In 5 minutes

Memory Tagging and how it improves C/C++ memory safety

Kostya Serebryany, Google

Aug 2018

https://arxiv.org/pdf/1802.09517.pdf

Memory Safety in C/C++ is a mess

- Heap-**use-after-free**
- Heap-**buffer-overflow**
- Stack-buffer-overflow
- Stack-use-after-return
- Stack-use-after-scope
- Global-buffer-overflow
- Use-of-uninitialized-memory
- ~~Intra-object-buffer-overflow~~ (separate story)

```
char *p = new char[20];

p[20] = …  // OMG

delete [] p;

p[0] = …   // OMG
```

Chrome Releases July 24, 2018

MT covers all High CVEs

[$5000][850350] High CVE-2018-6153: Stack buffer overflow in Skia. *Reported by Zhen Zhou ...*
[$3000][848914] High CVE-2018-6154: Heap buffer overflow in WebGL. *Reported by Omair on 2018-06-01*
[$N/A][842265] High CVE-2018-6155: Use after free in WebRTC. *Reported by Natalie Silvanovich...*
[$N/A][841962] High CVE-2018-6156: Heap buffer overflow in WebRTC. Reported by Natalie Silvanovich ...
[$N/A][840536] High CVE-2018-6157: Type confusion in WebRTC. Reported by Natalie Silvanovich ...
[$2000][841280] Medium CVE-2018-6158: Use after free in Blink. *Reported by Zhe Jin(金哲)...*
[$2000][837275] Medium CVE-2018-6159: Same origin policy bypass in ServiceWorker.*Reported by Jun Kokatsu ...*
[$1000][839822] Medium CVE-2018-6160: URL spoof in Chrome on iOS. *Reported by evi1m0 ...*
[$1000][826552] Medium CVE-2018-6161: Same origin policy bypass in WebAudio.*Reported by Jun Kokatsu ...*
[$1000][804123] Medium CVE-2018-6162: Heap buffer overflow in WebGL. *Reported by Omair on 2018-01-21*
[$500][849398] Medium CVE-2018-6163: URL spoof in Omnibox. *Reported by Khalil Zhani on 2018-06-04*
[$500][848786] Medium CVE-2018-6164: Same origin policy bypass in ServiceWorker.*Reported by Jun Kokatsu*
[$500][847718] Medium CVE-2018-6165: URL spoof in Omnibox. *Reported by evi1m0 of Bilibili Security ...*
[$500][835554] Medium CVE-2018-6166: URL spoof in Omnibox. *Reported by Lnyas Zhang on 2018-04-21*
[$500][833143] Medium CVE-2018-6167: URL spoof in Omnibox. *Reported by Lnyas Zhang on 2018-04-15*
[$500][828265] Medium CVE-2018-6168: CORS bypass in Blink. *Reported by Gunes Acar and Danny Y. Huang of Princeton University, ...*
[$500][394518] Medium CVE-2018-6169: Permissions bypass in extension installation .*Reported by Sam P on 2014-07-16*
[$TBD][862059] Medium CVE-2018-6170: Type confusion in PDFium. *Reported by Anonymous on 2018-07-10*
[$TBD][851799] Medium CVE-2018-6171: Use after free in WebBluetooth. *Reported by amazon@mimetics.ca on 2018-06-12*
[$TBD][847242] Medium CVE-2018-6172: URL spoof in Omnibox. *Reported by Khalil Zhani on 2018-05-28*
[$TBD][836885] Medium CVE-2018-6173: URL spoof in Omnibox. *Reported by Khalil Zhani on 2018-04-25*
[$N/A][835299] Medium CVE-2018-6174: Integer overflow in SwiftShader. *Reported by Mark Brand of Google Project Zero on 2018-04-20*
[$TBD][826019] Medium CVE-2018-6175: URL spoof in Omnibox. *Reported by Khalil Zhani on 2018-03-26*
[$N/A][666824] Medium CVE-2018-6176: Local user privilege escalation in Extensions.*Reported by Jann Horn of Google Project Zero on 2016-11-18*

Every 6-8 weeks on https://chromereleases.googleblog.com, since ~ 2011

ASAN is far from perfect

- **~2x Memory overhead**
  - Shadow, Redzones, Quarantine
  - Also ~2x CPU and Code Size overhead

- Buffer overflows:
  - may jump over redzone

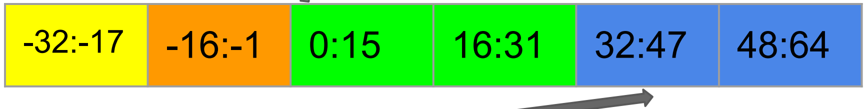
- Use-after-free
  - may “outlive” quarantine

Memory Tagging (MT) in one slide

- 64-bit architectures only
- Every aligned 16 bytes of memory have a 8-bit tag (TG=16, TS=8)
  - Other values for TG/TS are possible
- Every pointer has a tag in the top byte
- malloc/alloca tags memory & pointers with the same tag
- Loads/stores fail on tag mismatch
- Detects use-after-free and buffer-overflow (heap, stack, globals)

Memory Tagging (TG=16, TS=8)

```
char *p = new char[20]; // 0xab007fffffff1240
```




```
p[32] = … // heap-buffer-overflow
```

Memory Tagging (TG=16, TS=8)

```
char *p = new char[20]; // 0xab007fffffff1240
```

-32:-17	-16:-1	0:15	16:31	32:47	48:64

```
delete [] p;   // ⇒
```

-32:-17	-16:-1	0:15	16:31	32:47	48:64

```
p[0] = ...  // heap-use-after-free
```

SPARC ADI

- Available in SPARC M7/M8 CPUs since ~2016
- TG=64, TS=4
- Tl;Dr:
  - works great
  - low CPU overhead, but forces malloc to align by 64
  - heap bugs only (no stack-buffer-overflows)

HWASAN (HardWare-assisted ASAN, Clang/LLVM)

- AArch64-only, needs top-byte-ignore
- TG=16, TS=8
- 2x CPU, **6% RAM,** ~2.5x code size

```
// int foo(int *a) { return *a; }
// clang -O2 --target=aarch64-linux -fsanitize=hwaddress -c load.c
    0:     08 dc 44 d3    ubfx    x8, x0, #4, #52   // shadow address
    4:     08 01 40 39    ldrb    w8, [x8]          // load shadow
    8:     09 fc 78 d3    lsr     x9, x0, #56       // address tag
    c:     3f 01 08 6b    cmp     w9, w8            // compare tags
   10:     61 00 00 54    b.ne    #12               // jump on mismatch
   14:     00 00 40 b9    ldr     w0, [x0]          // original load
   18:     c0 03 5f d6    ret
   1c:     40 20 21 d4    brk     #0x902            // trap
```

MT vs ASAN

- MT:
  - Small RAM overhead
    - 6% with TG=16 TS=8
    - 0.7% with TG=64 TS=4
  - Detection of buffer overflows far from bounds
  - Detection of use-after-free long after deallocation
  - (opionally) initializes memory as a side effect

- ASAN:
  - Precise 1-byte buffer-overflow detection
  - More portable (32-bit, non-aarch64)

MT is good for

- Testing
  - Alternative to ASAN, consumes much less RAM
- Bug detection in production
  - Crowd-sourced bug detection
  - If CPU, RAM, Code size overheads are tolerable
  - SPARC ADI - yes, HWASAN - hm, maybe
- Security mitigation: likely yes.

Home work

Analyze you favourite exploit: is it preventable by MT?

Ask your CPU vendor to implement memory tagging