Sanitize, Fuzz, and Harden Your C++ Code

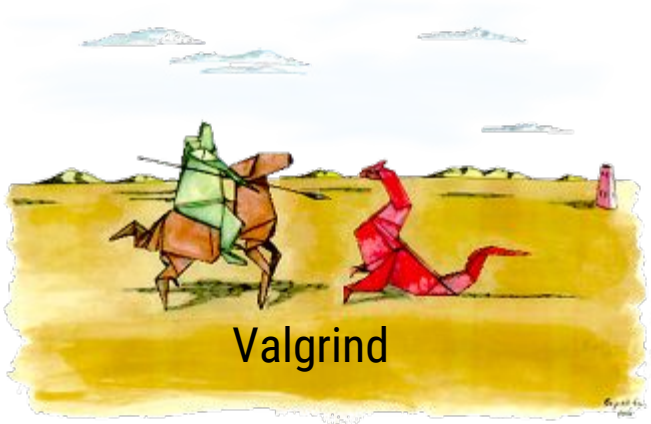
Kostya Serebryany (kcc@google.com)

Agenda

1. Sanitize your code.
2. Fuzz your code.
3. Harden your code.

Sanitize your code,

~2007

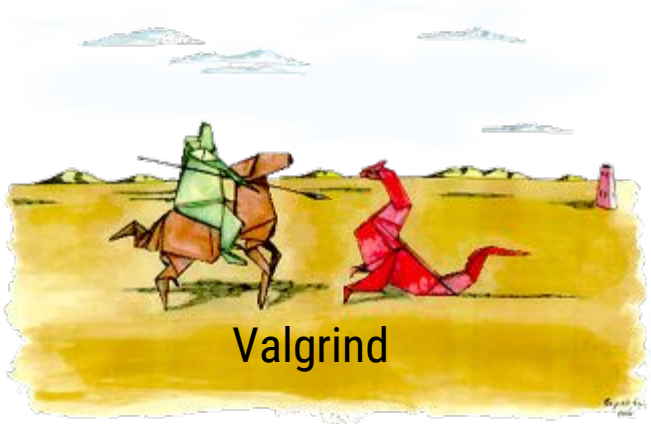


Based on **binary** instrumentation.

vs.

10,000,000+ lines
C++ code

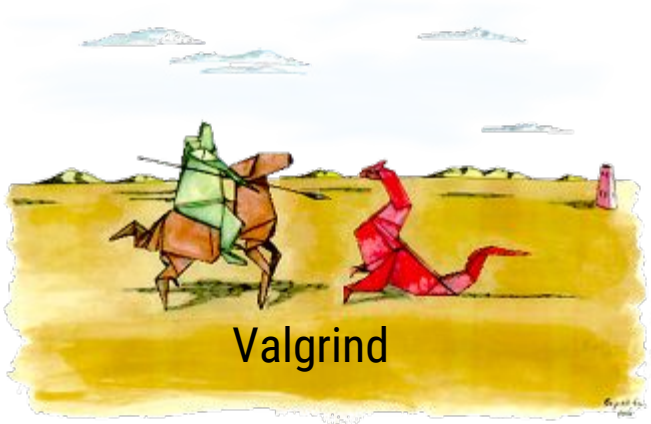
~2007



Based on **binary** instrumentation.

100s bugs found!

~2007



Based on **binary** instrumentation.

100s bugs found!

Large CPU overhead

20x slow down

~2011

Sanitizers
- ASan
- TSan
- MSan
- UBSan

vs.

100,000,000+ lines
C++ code

New tools, based on **compiler** instrumentation.
Available in LLVM and GCC (both open-source)

~2011

Sanitizers

- ASan: Address Sanitizer detects use-after-free, buffer-overflow, and leaks.
- TSAN: Thread Sanitizer detects data races, deadlocks.
- MSAN: Memory Sanitizer detects uses of uninitialized memory.
- UBSan: Undefined Behavior Sanitizer detects… that.

