

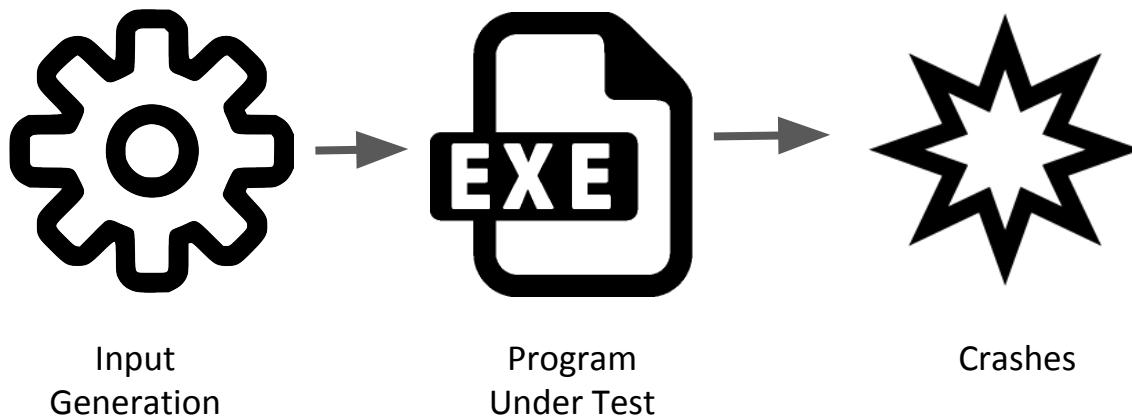


Fuzzing Binaries for Memory Safety Errors with QASan

Andrea Fioraldi, Daniele Cono D'Elia, Leonardo Querzoni

Fuzz Testing or Fuzzing

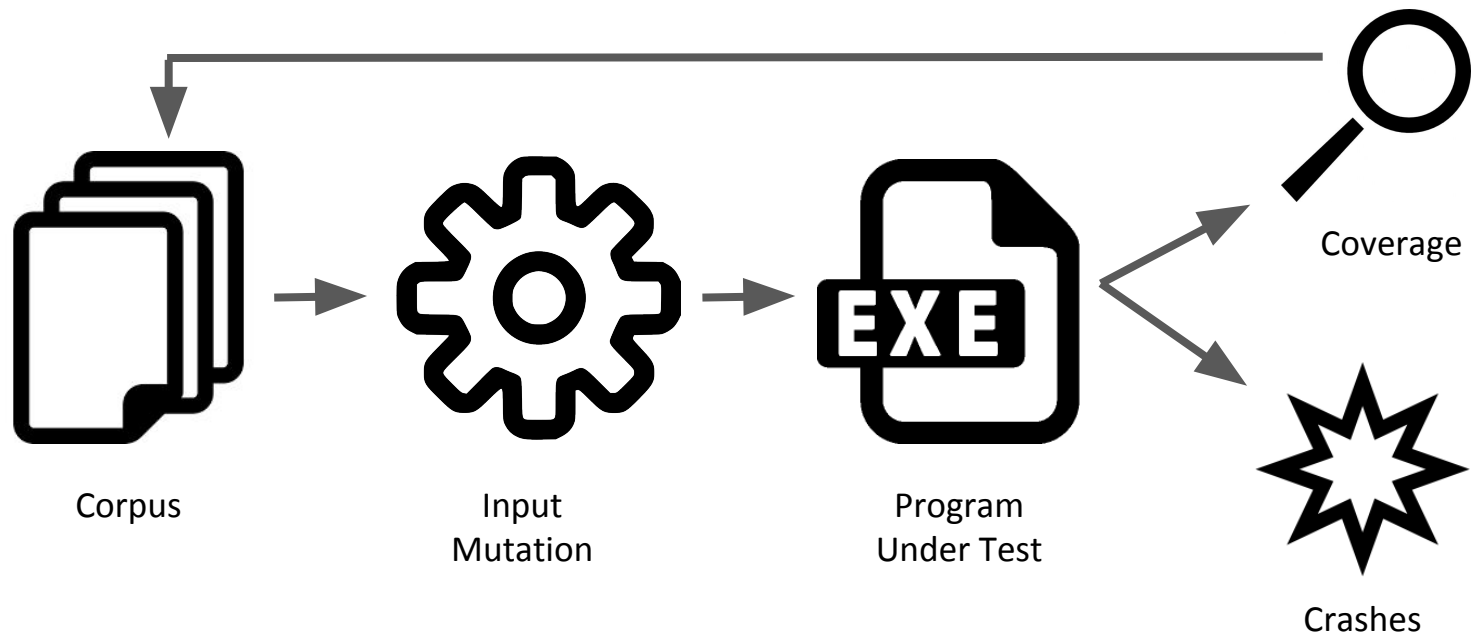
- A very effective random testing technique that discovered thousands of bugs



