IEEE COMPUTER SOCIETY

Fuzzing Binaries for Memory Safety Errors with QASan

Andrea Fioraldi, Daniele Cono D'Elia, Leonardo Querzoni

#IEEESecDev https://secdev.ieee.org/2020

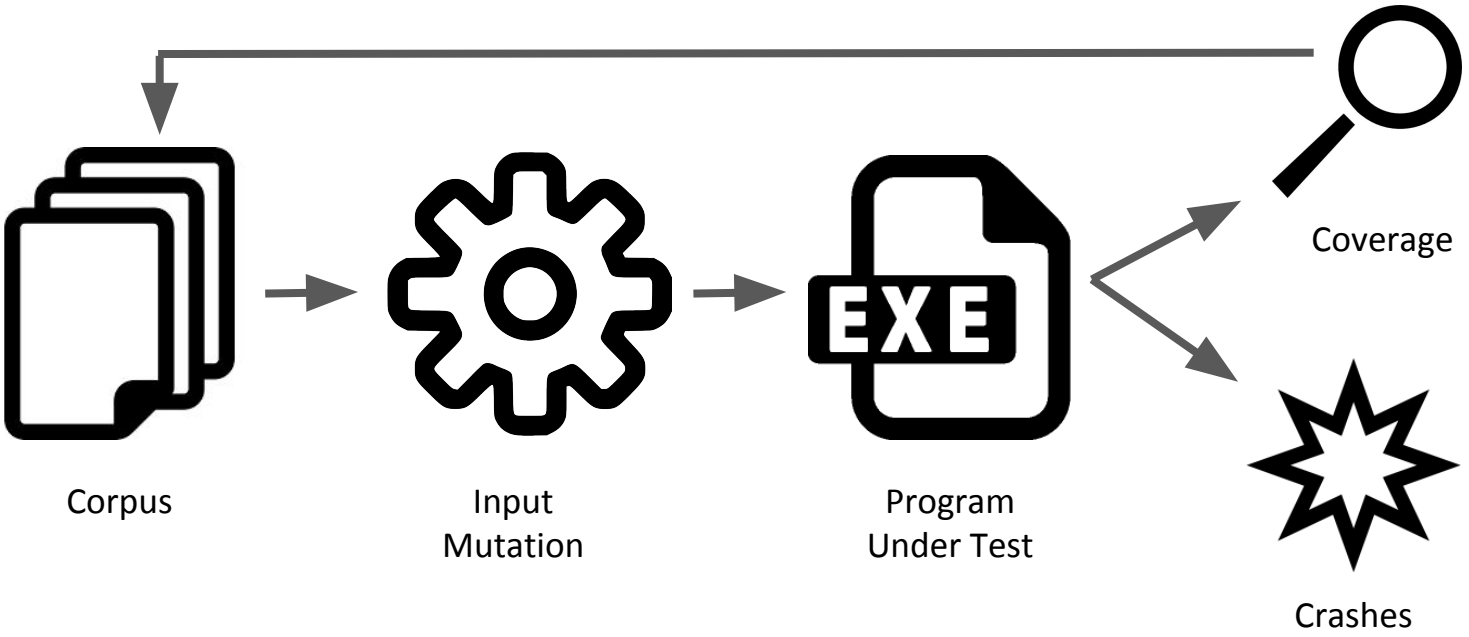
Fuzz Testing or Fuzzing

- 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?