New tools, based on **compiler** instrumentation.
Available in LLVM and GCC (both open-source)

```
int main(int argc, char **argv) {
   int *array = new int[100];
   delete [] array;
   return array[argc]; } // BOOM
```

```
% clang++ -O1 -fsanitize=address a.cc && ./a.out
==30226== ERROR: AddressSanitizer heap-use-after-free
READ of size 4 at 0x7faa07fce084 thread T0
   #0 0x40433c in main a.cc:4
0x7faa07fce084 is located 4 bytes inside of 400-byte region
freed by thread T0 here:
   #0 0x4058fd in operator delete[](void*) _asan_rtl_
   #1 0x404303 in main a.cc:3
previously allocated by thread T0 here:
   #0 0x405579 in operator new[](unsigned long) _asan_rtl_
   #1 0x4042f3 in main a.cc:2
```

Sanitizers are *fast*!

- Valgrind introduced 20x slowdown; *San is ~2x slowdown
- We were able to use Sanitizers on 99% of unit tests
- Running all Chromium unit tests on nearly every new code commit, so bugs can be reverted quickly.
- Found hundreds of bugs, old ones and regressions ([1], [2], [3])

But...

Test coverage is never perfect.
Bugs still creep into production.

Fuzz your code,

Fuzzing (or Fuzz Testing)

- Generate a huge number of test inputs
  - increase code coverage and make the code misbehave

- At Google:
  - Several teams, several frameworks, hundreds of fuzzers
  - 5000+ CPU cores doing fuzz testing 24/7
  - 10x more bugs found compared to unit tests
  - 5000+ bugs in Chromium, 1200+ bugs in ffmpeg

libFuzzer: guided fuzzing for APIs

- Start with some test corpus (may be empty)
- Provide your own target function:
  - **(const uint8_t *Data, size_t Size)**
- Build it with special compiler instrumentation (LLVM)
  - Add one of the sanitizers for better results
- Run on many CPUs
  - The test corpus will grow
  - Bugs will be reported, reproducers will be recorded

FreeType Example

```
int LLVMFuzzerTestOneInput(const uint8_t * data, size_t size) {
  FT_Face face;
  if (size < 1) return 0;
  if (!FT_New_Memory_Face(library, data, size, 0, &face)) {
    FT_Done_Face(face);
  }
  return 0;
}
```

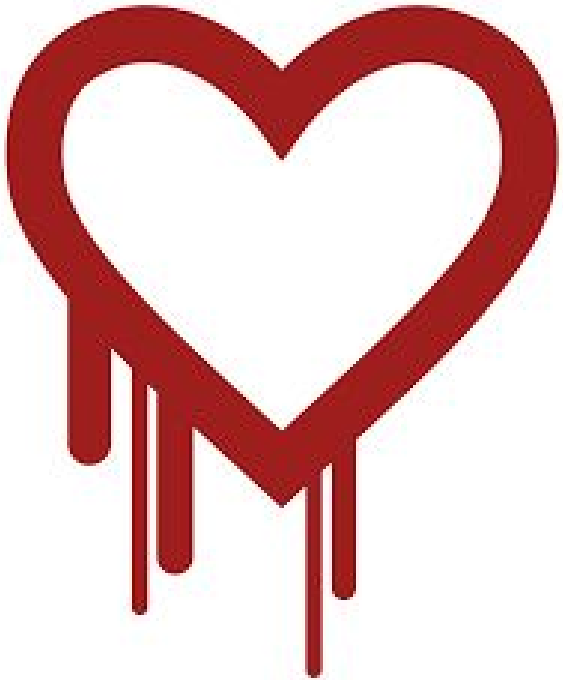
#45999 **left shift of negative value -4592**
#45989 **leak** in t42_parse_charstrings
#45987 512 byte input **consumes 1.7Gb** / 2 sec to process
#45986 **leak** in ps_parser_load_field
#45985 **signed integer overflow**: -35475362522895417 * -8256 cannot be represented in t
#45984 **signed integer overflow**: 2 * 1279919630 cannot be represented in type 'int
#45983 runtime error: **left shift of negative value -9616**
#45966 **leaks** in parse_encoding, parse_blend_design_map, t42_parse_encoding
#45965 left shift of 184 by 24 places cannot be represented in type 'int'
#45964 **signed integer overflow**: 6764195537992704 * 7200 cannot be represented in type
#45961 FT_New_Memory_Face consumes 6Gb+
#45955 **buffer overflow** in T1_Get_Private_Dict/strncmp
#45938 **shift exponent 2816 is too large** for 64-bit type 'FT_ULong'
#45937 **memory leak** in FT_New_Memory_Face/FT_Stream_OpenGzip
#45923 **buffer overflow** in T1_Get_Private_Dict while doing FT_New_Memory_Face
#45922 **buffer overflow** in skip_comment while doing FT_New_Memory_Face
#45920 FT_New_Memory_Face **takes infinite time** (in PS_Conv_Strtol)
#45919 FT_New_Memory_Face **consumes 17Gb** on a small input

...

