Implementing KMSAN on PowerPC

Everything Open 2024

Nicholas Miehlbradt

How did I get here?

- PowerPC Kernel Developer at IBM
- Worked on other memory sanitizer(ish) tools
  - KFENCE
  - Arch specific harden user copy feature

Everything Open2024

Overview

- What is KMSAN
  - Kernel side
  - Compiler side
- Implementing KMSAN
- What has come of it

What is KMSAN

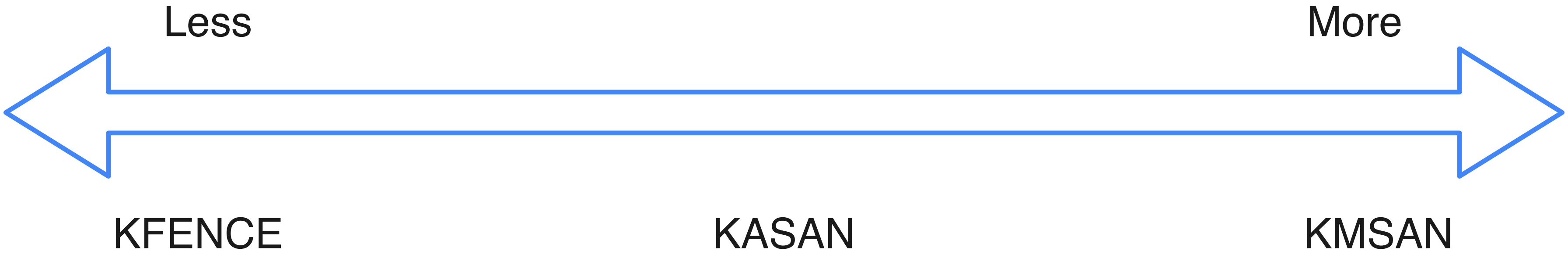
Everything Open 2024

What is KMSAN

Big Idea:

Detect and report undefined uses of uninitialized memory and uninitialized memory copied to user.

What is KMSAN



Less

More

KFENCE

KASAN

KMSAN

What counts as undefined?

Ok

- `r = uninit + b`

Not Ok

- Conditions

```
if (uninit) ...
```

- Pointer Dereferences

```
*uninit
```

- Array Accesses

```
uninit[i]
buf[uninit]
```

Everything Open 2024

Whats the problem?

```
void main() {
    int c;
    if (c)
        printf(“got here\n”);
    else
        printf(“got there\n”);
}
```

Possible outputs

```
$ main.c
got here

$ main.c
got there

$ main.c
got here
got there

$ main.c
```

What is KMSAN

- For every byte there is a shadow byte
  - Track whether each bit is initialized or not
- For every 4 bytes there are 4 origin bytes
  - Tracks where the 4 bytes were last written to

Everything Open 2024

```
BUG: KMSAN: uninit-value in test_uninit_kmsan_check_memory+0x1be/0x380 [kmsan_test]
 test_uninit_kmsan_check_memory+0x1be/0x380 mm/kmsan/kmsan_test.c:273
 kunit_run_case_internal lib/kunit/test.c:333

Uninit was stored to memory at:
 do_uninit_local_array+0xfa/0x110 mm/kmsan/kmsan_test.c:260
 test_uninit_kmsan_check_memory+0x1a2/0x380 mm/kmsan/kmsan_test.c:271

Local variable uninit created at:
 do_uninit_local_array+0x4a/0x110 mm/kmsan/kmsan_test.c:256
 test_uninit_kmsan_check_memory+0x1a2/0x380 mm/kmsan/kmsan_test.c:271

Bytes 4-7 of 8 are uninitialized
Memory access of size 8 starts at ffff888083fe3da0
```

```
static void test_uninit_kmsan_check_memory(struct kunit *test) {
    volatile char local_array[8] = { 0 };

    do_uninit_local_array((char *)local_array, 4, 7);

    kmsan_check_memory((char *)local_array, 8);
}

static void do_uninit_local_array(char *array, int start, int stop) {
    volatile char uninit;

    for (int i = start; i < stop; i++)
        array[i] = uninit;
}
```

Everything Open 2024

How it works

For every three pages:

- One is used as normal
- One becomes the shadow metadata
- One becomes the origin metadata