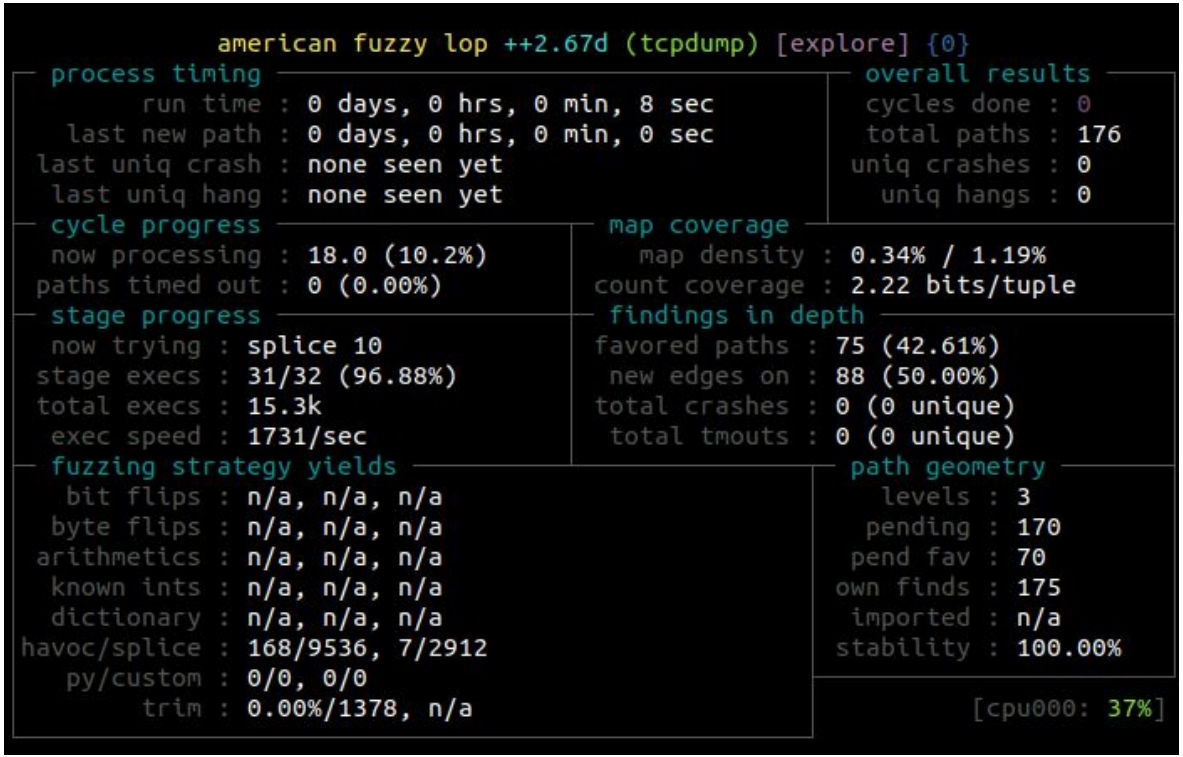
- Get coverage with
  - Dynamic Binary Translation (QEMU, Intel PIN, DynamoRIO, ...)
  - Hardware support (Intel PT)
  - Static Rewriting (DynInst, e9patch, RetroWrite (x86_64 only), …)

- 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)

