binary Translation (QEMU, Intel PIN, DynamoRIO, ...)
  - Hardware support (Intel PT)
  - Static Rewriting (DynInst, e9patch, RetroWrite (x86_64 only), ...)
What about closed-source binaries?

● Get coverage with
  ○ Dynamic Binary Translation (QEMU, Intel PIN, DynamoRIO, ...)
  ○ Hardware support (Intel PT)
  ○ Static Rewriting (DynInst, e9patch, RetroWrite (x86_64 only), ...)

● Sanitize with
  ○ Static Rewriting (RetroWrite (x86_64 only))
Fuzzing with (AFL++) QEMU

- Block caching to parent process when forking
- Analyses for comparison instructions (CompareCoverage, CmpLog)
- 2x slowdown compared to afl-gcc in fork() mode, faster in persistent or snapshot mode
- Wide range of architectures (i386, ARM, MIPS, s390x, RISC-V, …)
- Stop execution without invoking the kernel scheduler (IPC-free fuzzing in the near future)
# Fuzzing with (AFL++) QEMU

<table>
<thead>
<tr>
<th>Process Timing</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time</td>
<td>Cycles done: 0</td>
</tr>
<tr>
<td>Last new path</td>
<td>Total paths: 176</td>
</tr>
<tr>
<td>Last uniq crash</td>
<td>Unique crashes: 0</td>
</tr>
<tr>
<td>Last uniq hang</td>
<td>Unique hangs: 0</td>
</tr>
<tr>
<td>Cycle progress</td>
<td></td>
</tr>
<tr>
<td>Now processing: 18.0 (10.2%)</td>
<td>Map coverage:</td>
</tr>
<tr>
<td>Paths timed out: 0 (0.00%)</td>
<td>Map density: 0.34% / 1.19%</td>
</tr>
<tr>
<td>Stage progress</td>
<td>Count coverage: 2.22 bits/tuple</td>
</tr>
<tr>
<td>Now trying: splice 10</td>
<td>Findings in depth:</td>
</tr>
<tr>
<td>Stage execs: 31/32 (96.88%)</td>
<td>Favored paths: 75 (42.61%)</td>
</tr>
<tr>
<td>Total execs: 15.3k</td>
<td>New edges on: 88 (50.00%)</td>
</tr>
<tr>
<td>Exec speed: 1731/sec</td>
<td>Total crashes: 0 (0 unique)</td>
</tr>
<tr>
<td>Fuzzing strategy yields</td>
<td>Total tmouts: 0 (0 unique)</td>
</tr>
<tr>
<td>Bit flips: n/a, n/a, n/a</td>
<td>Path geometry:</td>
</tr>
<tr>
<td>Byte flips: n/a, n/a, n/a</td>
<td>Levels: 3</td>
</tr>
<tr>
<td>Arithmetics: n/a, n/a, n/a</td>
<td>Pending: 170</td>
</tr>
<tr>
<td>Known ints: n/a, n/a, n/a</td>
<td>Pend fav: 70</td>
</tr>
<tr>
<td>Dictionary: n/a, n/a, n/a</td>
<td>Own finds: 175</td>
</tr>
<tr>
<td>Havoc/splice: 168/9536, 7/2912</td>
<td>Imported: n/a</td>
</tr>
<tr>
<td>Py/custom: 0/0, 0/0</td>
<td>Stability: 100.00%</td>
</tr>
<tr>
<td>Trln: 0.66%/1378, n/a</td>
<td></td>
</tr>
</tbody>
</table>

Fuzzing Binaries for Memory Safety Errors with QASan
Sanitize with QEMU?