OpenSSL Example

```
SSL_CTX *sctx = Init();

extern "C" int LLVMFuzzerTestOneInput(unsigned char * Data, size_t Size) {
  SSL *server = SSL_new(sctx);
  BIO *sinbio = BIO_new(BIO_s_mem());
  BIO *soutbio = BIO_new(BIO_s_mem());
  SSL_set_bio(server, sinbio, soutbio);
  SSL_set_accept_state(server);
  BIO_write(sinbio, Data, Size);
  SSL_do_handshake(server);
  SSL_free(server);
  return 0;
}
```



Exact commands: llvm.org/docs/LibFuzzer.html

Hey, I already know that HeartBleed was found with fuzzing!!!

But did you know how simple and fast it could be?

https://code.google.com/p/chromium/codesearch#chromium/src/testing/libfuzzer/fuzzers/libxml_xml_read_memor

Search Code

[chromium] // src / testing / libfuzzer / fuzzers / libxml_xml_read_memory_fuzzer.cc

Files | Outline

- brotli_fuzzer.cc
- courgette_fuzzer.cc
- dns_record_fuzzer.cc
- empty_fuzzer.cc
- es_parser_adts_fuzzer.cc
- es_parser_h264_fuzzer.cc
- es_parser_mpeg1audio_fuzzer.cc
- ftp_ctrl_response_fuzzer.cc
- ftp_directory_listing_fuzzer.cc
- gfx_png_image_fuzzer.cc
- http_chunked_decoder_fuzzer.cc
- icu_uregex_open_fuzzer.cc
- language_detection_fuzzer.cc
- libexif_parser_fuzzer.cc
- libpng_read_fuzzer.cc
- libxml_xml_read_memory_fuzzer.cc
- mp4_box_reader_fuzzer.cc
- pdfium_fuzzer.cc
- quic_crypto_framer_parse_message_fuzzer.cc
- snappy_fuzzer.cc
- sqlite3_prepare_v2_fuzzer.cc

libxml_xml_read_memory_fuzzer.cc Layers Find Goto Link View in Related files

```
1  // Copyright (c) 2015 The Chromium Authors. All rights reserved.
2  // Use of this source code is governed by a BSD-style license that can be
3  // found in the LICENSE file.
4
5  #include "libxml/parser.h"
6
7  void ignore (void * ctx, const char * msg, ...) {
8    // Error handler to avoid spam of error messages from libxml parser.
9  }
10
11 extern "C" int LLVMFuzzerTestOneInput(const unsigned char *data, size_t size) {
12   xmlSetGenericErrorFunc(NULL, &ignore);
13
14   if (auto doc = xmlReadMemory(reinterpret_cast<const char *>(data),
15                                size, "noname.xml", NULL, 0)) {
16     xmlFreeDoc(doc);
17   }
18
19   return 0;
20 }
```

Fuzzing is the next best thing (after unit testing).

But...

Some bugs still evade us!

Harden your code.

Threat #1: Buffer-overflow/use-after-free overwrites a function pointer (or vptr) by an attacker-controlled value.

Example: Hijacked VPTR in Chromium, Pwn2Own 2013 (CVE-2013-0912)

Control Flow Integrity (CFI)

```
void Bad() { puts("BOOO"); }
struct Expr {
  long a[2];
  long (*Op)(long *);
};
int main(int argc, char **argv) {
  Expr e;
  e.a[2 * argc] = (long)&Bad;
  e.Op(e.a);
}
```

```
% clang a.c && ./a.out
BOOO

% clang -flto -fsanitize=cfi a.c && ./a.out
Illegal instruction (core dumped)
```

Control Flow Integrity (CFI)

- Statically compute the exact set of allowed function pointers for every indirect (or virtual) call
  - Today, require LTO (link-time optimization)
- Insert a constant-time check before every call site.
  - At most one memory load per check

Threat #2: Stack-buffer-overflow overwrites return address by an attacker-controlled value.