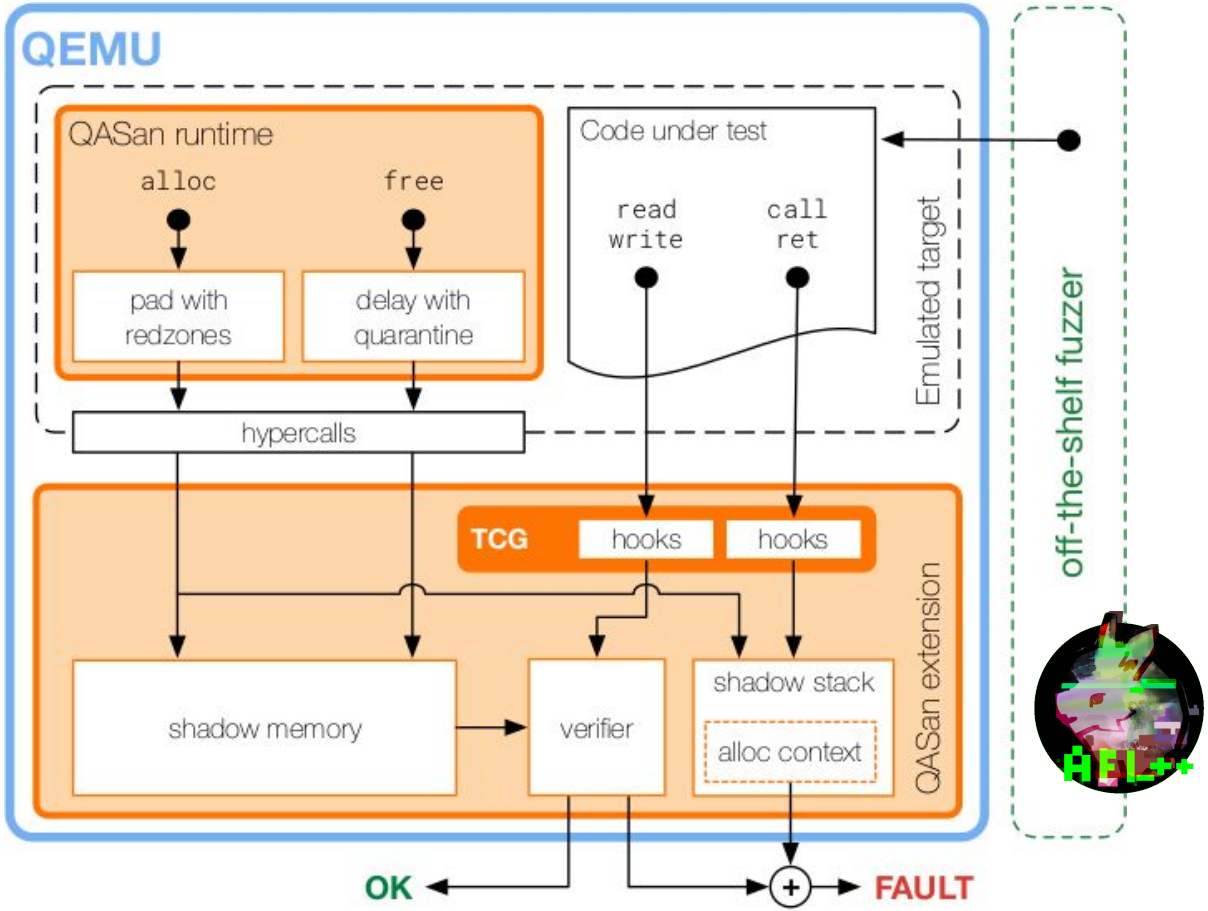
Fuzzing with (AFL++) QEMU

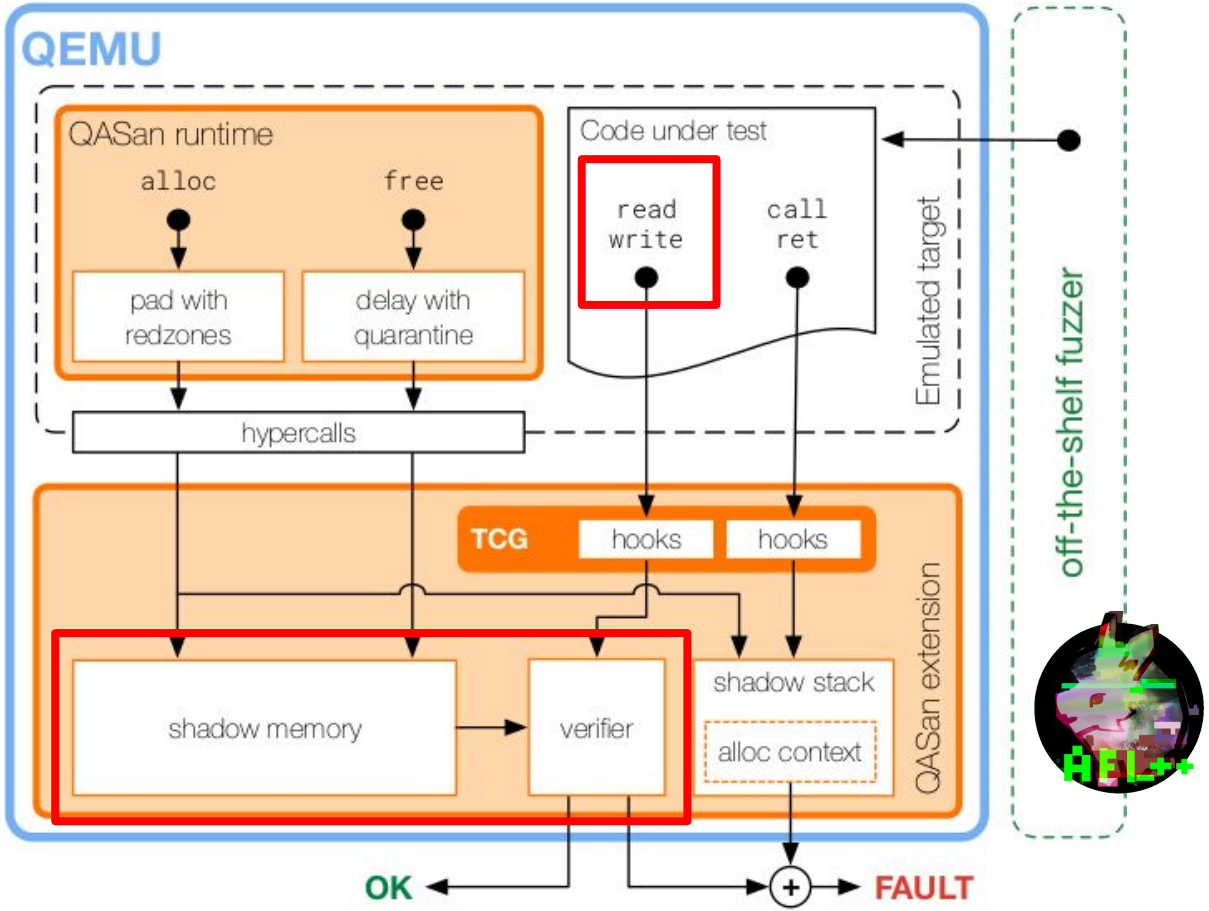


Sanitize with QEMU?

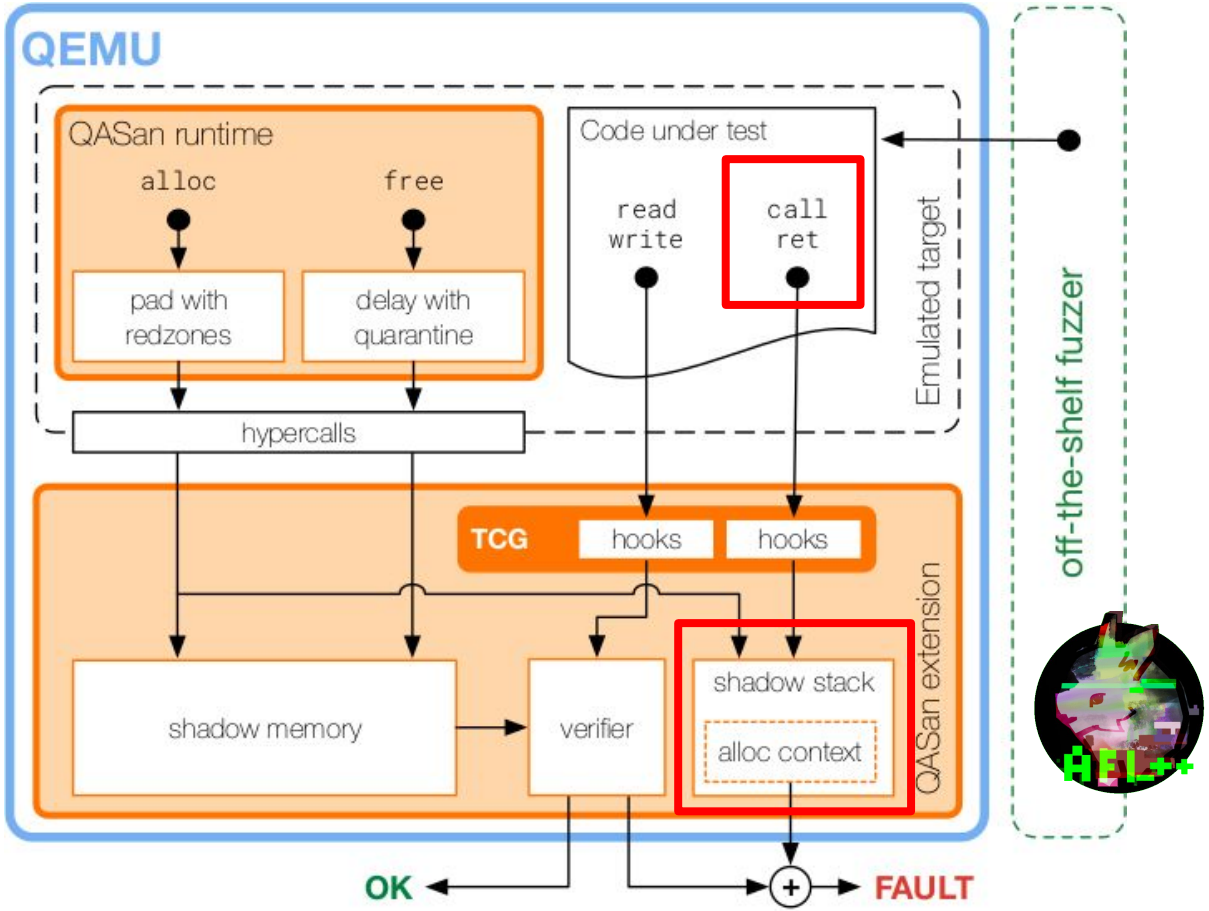
- Sanitize libraries
- Fast instrumentation with DBT
- Shadow memory outside the guest
- Immediate setup
- Cannot sanitize stack and globals when binary-only :(

