GWP-ASAN

Finding the worst bugs in prod

Dmitry Vyukov, dvyukov@

Linux Plumbers 2019

More bugs!

But WHY? ¯_(ツ)_/¯

Not all bugs are equal!

Want: **BAD** bugs...

that affect **production**

GWP-ASAN to the rescue!

"**G**WP-ASan **W**ill **P**rovide **A**llocation **SAN**ity"

KASAN Crash Course

- Finds use-after-free, out-of-bounds, double-free

KASAN Crash Course

- Finds use-after-free, out-of-bounds, double-free
- compiler instrumentation before each memory access
  - check if the access is legit

KASAN Crash Course

- Finds use-after-free, out-of-bounds, double-free
- compiler instrumentation before each memory access
    - check if the access is legit
- ⅛-th shadow memory
    - 1 shadow byte encodes addressability of 8 kernel bytes

KASAN Crash Course

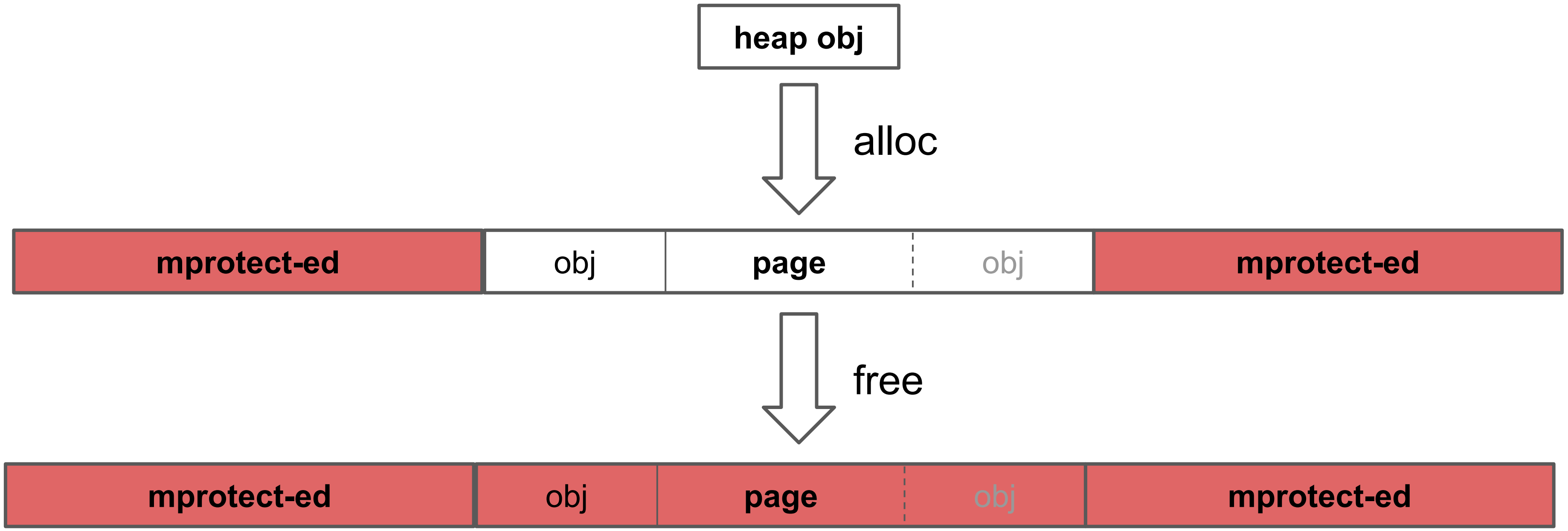
- Finds use-after-free, out-of-bounds, double-free
- compiler instrumentation before each memory access
  - check if the access is legit
- ⅛-th shadow memory
  - 1 shadow byte encodes addressability of 8 kernel bytes
- red-zones around objects (16-4096 bytes)
- quarantine for freed objects (1/32-th of RAM)

