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 2015

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

lcamtuf
@lcamtuf

Following

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

RETWEETS 42 FAVORITES 24

4:59 PM - 20 Oct 2014

FUZZING BINUTILS

Hundreds of bugs

WHAT IS FUZZING?

Test software with random malformed input

THE PAST

Dumb fuzzing: Only finds the easy bugs

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

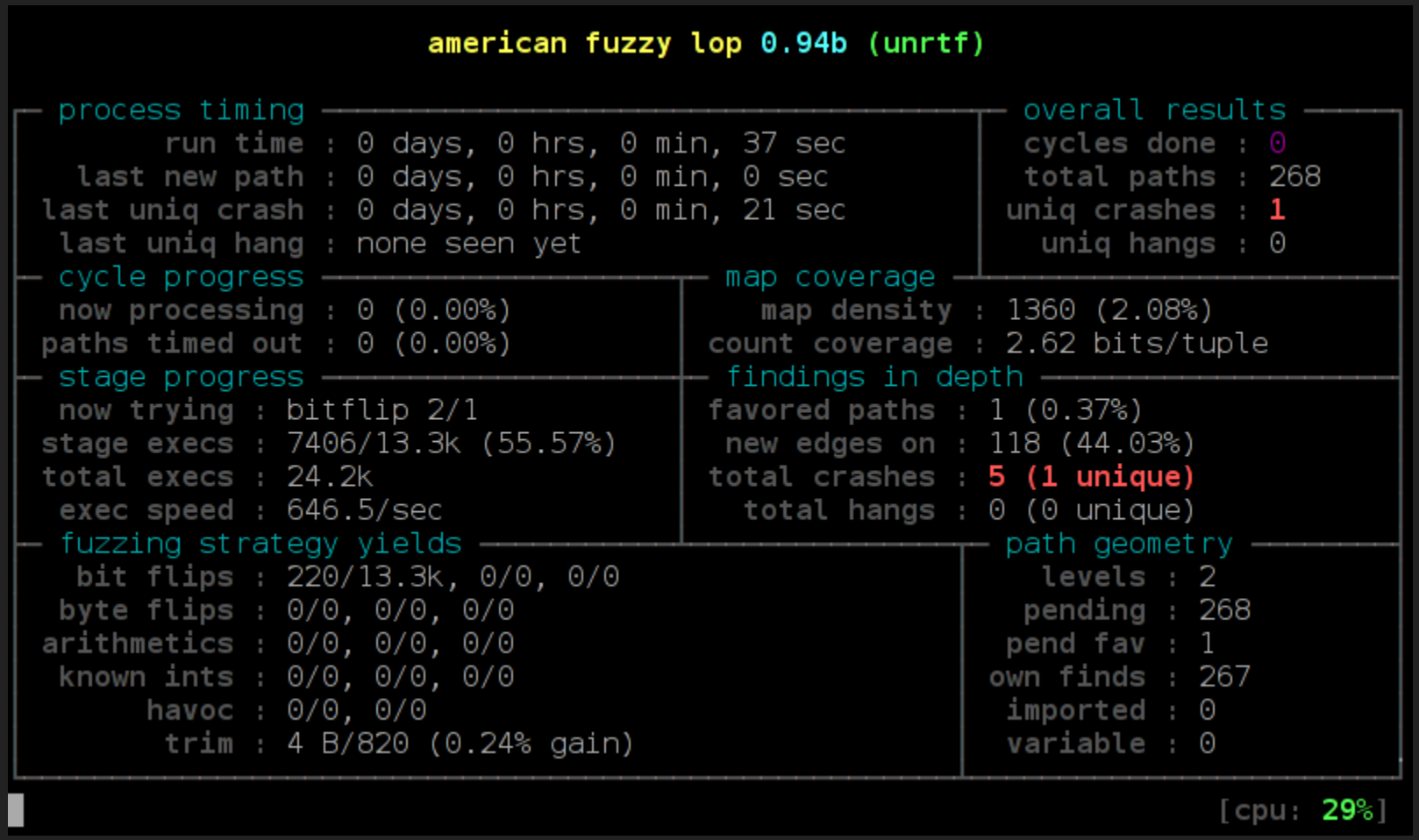
AMERICAN FUZZY LOP

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})

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

NTPD remote crash (CVE-2015-7855)

Libreoffice GUI interaction crashes

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

BN_SQR BUG (CVE-2014-3570)

Wrong calculation in one out of 2^128 cases

No way to find this with random testing

AFL can find it (credit: Ralph-Philipp Weinmann)

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.

KASAN

ASAN for the Linux Kernel.

Userspace and Kernel ASAN independent of each other.

Found a bug in my GPU driver just by booting with KASAN.

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

C - REPLACE, MITIGATE, FIX

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

GET RID OF C

Safer programming languages

Go and Rust new rising stars

Some interesting projects: Servo (browser engine), MirageOS

MITIGATION

Old: noexec pages, ASLR, stack canaries

Most Linux distros don't enable proper ASLR (-fpic/-pie)

New: Safe Stack, Code flow integrity (clang, Chrome is testing this), RAP

THANKS FOR LISTENING

Use Address Sanitizer!

Fuzz your software.

Questions?

https://fuzzing-project.org/