AddressSanitizer + QEMU = QASan

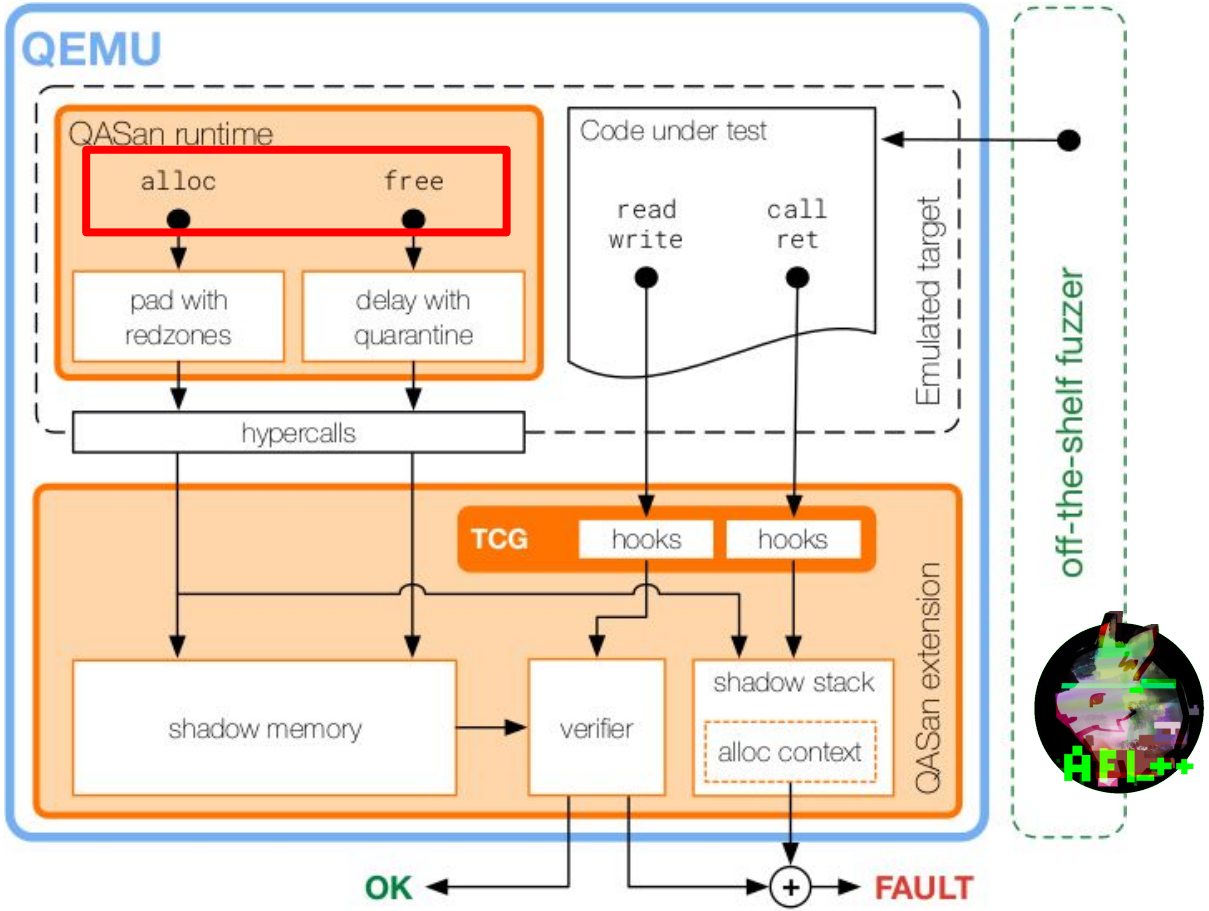

QEMU
QASan runtime
alloc
free
pad with redzones
delay with quarantine
Code under test
read
write
call
ret
Emulated target
hypercalls
TCG
hooks
hooks
shadow memory
verifier
shadow stack
alloc context
QASan extension
OK
FAULT
off-the-shelf fuzzer
AFL++