Example: One of the “libstagefright” bugs (CVE-2015-3869)

SafeStack

```
void Bad() { puts("BOOO"); exit(0); }

int main(int argc, char **argv) {
  long array[10];
  array[argc * 13] = (long)&Bad;
}
```

```
% clang a.c && ./a.out
BOOO

% clang -fsanitize=safe-stack a.c && ./a.out
%
```

SafeStack

- For every thread create a “shadow stack”
  - a separate memory region the size of the original stack
- Place all local variables on the shadow stack
- If a buffer overflow happens, the return address can not be overwritten
- May combine with stack cookies

CFI & SafeStack Overhead

- 1-2% CPU (each)
- 3-8% code size (each)

Security is either in-depth or none...

Sanitize, Fuzz, and Harden Your C++ code.

Linux Kernel too!

GPF in keyring_destroy CVE-2015-7872
Uninterruptable hang in sendfile
Infinite loop in ip6_fragment
GPF in rt6_uncached_list_flush_dev
GPF in shm_lock
Use-after-free in ep_remove_wait_queue
Unkillable processes due to PTRACE_TRACEME
Paging fault with hard IRQs disabled in getsockopt
Resource leak in unshare
WARNING in task_participate_group_stop
lockdep warning in ip_mc_msfget
GPF in tcp_sk_init/icmp_sk_init
Use-after-free in unshare
Use-after-free in selinux_ip_postroute_compat
Use-after-free in ipv4_conntrack_defrag
Deadlock between bind and splice
Deadlock between setsockopt/getsockopt
WARNING in shmem_evict_inode
deadlock between tty_write and tty_send_xchar
tty,net: **use-after-free** in x25_asy_open_tty
deadlock during fuseblk shutdown
another uninterruptable hang in sendfile
GPF in add_key
yet another uninterruptable hang in sendfile
use-after-free in sctp_do_sm

WARNING in gsm_cleanup_mux
WARNING in handle_mm_fault
use-after-free in sock_wake_async
use-after-free in irtty_open
WARNING in tcp_recvmsg
use-after-free in tty_check_change
GPF in process_one_work (flush_to_ldisc)
Freeing active kobject in pps_device_destruct
gigaset: freeing an active object
use-after-free in ip6_setup_cork
user-controllable kmalloc size in sctp_getsockopt_local_addrs
net: **use after free** in ip6_make_skb
user-controllable kmalloc size in bpf syscall
deadlock in perf_ioctl
heap **out-of-bounds** access in array_map_update_elem
memory leak in do_ipv6_setsockopt
memory leak in alloc_huge_page
jump label: negative count!
signed integer overflow in ktime_add_safe
undefined shift in __bpf_prog_run
use-after-free in __perf_install_in_context
use-after-free in ip6_xmit
int overflow in io_getevents
WARNING in crypto_wait_for_test
use-after-free in inet6_destroy_sock
another **use-after-free** in sctp_do_sm

GPF in keyctl
use-after-free in pptp_connect
Information leak in pptp_bind
Information leak in llcp_sock_bind/llcp_raw_sock_bind
Information leak in sco_sock_bind
perf: stalls in perf_install_in_context
BUG_ON(!PageLocked(page)) in munlock_vma_page
net: **heap-out-of-bounds** in sock_setsockopt
use-after-free in sixpack_close
use-after-free in skcipher_sock_destruct
bad page state due to PF_ALG socket
GPF in lrw_crypt
use-after-free in hash_sock_destruct
GPF in gf128mul_64k_bbe
net: user-controllable kmalloc size in __sctp_setsockopt_connectx
fs: WARNING in locks_free_lock_context
9p: sleeping function called from invalid context in v9fs_vfs_atomic_open_dotl
use-after-free in skcipher_bind
crypto: **use-after-free** in rng_recvmsg
crypto: deadlock in alg_setsockopt
crypto: **use-after-free** in alg_bind
...

clang.llvm.org/docs/**AddressSanitizer**.html
clang.llvm.org/docs/**ThreadSanitizer**.html
clang.llvm.org/docs/**MemorySanitizer**.html
clang.llvm.org/docs/**UndefinedBehaviorSanitizer**.html

llvm.org/docs/**LibFuzzer**.html

clang.llvm.org/docs/**ControlFlowIntegrity**.html
clang.llvm.org/docs/**SafeStack**.html

Backup

CFI: VPTR Layout example (simplified)

```
B *b = …
b->f(); // Check VPTR
```