Challenges

- Trigger as many faults as possible in a given time window

Coverage-guided Fuzzing



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
- ...



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)

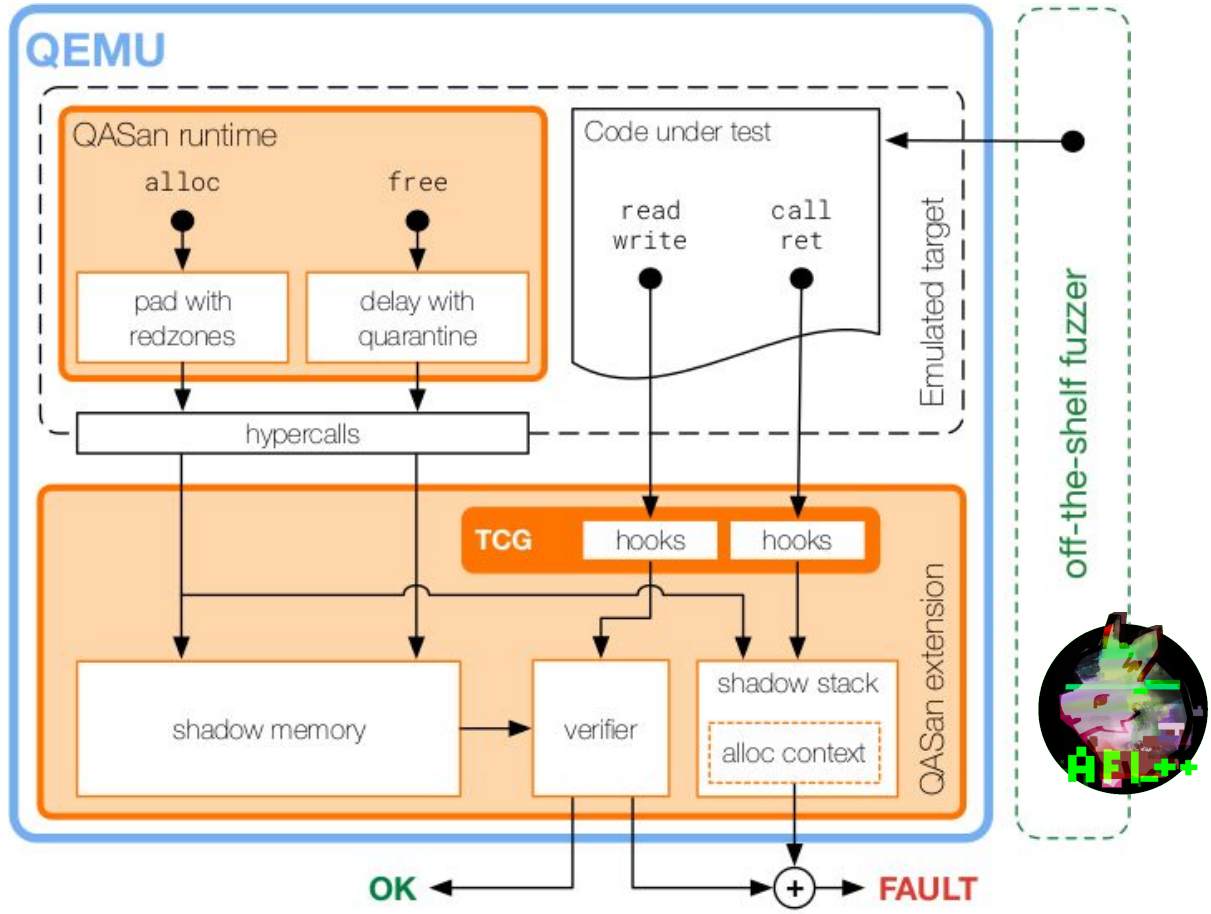
Fuzzing with (AFL++) QEMU

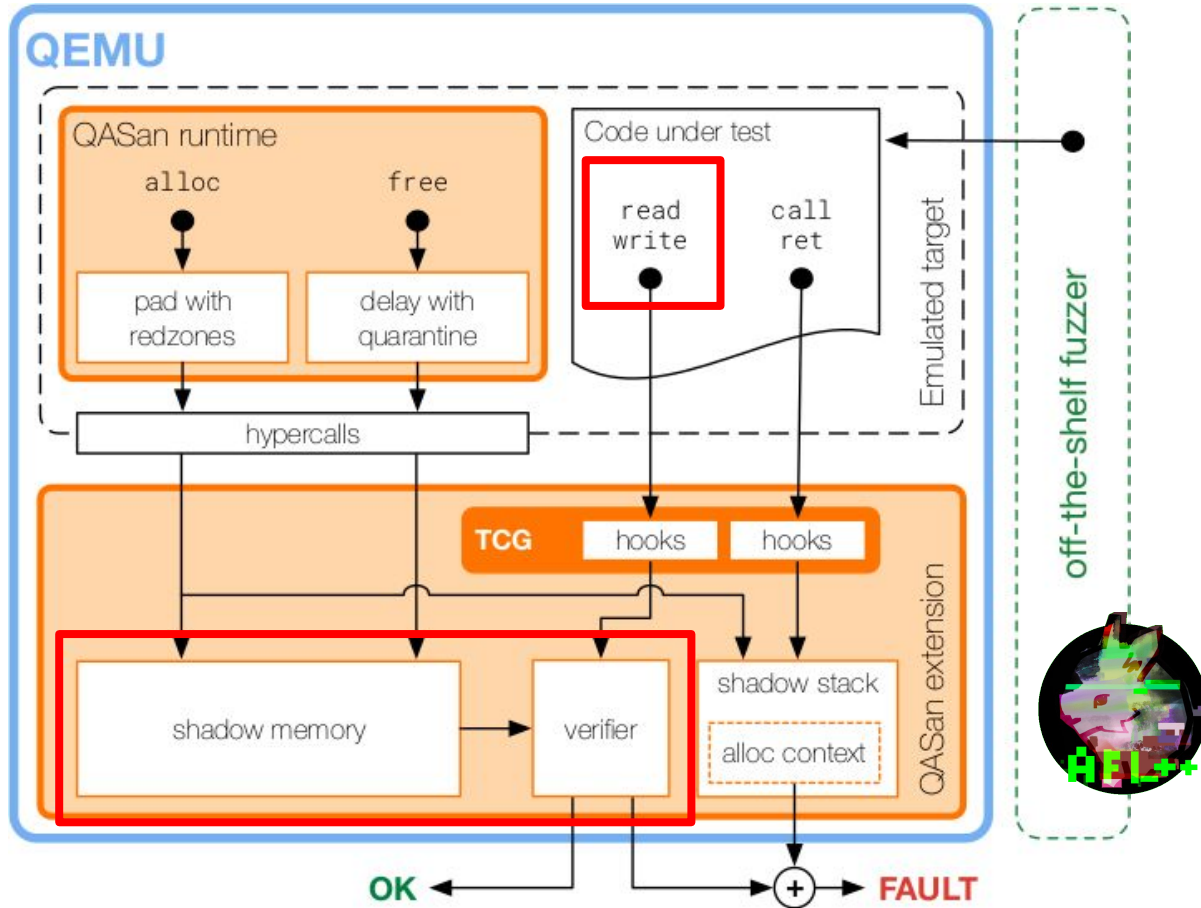
```
american fuzzy lop ++2.67d (tcpdump) [explore] {0}
-----
process timing
  run time      : 0 days, 0 hrs, 0 min, 8 sec
  last new path : 0 days, 0 hrs, 0 min, 0 sec
  last uniq crash : none seen yet
  last uniq hang : none seen yet
-----
cycle progress
  now processing : 18.0 (10.2%)
  paths timed out : 0 (0.00%)
-----
stage progress
  now trying : splice 10
  stage execs : 31/32 (96.88%)
  total execs : 15.3k
  exec speed  : 1731/sec
-----
fuzzing strategy yields
  bit flips      : n/a, n/a, n/a
  byte flips     : n/a, n/a, n/a
  arithmetics    : n/a, n/a, n/a
  known ints     : n/a, n/a, n/a
  dictionary     : n/a, n/a, n/a
  havoc/splice  : 168/9536, 7/2912
  py/custom     : 0/0, 0/0
  trim          : 0.00%/1378, n/a
-----
overall results
  cycles done : 0
  total paths : 176
  uniq crashes : 0
  uniq hangs  : 0
-----
map coverage
  map density : 0.34% / 1.19%
  count coverage : 2.22 bits/tuple
-----
findings in depth
  favored paths : 75 (42.61%)
  new edges on : 88 (50.00%)
  total crashes : 0 (0 unique)
  total tmouts  : 0 (0 unique)
-----
path geometry
  levels : 3
  pending : 170
  pend fav : 70
  own finds : 175
  imported : n/a
  stability : 100.00%
-----
[cpu000: 37%]
```