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
```

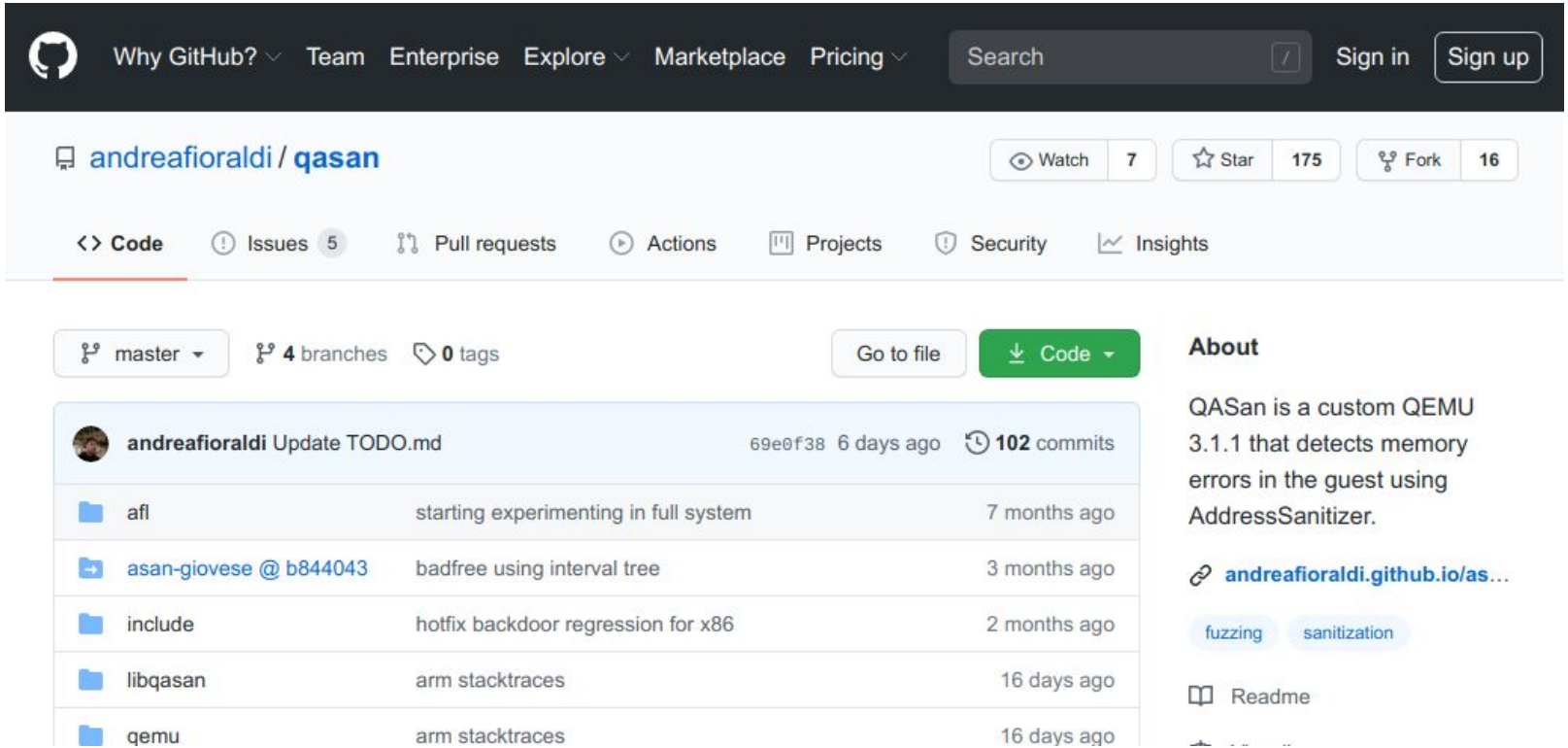
```
    #7 0x0000004247b1 in xmlParseDocument /home/andrea/Desktop/libxml2/parser.c:10772
    #8 0x000000429fe4 in xmlDoRead /home/andrea/Desktop/libxml2/parser.c:15299
    #9 0x000000406945 in parseAndPrintFile /home/andrea/Desktop/libxml2/xmllint.c:?
    #10 0x000000404315 in main /home/andrea/Desktop/libxml2/xmllint.c:3762
    #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/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
  0x0000800e86a0: 00 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 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 00
  0x0000800e86e0: 00 00 fa 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 fd
  0x0000800e8700: fd fd fd fd fd fd fb 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/andreafioraldi/qasan



Thank You!

https://github.com/andreafioraldi/qasan