THE FUZZING PROJECT

Can we run C with fewer bugs?

Hanno Böck

https://hboeck.de/

WHO AM I?

Hanno Böck

Freelance journalist (Golem.de, Zeit Online, taz, LWN)

Started Fuzzing Project November 2014

Since May 2015: Supported by Linux Foundation's Core Infrastructure Initiative

FUZZING?

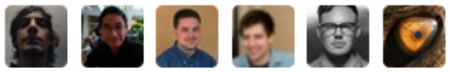
Throw garbage at software

lcamtuf
@lcamtuf

Following

Quick quiz: would you ever run strings on an untrusted file?

RETWEETS	FAVORITES
42	24



4:59 PM - 20 Oct 2014

FUZZING BINUTILS

Hundreds of bugs

THE C PROBLEM

C/C++ responsible for many common bug classes (Buffer overflows, use after free etc.)

Replacing C is good, but we'll have to live with it for a while

Mitigation: Good, but incomplete.

THE PAST

Dumb fuzzing: Only finds the easy bugs

Template-based fuzzing: a lot of work for each target

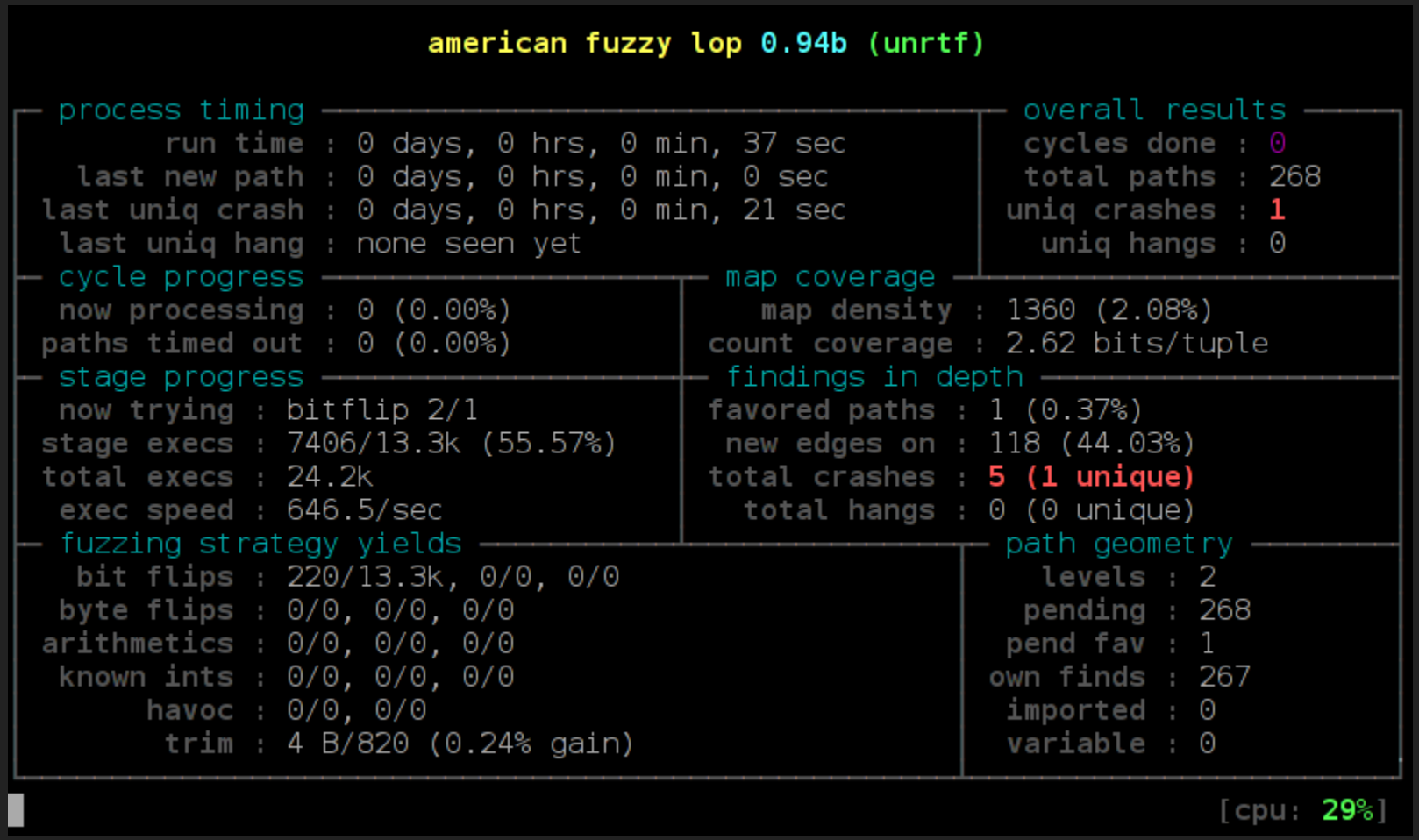
AMERICAN FUZZY LOP

AMERICAN FUZZY LOP (AFL)