KASAN Crash Course

- Finds use-after-free, out-of-bounds, double-free
- compiler instrumentation before each memory access
    - check if the access is legit
- ⅛-th shadow memory
    - 1 shadow byte encodes addressability of 8 kernel bytes
- red-zones around objects (16-4096 bytes)
- quarantine for freed objects (1/32-th of RAM)
- **~~2x slowdown, ~~2x memory overhead**

:(

Electric fence (Page Heap, Page Guard)



Electric fence

- Pro: Detects bugs!
- Pro: No instrumentation -- can be enabled at runtime
- Con: Really expensive
    - Horrible heap fragmentation
        - A 1-byte allocation requires a full 4KB page for buffer overflow detection
        - Most allocations are much smaller than 4KB
    - Slow
        - malloc/free require a system call to mmap/mprotect (DTLB flush)

:(

Use the **sampling**, Luke!

Electric fence + Sampling

- Sample every N-th allocation
- Surround with guard pages, protect on free

Electric fence + Sampling

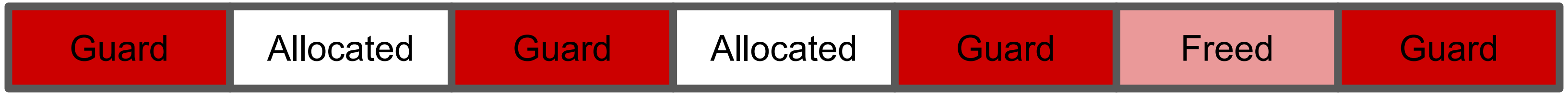
- Sample every N-th allocation
- Surround with guard pages, protect on free
- CPU overhead: set sample rate low enough (1/1000)
- RAM overhead: limit max number of sampled objects

Electric fence + Sampling

- Sample every N-th allocation
- Surround with guard pages, protect on free
- CPU overhead: set sample rate low enough (1/1000)
- RAM overhead: limit max number of sampled objects
- Ideally: applied to a large set of machines

Memory layout

Fixed number of pages preallocated on startup



128 * 8KB = 1MB

Metainfo

- Alloc/free stack, PID, ...
- Provide actionable reports
- Stored somewhere on the side
- Updated only on slow-paths

Current use

- in LLVM upstream (enabled in SCUDO allocator)

Current use

- in LLVM upstream (enabled in SCUDO allocator)
- Internally
  - enabled in all tests
    - more aggressive sampling, .5% CPU

Current use

- in LLVM upstream (enabled in SCUDO allocator)
- Internally
  - enabled in all tests
    - more aggressive sampling, .5% CPU
  - enabled in prod by default
    - more conservative sampling, ~0% CPU
    - hundreds of bugs (most "P0")
    - bugs are auto-filed

Current use

- in LLVM upstream (enabled in SCUDO allocator)
- Internally
  - enabled in all tests
    - more aggressive sampling, .5% CPU
  - enabled in prod by default
    - more conservative sampling, ~0% CPU
    - hundreds of bugs (most "P0")
    - bugs are auto-filed
- Chrome
  - enabled in subset of end-user builds
  - lots of bugs (some are not public)

Current use

- in LLVM upstream (enabled in SCUDO allocator)
- Internally
    - enabled in all tests
        - more aggressive sampling, .5% CPU
    - enabled in prod by default
        - more conservative sampling, ~0% CPU
        - hundreds of bugs
        - bugs are auto-filed
- Chrome
    - enabled in subset of end-user builds
    - lots of bugs (some are not public)
- Android
    - WIP

Potential kernel uses

- Distros
- Cloud/datacenters
- Consumer/industrial devices
- [your use case here]

kmalloc fast-path

Can we do it with zero overhead?

Piggyback on some existing slow-path?

Potential tunables

- enable/disable
- panic/don't panic
- memory overhead
- sampling rate

Small devices

Feasible?

Less code/workload -> fewer allocations

Sampling strategy

- persistent objects deplete the pool
- infrequent reboots
- don't sample during boot?
- start sampling after X uptime?
- don't sample some allocations

Volunteers?

Thanks!

Q&A

Dmitry Vyukov, dvyukov@