- Sanitize libraries
- Fast instrumentation with DBT
- Shadow memory outside the guest
- Immediate setup
- Cannot sanitize stack and globals when binary-only :(
AddressSanitizer + QEMU = QASan
Fuzzing Binaries for Memory Safety Errors with QASan
Fuzzing Binaries for Memory Safety Errors with QASan

Instrument memory accesses to check for violations
Instrument calls to maintain a shadow call stack and track allocations contexts
Replace the allocator to clobber invalid regions in the shadow memory.
Hypercalls

- Fake syscall
- Backdoor

syscall(QASAN_FAKESYS_NR, action, arg1, arg2, arg3)
Hypercalls

```c
# void* qasan_backdoor(int, void*, void*, void*)

qasan_backdoor:
  mov rax, rdi  # action
  mov rdi, rsi  # arg1
  mov rsi, rdx  # arg2
  mov rdx, rcx  # arg3
  .byte 0x0f
  .byte 0x3a
  .byte 0xf2
  ret
```

- Fake syscall
- Backdoor
Hypercalls

```c
# void* qasan_backdoor(int, void*, void*, void*)

qasan_backdoor:
    mov rax, rdi  # action
    mov rdi, rsi  # arg1
    mov rsi, rdx  # arg2
    mov rdx, rcx  # arg3
    .byte 0x0f
    .byte 0x3a
    .byte 0xf2
    ret
```

- Fake syscall
- Backdoor
Symbol hooking

```c
char *strcpy(char *dest, const char *src) {
    size_t l = __libqasan_strlen(src) + 1;
    QASAN_LOAD(src, l);
    QASAN_STORE(dest, l);
    return __libqasan_memcpy(dest, src, l);
}
```

- Replace common libraries routines with checked versions
- Not needed when libraries are instrumented
Function hotpatching

- Some optimized libc functions speculate about page boundaries when reading buffers: this is generally fine but represents a violation for the sanitizer if libc is instrumented.
- At startup, QASan hotpatches critical functions in libc using trampolines. Symbol hooking is not enough, as the original implementation is still called from internal libc functions.
## Heap bugs detection (Juliet dataset, TN 50% FP 0%)

<table>
<thead>
<tr>
<th></th>
<th>QASan TP</th>
<th>QASan FN</th>
<th>ASan TP</th>
<th>ASan FN</th>
<th>Memcheck TP</th>
<th>Memcheck FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap-based Buffer</td>
<td>47.88</td>
<td>2.12</td>
<td>47.17</td>
<td>2.83</td>
<td>47.88</td>
<td>2.12</td>
</tr>
<tr>
<td>Overflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Double-Free</td>
<td>50.0</td>
<td>0.0</td>
<td>50.0</td>
<td>0.0</td>
<td>50.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Use-After-Free</td>
<td>50.0</td>
<td>0.0</td>
<td>50.0</td>
<td>0.0</td>
<td>50.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Freeing non-Heap Memory</td>
<td>49.98</td>
<td>0.0</td>
<td>50.0</td>
<td>0.0</td>
<td>50.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
More bugs, limited overhead

<table>
<thead>
<tr>
<th>Program</th>
<th>Reported bugs</th>
<th>Executions per second (avg)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>standard</td>
<td>QASan</td>
</tr>
<tr>
<td>c-ares</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>guetzli</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>json</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>libxml2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>openssl</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>pcre2</td>
<td>16</td>
<td>29</td>
</tr>
<tr>
<td>re2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>woff2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fuzzing Binaries for Memory Safety Errors with QASan
==3857==ERROR: QEMU-AddressSanitizer: heap-buffer-overflow on address 0x000000783660 at pc 0x0000004245ac
bp 0x0000004f1a40 sp 0x7f977e4799e0
READ of size 1 at 0x000000783660 thread T3857
#0 0x0000004245ac in xmlParseXMLDecl /home/andrea/Desktop/libxml2/parser.c:10666
#1 0x0000004247b1 in xmlParseDocument /home/andrea/Desktop/libxml2/libxml2.c:10772
#2 0x000000429fe4 in xmlDoRead /home/andrea/Desktop/libxml2/libxml2.c:15299
#3 0x000000406945 in parseAndPrintFile /home/andrea/Desktop/libxml2/xmllint.c:?
#4 0x0000004094315 in main /home/andrea/Desktop/libxml2/xmllint.c:3762
#5 0x7f977ca2b97 in __libc_start_main /build/glibc-OTsEL5/glibc-2.27/csu/../csu/libc-start.c:344
#6 0x7f977d8498f3 in __libc_start_main (/home/andrea/qasan/libqasan.so+0x28f3)
#7 0x0000004024aa in __start (/home/andrea/Desktop/libxml2/xmllint.orig+0x24aa)