Smart fuzzing, quick and easy

Code instrumentation

Watches for new code paths


american fuzzy lop 0.94b (unrtf)
process timing
run time : 0 days, 0 hrs, 0 min, 37 sec
last new path : 0 days, 0 hrs, 0 min, 0 sec
last uniq crash : 0 days, 0 hrs, 0 min, 21 sec
last uniq hang : none seen yet
overall results
cycles done : 0
total paths : 268
uniq crashes : 1
uniq hangs : 0
cycle progress
now processing : 0 (0.00%)
paths timed out : 0 (0.00%)
map coverage
map density : 1360 (2.08%)
count coverage : 2.62 bits/tuple
stage progress
now trying : bitflip 2/1
stage execs : 7406/13.3k (55.57%)
total execs : 24.2k
exec speed : 646.5/sec
findings in depth
favored paths : 1 (0.37%)
new edges on : 118 (44.03%)
total crashes : 5 (1 unique)
total hangs : 0 (0 unique)
fuzzing strategy yields
bit flips : 220/13.3k, 0/0, 0/0
byte flips : 0/0, 0/0, 0/0
arithmetics : 0/0, 0/0, 0/0
known ints : 0/0, 0/0, 0/0
havoc : 0/0, 0/0
trim : 4 B/820 (0.24% gain)
path geometry
levels : 2
pending : 268
pend fav : 1
own finds : 267
imported : 0
variable : 0
[cpu: 29%]

AFL SUCCESS STORIES

Bash Shellshock variants (CVE-2014-{6277,6278})

Stagefright vulnerabilities (CVE-2015-{1538,3824,3827,3829,3864,3876,6602})

GnuPG (CVE-2015-{1606,1607,9087})

OpenSSH out-of-bounds in handshake

OpenSSL (CVE-2015-{0288,0289,1788,1789,1790,3193})

BIND remote crashes (CVE-2015-{5477,2015,5986})

NTPD remote crash (CVE-2015-7855)

Libreoffice GUI interaction crashes

FUZZING MATH

$0x0505\ 05050505^{2} \bmod 0x41\ 41414141\ 41414141\ 41412741\ 41414141\ 41414141\ 41414141\ 41414141\ 41414141\ 41414141\ 41414141\ 41414141\ 41414141\ 41414141\ 41414141\ 41418000\ 00000005$

$= 0x19324B\ 647D967D\ 644B3219\ ?$

$= 0x34\ 34343434\ 34343434\ 34341F67\ 67676767\ 67676767\ 67676767\ 67676767\ 67676767\ 67676767\ 67676767\ 67676767\ 67676767\ 67676767\ 67676774\ 74747474\ 74747474\ 74746F41\ 41414141\ 41417373\ 73737373\ 73737373\ 73737373\ 73737373\ 73737373\ 73737373\ 73737373\ 73737373\ 73737373\ 73737373\ 73737373\ 73737373\ 73738000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 00000000\ 0019324B\ 647D967D\ 644B321D\ ?$

16 crypto/bn/asm/x86_64-mont.pl View

```
@@ -1378,7 +1378,6 @@
1378 1378         lea     8*8($nptr),$nptr
1379 1379         xor     %rax,%rax
1380 1380         mov     8(%rsp),%rdx            # pull end of t[]
1381      -       xor     $carry,$carry
1382 1381         cmp     0(%rsp),$nptr           # end of n[]?
1383 1382         jae     .L8x_no_tail
1384 1383
@@ -1491,17 +1490,10 @@
1491 1490  .align 32
1492 1491  .L8x_tail_done:
1493 1492         add     (%rdx),%r8              # can this overflow?
1494      -       adc     \$0,%r9
1495      -       adc     \$0,%r10
1496      -       adc     \$0,%r11
1497      -       adc     \$0,%r12
1498      -       adc     \$0,%r13
1499      -       adc     \$0,%r14
1500      -       adc     \$0,%r15
1501      -       sbb     %rax,%rax
     1493 +       xor     %rax,%rax
1502 1494
1503      -.L8x_no_tail:
1504 1495         neg     $carry
     1496 +.L8x_no_tail:
1505 1497         adc     8*0($tptr),%r8
1506 1498         adc     8*1($tptr),%r9
1507 1499         adc     8*2($tptr),%r10
@@ -1510,9 +1502,7 @@
1510 1502         adc     8*5($tptr),%r13
1511 1503         adc     8*6($tptr),%r14
1512 1504         adc     8*7($tptr),%r15
1513      -       sbb     $carry,$carry
1514      -       neg     %rax
1515      -       sub     $carry,%rax             # top-most carry
     1505 +       adc     \$0,%rax                # top-most carry
1516 1506
1517 1507         mov     40(%rsp),$nptr          # restore $nptr
1518 1508
```