Sanitize with QEMU?

- Sanitize libraries
- Fast instrumentation with DBT
- Shadow memory outside the guest
- Immediate setup
- Cannot sanitize stack and globals when binary-only :(

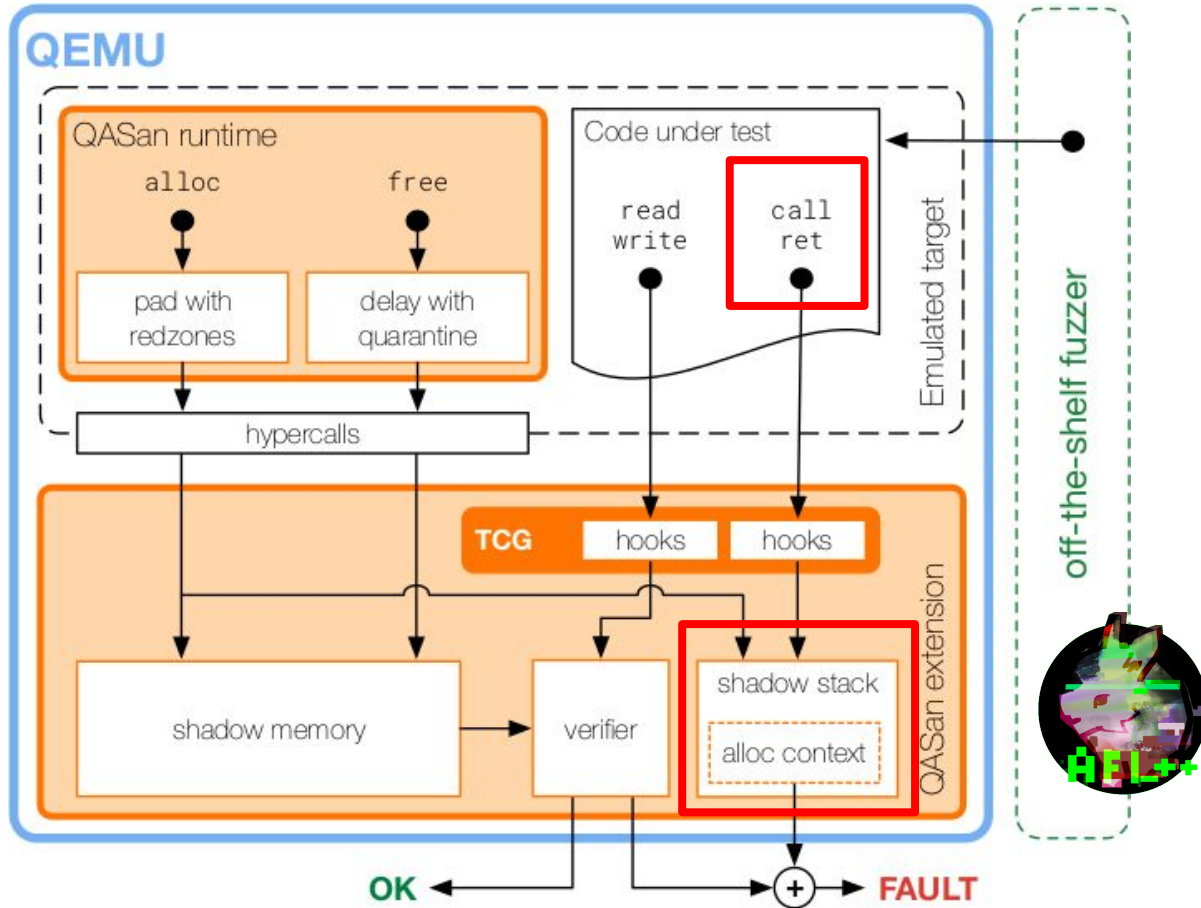
AddressSanitizer + QEMU = QASan



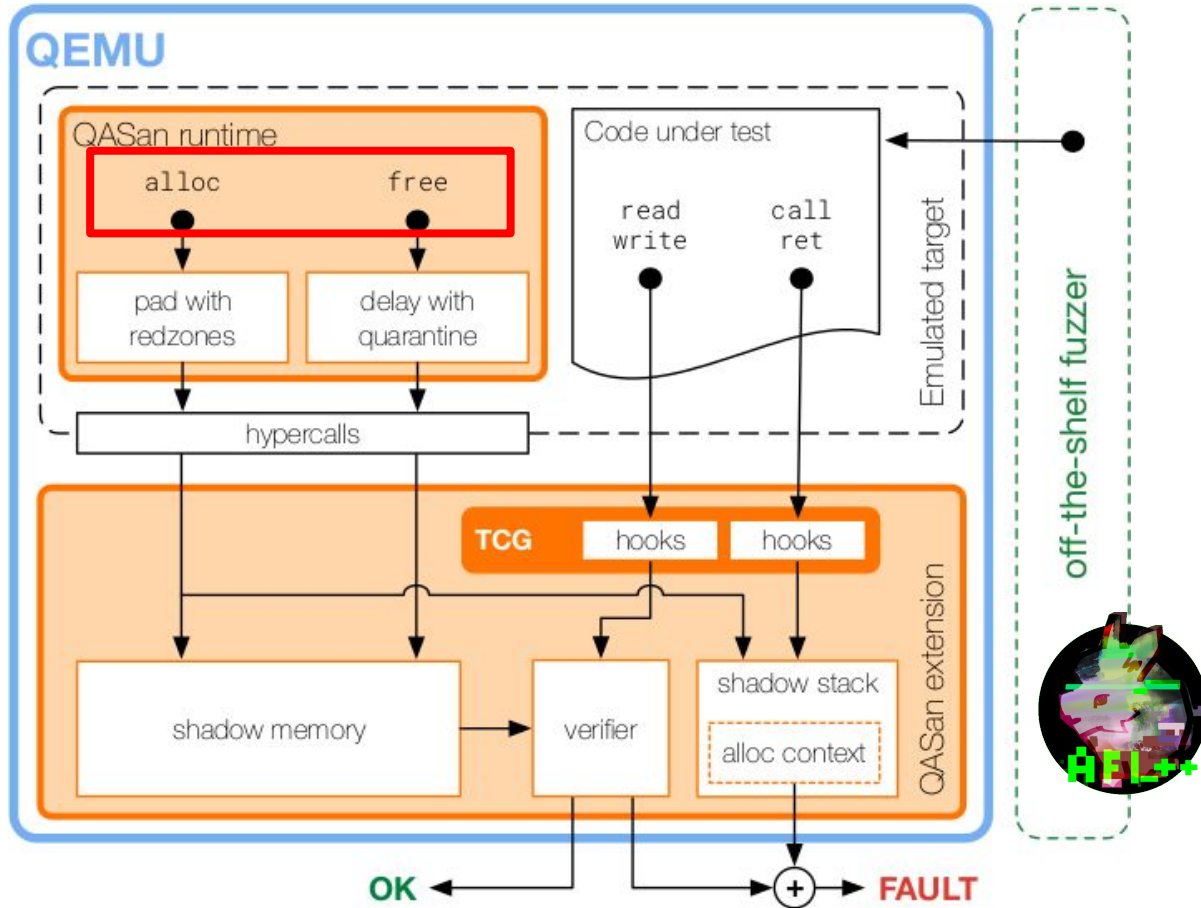


Instrument memory accesses to check for violations





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

```
# 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

```
# 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

```
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%)

	QASan TP	QASan FN	ASan TP	ASan FN	Memcheck TP	Memcheck FN
Heap-based Buffer Overflow	47.88	2.12	47.17	2.83	47.88	2.12
Double-Free	50.0	0.0	50.0	0.0	50.0	0.0
Use-After-Free	50.0	0.0	50.0	0.0	50.0	0.0
Freeing non-Heap Memory	49.98	0.0	50.0	0.0	50.0	0.0