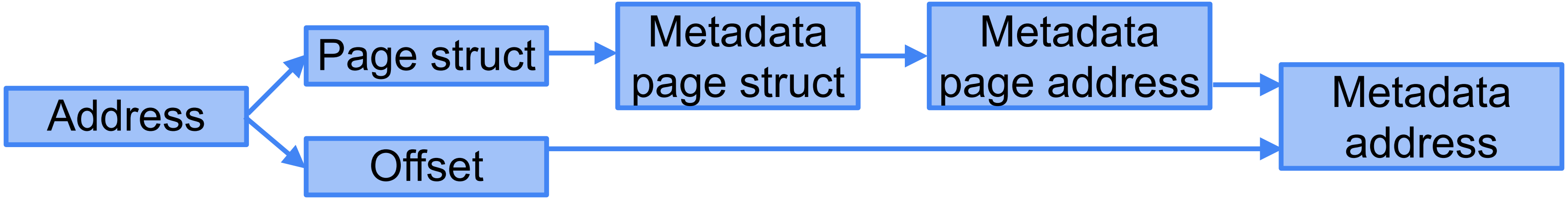
Stored in the page struct

```
 struct page {
    ...
    struct page *shadow, *origin;
    ...
};
```

This association is made when pages are returned from the memblock allocator

How it works

For physical addresses or linear map



Address

Page struct

Offset

Metadata page struct

Metadata page address

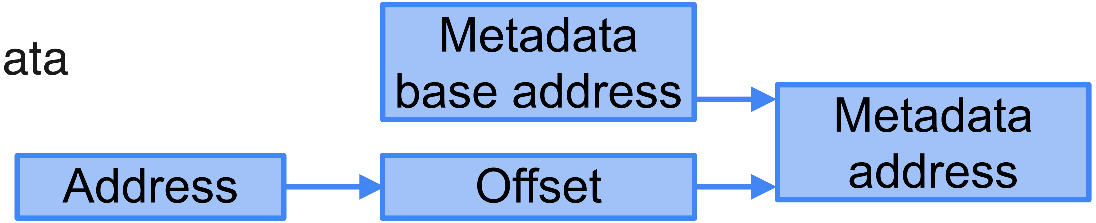
Metadata address

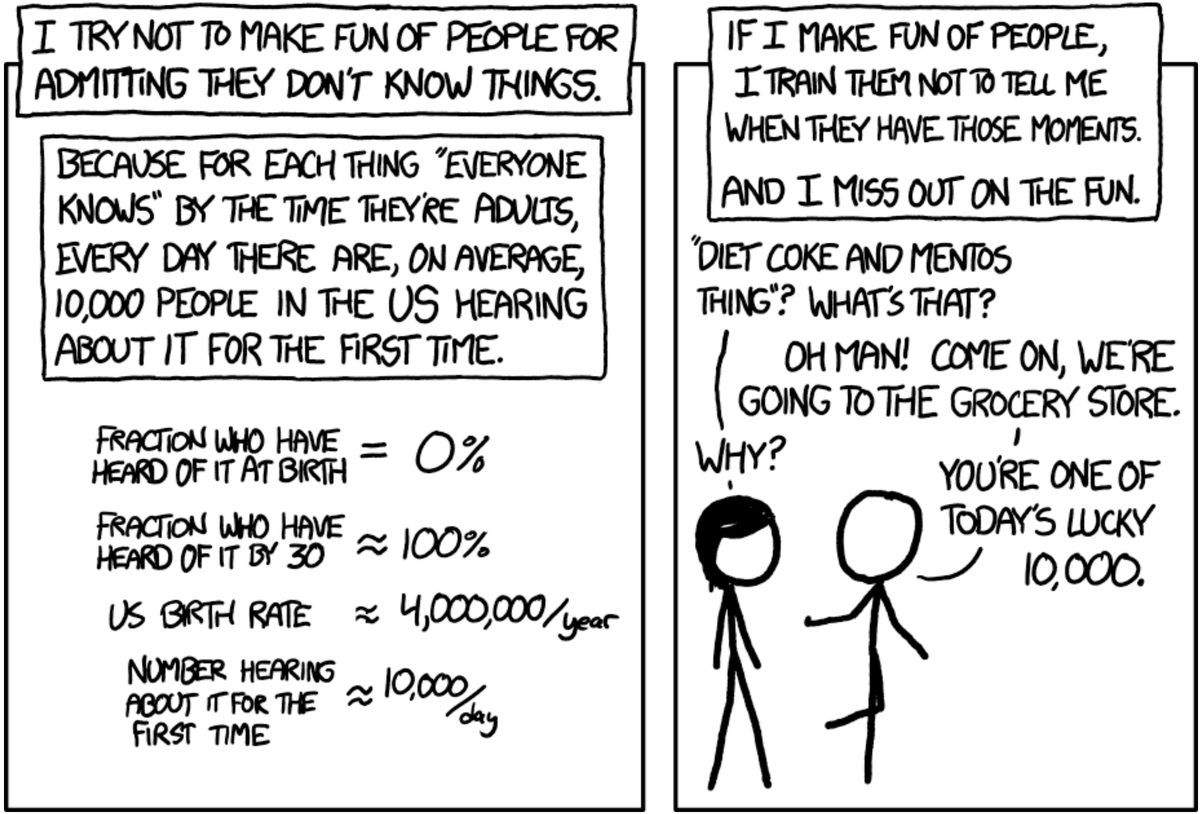
How it works

For vmalloc addresses

- vmalloc region is divided into four
- 1st quarter is the new vmalloc region
- 2nd quarter is shadow metadata
- 3rd quarter is origin metadata

- When a page is mapped into virtual memory it's shadow and origin pages are mapped into the right place




I TRY NOT TO MAKE FUN OF PEOPLE FOR ADMITTING THEY DON'T KNOW THINGS.
BECAUSE FOR EACH THING "EVERYONE KNOWS" BY THE TIME THEY'RE ADULTS, EVERY DAY THERE ARE, ON AVERAGE, 10,000 PEOPLE IN THE US HEARING ABOUT IT FOR THE FIRST TIME.
FRACTION WHO HAVE HEARD OF IT AT BIRTH = 0%
FRACTION WHO HAVE HEARD OF IT BY 30 ≈ 100%
US BIRTH RATE ≈ 4,000,000/year
NUMBER HEARING ABOUT IT FOR THE FIRST TIME ≈ 10,000/day
IF I MAKE FUN OF PEOPLE, I TRAIN THEM NOT TO TELL ME WHEN THEY HAVE THOSE MOMENTS.
AND I MISS OUT ON THE FUN.
"DIET COKE AND MENTOS THING"? WHAT'S THAT?
OH MAN! COME ON, WE'RE GOING TO THE GROCERY STORE.
WHY?
YOU'RE ONE OF TODAY'S LUCKY 10,000.

Everything Open 2024

Propagating metadata

```
struct shadow_origin_ptr {
    void *shadow, *origin