$0x0F\ FFFFFFFF\ FFFFFFFF^0 \bmod 1$

$= 0 \text{ or } 1 \ ?$

NETTLE ECC / NIST P256

point (0xFFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF 001C2C00, 0x9731275B 8E973CEA FD8ABF5A 6E16A177 F05A3451 14FBC752 7B3A60BC 65FE606A) * 1 !=
point (0xFFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFE FFFFFFFF 001C2C00 , 0x9731275B 8E973CEA FD8ABF5A 6E16A177 F05A3451 14FBC752 7B3A60BC 65FE606A)

ADDRESS SANITIZER (ASAN)

If you only take away one thing from this talk:

Use Address Sanitizer!

-fsanitize=address in gcc/clang

SPOT THE BUG!

```
int main() {
        int a[2] = {1, 0};
        printf("%i", a[2]);
}
```

```
=================================================================
==577==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffe64bfb498 at pc 0x400a06 bp 0x7ffe64bfb460 sp 0x7ffe64bfb450
READ of size 4 at 0x7ffe64bfb498 thread T0
    #0 0x400a05 in main /tmp/test.c:3
    #1 0x7f701400262f in __libc_start_main (/lib64/libc.so.6+0x2062f)
    #2 0x400878 in _start (/tmp/a.out+0x400878)

Address 0x7ffe64bfb498 is located in stack of thread T0 at offset 40 in frame
    #0 0x400955 in main /tmp/test.c:1

  This frame has 1 object(s):
    [32, 40) 'a' <== Memory access at offset 40 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /tmp/test.c:3 main
Shadow bytes around the buggy address:
  0x10004c977640: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004c977650: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004c977660: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004c977670: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004c977680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1
=>0x10004c977690: f1 f1 00[f4]f4 f4 00 00 00 00 00 00 00 00 00 00
  0x10004c9776a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004c9776b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004c9776c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004c9776d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004c9776e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Contiguous container OOB:fc
  ASan internal:           fe
==577==ABORTING
```

ADDRESS SANITIZER HELPS

Finds lots of hidden memory access bugs like out of bounds read/write (Stack, Heap, Global), use-after-free etc.

FINDING HEARTBLEED WITH AFL+ASAN

Small OpenSSL handshake wrapper

AFL finds Heartbleed within 6 hours

LibFuzzer needs just 5 Minutes

ADDRESS SANITIZER

If ASAN catches all these typical C bugs...

... can we just use it in production?

ASAN IN PRODUCTION

Yes, but not for free

50 - 100 % CPU and memory overhead

Example: Hardened Tor Browser

GENTOO LINUX WITH ASAN

Everything compiled with ASAN except a few core packages (gcc, glibc, dependencies)

FIXING PACKAGES

Memory access bugs in normal operation.

These need to be fixed.

bash, shred, python, syslog-ng, nasm, screen, monit, nano, dovecot, courier, proftpd, claws-mail, hexchat, ...

PROBLEMS / CHALLENGES

ASAN executable + non-ASAN library: fine

ASAN library + non-ASAN executable: breaks

Build system issues (mostly libtool)

Custom memory management (boehm-gc, jemalloc, tcmalloc)

IT WORKS

Running server with real webpages.

But: More bugs need to be fixed.

OTHER TOOLS

KASAN AND SYZCKALLER

KASAN: ASAN for the Linux Kernel.

syzkaller: syscall fuzzing similar to afl

UNDEFINED BEHAVIOR SANITIZER (UBSAN)

Finds code that is undefined in C

Invalid shifts, int overflows, unaligned memory access, ...

Problem: Just too many bugs, problems rare

There's also TSAN (Thread sanitizer, race conditions) and MSAN (Memory Sanitizer, uninitialized memory)

AFL AND NETWORKING

Fuzzing network connections, experimental code by Doug Birdwell

Usually a bit more brittle than file fuzzing

Not widely used yet

AFL AND ANDROID

Implementation from Intel just released

Promising (Stagefright)

Android Security desperately needs it

WHAT HAS THIS TO DO WITH FREE SOFTWARE?

Remember the many eyes principle?

"Free software is secure - because everyone can look at the source and find the bugs."

We have to actually *do* that.

QUESTION TO THE AUDIENCE

Do you develop / maintain software? In C?

Do you know / use Fuzzing and Address Sanitizer?

If not: Why not?

THANKS FOR LISTENING

Use Address Sanitizer!

Fuzz your software.

Questions?

https://fuzzing-project.org/