More bugs, limited overhead

Program	Reported bugs		Executions per second (avg)		
	standard	QASan	standard	QASan	overhead
c-ares	0	1	859	618	1.39x
guetzli	1	1	642	426	1.51x
json	1	2	662	472	1.40x
libxml2	0	2	441	350	1.26x
openssl	0	1	118	43	2.74x
pcre2	16	29	613	457	1.34x
re2	0	0	653	448	1.46x
woff2	0	0	550	246	2.24x


```
==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/parser.c:10772
#2 0x000000429fe4 in xmlDoRead /home/andrea/Desktop/libxml2/parser.c:15299
#3 0x000000406945 in parseAndPrintFile /home/andrea/Desktop/libxml2/xmllint.c:?
#4 0x000000404315 in main /home/andrea/Desktop/libxml2/xmllint.c:3762
#5 0x7f977ca92b97 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 __libqasan_malloc (/home/andrea/qasan/libqasan.so+0x457b)
#1 0x7f977d849ae2 in malloc (/home/andrea/qasan/libqasan.so+0x2ae2)
#2 0x000000490bbc in xmlBufCreate /home/andrea/Desktop/libxml2/buf.c:136
#3 0x0000004101a0 in xmlSwitchInputEncodingInt /home/andrea/Desktop/libxml2/parserInternals.c:1196
#4 0x00000041024a in xmlSwitchToEncodingInt /home/andrea/Desktop/libxml2/parserInternals.c:1281
#5 0x00000041e643 in xmlParseEncodingDecl /home/andrea/Desktop/libxml2/parser.c:?
#6 0x000000424470 in xmlParseXMLDecl /home/andrea/Desktop/libxml2/parser.c:10631
#7 0x0000004247b1 in xmlParseDocument /home/andrea/Desktop/libxml2/parser.c:10772
#8 0x000000429fe4 in xmlDoRead /home/andrea/Desktop/libxml2/parser.c:15299
#9 0x000000406945 in parseAndPrintFile /home/andrea/Desktop/libxml2/xmllint.c:?
#10 0x000000404315 in main /home/andrea/Desktop/libxml2/xmllint.c:3762
#11 0x7f977ca92b97 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/pars
er.c:10666
```

