# AddressSanitizer
# No more memory corruption

Konstantin Serebryany
Alexander Potapenko
June 9 2011

Google™

# AddressSanitizer (ASan)

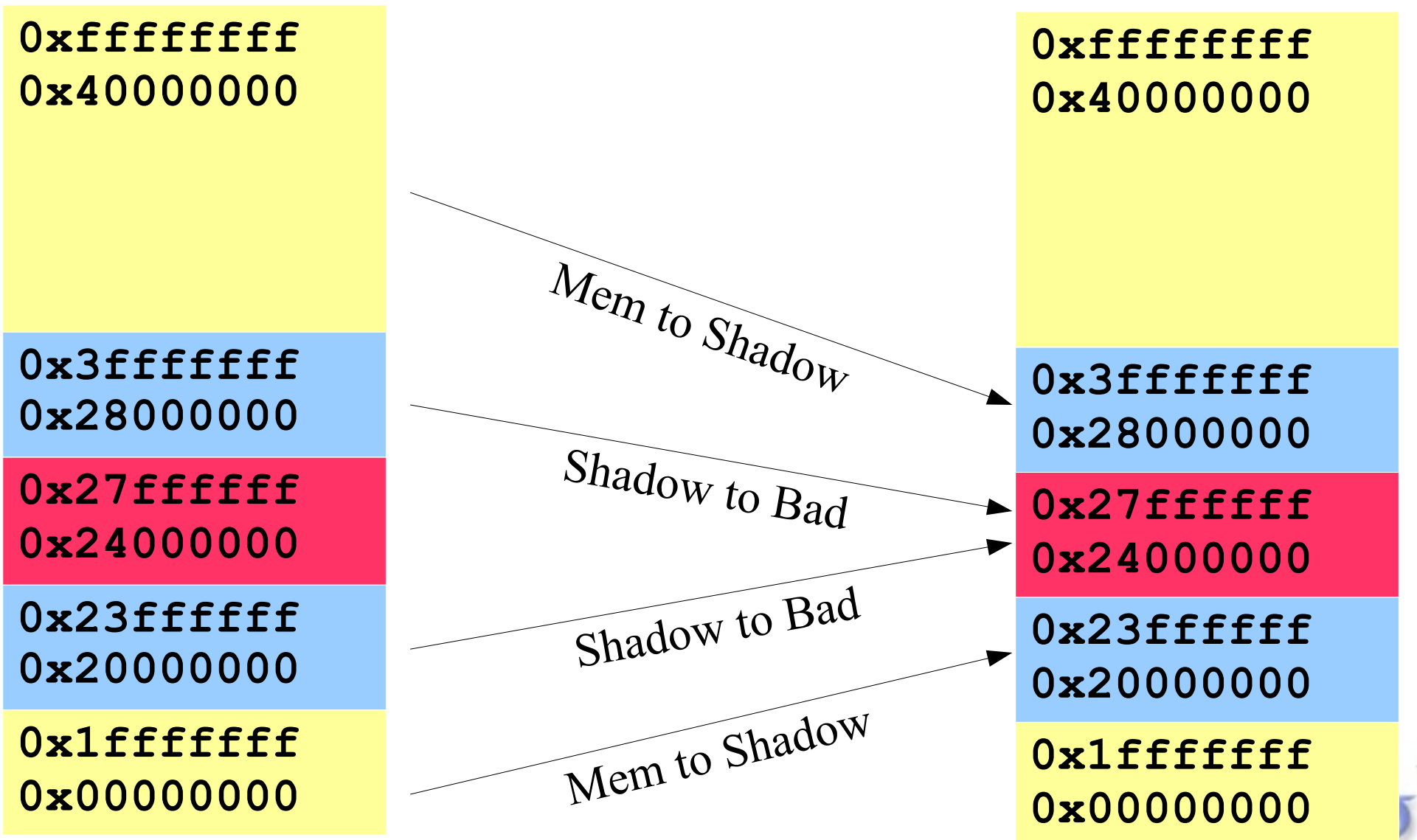
- Many memory error detectors exist:
  - Instrumentation:
    - Valgrind, DrMemory, Purify, BoundsChecker, Insure++, Intel Inspector, mudflap, ...
  - Guard page (libgmalloc, Electric Fence, Page Heap, …)
  - Debug malloc (google perftools, glibc,. ..)
- **AddressSanitizer** (ASan): fast address sanity checker
  - Use-after-free
  - Out-of-bound (overflow/underflow) for heap and stack
  - Linux, ChromeOS; Soon: Mac
  - ~2x slowdown (faster than Debug build!)
  - LLVM instrumentation module + specialized malloc

Google™

# Generic addressability checking

- malloc()/free() replacement library (most tools):
  - poison redzones around malloc-ed memory
  - poison memory on free()
  - delay reuse of free-ed memory
- Stack poisoning (some tools)
- Instrument all loads and stores
  - `if (IsPoisoned(mem)) Crash();`
- Tricky: how to implement `IsPoisoned()` and `Crash()`

# Shadow = (Mem >> 3) + kOffset



# Poison

- 9 states for any 8 aligned bytes
  - All 8 are addressable
    - Shadow = 0
  - All 8 are not addressable
    - Shadow = -1
  - First k (1..7) are addressable, the rest 8-k are not.
    - Shadow = k

# Instrumentation: one extra load

Instrumenting 8-byte access to **Mem**:

```
Shadow = *(int8_t*)((Mem >> 3) + kOffset);
if (Shadow) {
  Crash(Mem, 8, isWrite);
}
```

Instrumenting N-byte (1,2,4) access to **Mem**:

```
Shadow = *(int8_t*)((Mem >> 3) + kOffset);
if (Shadow) {
  if ((int8_t)((Mem & 7) + N) > Shadow)
    Crash(Mem, N, isWrite);
}
```

Google

# Issue: unaligned, partially OOB

```
int *x = new int[2]; // 8 bytes:    [0,7].
int *u = (int*)((char*)x + 6);
*u = 1;  // Access to range [6-9]
```

- Solution 1: byte-to-byte mapping
  - Only 64-bit (?)
  - More memory, slower poisoning
- Solution 2: special case (`if (mem & 3)` …)
  - Slower, more code
- Solution 3: ignore
- Solution 4: forbid

# Crash()

- Need: address, isWrite, Size
- Call is harder to deploy, creates prologue/epilogue
- With disassembler: int3/ud2 is enough
- W/o disassembler: various ways, current is 5-6 bytes:

```
89 c0   mov    %ecx,%eax   ; address in %eax
0f 0b   ud2a               ; generate SIGILL
58      pop    %eax        ; Size, isWrite
```

# Compile-time optimizations

```
// Instrument only the first access
*a = …
if (...) *a = …
// Instrument only the second access (?)
if (...) *a = …
*a = …
// Instrument only a[0] and a[n-1]
for (int i = 0; i < n; i++) a[i] = i;
// Combine two accesses into one
struct { int a, b; } x; …
x.a = …; x.b = …
```

Google

# Q&A

Google™