0x000000783660 is located 0 bytes to the right of 4096-byte region [0x000000782660,0x000000783660)
allocated by thread T3857 here:
#0 0x7f977d84b57b in __libc_qasan_malloc (/home/andrea/qasan/libqasan.so+0x457b)
#1 0x7f977d84aee2 in malloc (/home/andrea/qasan/libqasan.so+0x2ae2)
#2 0x0000004090bbc in xmlBufCreate /home/andrea/Desktop/libxml2/libxml2.c:136
#3 0x0000004101a0 in xmlSwitchInputEncodingInt /home/andrea/Desktop/libxml2/parserInternals.c:1196
#4 0x0000004102aa in xmlSwitchToEncodingInt /home/andrea/Desktop/libxml2/parserInternals.c:1281
#5 0x000000416e43 in xmlParseEncodingDecl /home/andrea/Desktop/libxml2/libxml2.c:?
#6 0x000000424470 in xmlParseXMLDecl /home/andrea/Desktop/libxml2/parser.c:10631
#7 0x0000004247b1 in xmlParseDocument /home/andrea/Desktop/libxml2/libxml2.c:10772
#8 0x000000429fe4 in xmlDoRead /home/andrea/Desktop/libxml2/libxml2.c:15299
#9 0x000000406945 in parseAndPrintFile /home/andrea/Desktop/libxml2/xmllint.c:?
#10 0x000000404315 in main /home/andrea/Desktop/libxml2/xmllint.c:3762
#11 0x7f977ca2b97 in __libc_start_main /build/glibc-OTsEL5/glibc-2.27/csu/../csu/libc-start.c:344
#12 0x7f977d8498f3 in __libc_start_main (/home/andrea/qasan/libqasan.so+0x28f3)
#13 0x0000004024aa in __start (/home/andrea/Desktop/libxml2/xmllint.orig+0x24aa)

SUMMARY: QEMU-AddressSanitizer: heap-buffer-overflow in xmlParseXMLDecl /home/andrea/Desktop/libxml2/parser.c:10666
Shadow bytes around the buggy address:
0x6008000e8670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e86a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e86b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e86c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e86d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e86e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e86f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8700: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8710: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8720: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8730: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8740: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8750: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8760: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8770: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8780: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e8790: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e87a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e87b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e87c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e87d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e87e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x6008000e87f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

SUMMARY: QEMU-AddressSanitizer: heap-buffer-overflow in xmlParseXMLDecl /home/andrea/Desktop/libxml2/parser.c:10566

Shadow bytes around the buggy address:
0x600000e8d670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x600000e8d680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x600000e8d690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x600000e8d6a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x600000e8d6b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x600000e8d6c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x600000e8d6d0: fb fb fb fb fb fb fb fb fb fb fb fb 00 00 00 00
0x600000e8d6e0: 00 00 fa fa fa fa fa fa fa fa fa fa fa fa fa fa
0x600000e8d6f0: fa fa fd fd fd fd fd fd fd fd fd fd fd fd fd fd
0x600000e8d700: fd fd fd fd fd fb fb fb fb fb fb fb fb fb fb fb
0x600000e8d710: fb fb fb fb fb fb fb fb fb 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable: 06
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone: fa
Heap right redzone: fb
Freed heap region: fd
Poisoned by user: fb
ASan internal: fe
Shadow gap: cc
==3857==ABORTING
Future directions

- Full-system sanitization
- Other sanitizers
- Stack use-after-return detection
Thank You!  

https://github.com/andreafioraldi/qasan
Thank You!  

https://github.com/andreafioraldi/qasan