```
Shadow bytes around the buggy address:
```

```
0x0000800e8670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000800e8680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000800e8690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
#7 0x0000004247b1 in xmlParseDocument /home/andrea/Desktop/libxml2/parser.c:10772
#8 0x000000429fe4 in xmlDoRead /home/andrea/Desktop/libxml2/parser.c:15299
#9 0x000000406945 in parseAndPrintFile /home/andrea/Desktop/libxml2/xmllint.c:?
#10 0x000000404315 in main /home/andrea/Desktop/libxml2/xmllint.c:3762
#11 0x7f977ca92b97 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:

```
0x0000800e8670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000800e8680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000800e8690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000800e86a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x0000800e86b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0000800e86c0: 00 00 00 00 00 00 00 00 00 00 00 00[fb]fb fb fb
0x0000800e86d0: fb fb fb fb fb fb fb fb fb fb fb fb 00 00 00
0x0000800e86e0: 00 00 fa fa fa fa fa fa fa fa fa fa fa fa fa
0x0000800e86f0: fa fa fd fd fd fd fd fd fd fd fd fd fd fd fd
0x0000800e8700: fd fd fd fd fd fd fb fb fb fb fb fb fb fb fb
0x0000800e8710: fb fb fb fb fb 00 00 00 00 00 00 00 fa fa fa fa
```

Shadow byte legend (one shadow byte represents 8 application bytes):

```
Addressable: 00
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone: fa
Heap right redzone: fb
Freed heap region: fd
Poisoned by user: f7
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/andreaforaldi/qasan>

The screenshot shows the GitHub repository page for `andreaforaldi/qasan`. The repository has 7 watchers, 175 stars, and 16 forks. The main navigation bar includes links for Code, Issues (5), Pull requests, Actions, Projects, Security, and Insights. The repository is currently on the `master` branch, with 4 other branches and 0 tags. A commit history table is visible, showing the most recent commit by `andreaforaldi` updating `TODO.md` 6 days ago. Below the commit history, a list of files and folders is shown, including `afI`, `asan-giovese @ b844043`, `include`, `libqasan`, and `qemu`. On the right side, the 'About' section describes QASan as a custom QEMU 3.1.1 that detects memory errors in the guest using AddressSanitizer. There are also links to `andreaforaldi.github.io/as...` and tags for `fuzzing` and `sanitization`.

File/Folder	Description	Time
afI	starting experimenting in full system	7 months ago
asan-giovese @ b844043	badfree using interval tree	3 months ago
include	hotfix backdoor regression for x86	2 months ago
libqasan	arm stacktraces	16 days ago
qemu	arm stacktraces	16 days ago

Thank You!

<https://github.com/andreafioraldi/qasan>

The screenshot shows the GitHub repository page for `andreafioraldi/qasan`. The repository is highlighted with a red box, showing 7 watches, 175 stars, and 16 forks. The repository content shows a commit history with files like `afl`, `asan-giovese`, `include`, `libqasan`, and `qemu`.

File	Description	Time
<code>afl</code>	starting experimenting in full system	7 months ago
<code>asan-giovese @ b844043</code>	badfree using interval tree	3 months ago
<code>include</code>	hotfix backdoor regression for x86	2 months ago
<code>libqasan</code>	arm stacktraces	16 days ago
<code>qemu</code>	arm stacktraces	16 days ago