}

char val = *ptr
struct shadow_origin_ptr val_meta =
    __msan_metadata_ptr_for_load_1(ptr)
...
```

```
r = a + b
r_shadow = a_shadow | b_shadow
```

Using metadata

```
void foo() {
    ...
    if (cond_var_shadow)
        __msan_warning();
    if (cond_var) {
        ...
    }
}
```

Compiler changes

```
-   if (IsX86_64)
+   if (IsX86_64 || IsPowerPC64)
      Res |= SanitizerKind::KernelMemory;
```

Skipping sanitization

Two ways:

1. `__no_sanitize_memory`
    - Don’t insert any instrumentation at all
2. `__no_kmsan_checks`
    - Don’t do any checks but mark return values as initialized

Implementing KMSAN

*For PowerPC64 guests

Getting it to compile

```
static inline void *arch_kmsan_get_meta_or_null(void *addr, bool is_origin)
{
    unsigned long addr64 = (unsigned long)addr, off;
    if (KERN_IO_START <= addr64 && addr64 < KERN_IO_END) {
        off = addr64 - KERN_IO_START;
        return (void *)off + (is_origin ? KERN_IO_ORIGIN_START : KERN_IO_SHADOW_START);
    } else {
        return 0;
    }
}

static inline bool kmsan_virt_addr_valid(void *addr)
{
    return (unsigned long)addr >= PAGE_OFFSET && pfn_valid(virt_to_pfn(addr));
}
```

Getting it to compile

… plus some macro definitions

```
#define VMALLOC_LEN ((__vmalloc_end - __vmalloc_start) >> 2)
#define VMALLOC_END (VMALLOC_START + VMALLOC_LEN)
#define KMSAN_VMALLOC_SHADOW_START VMALLOC_END
#define KMSAN_VMALLOC_ORIGIN_START (VMALLOC_END + VMALLOC_LEN)
```

Thank you

Everything Open 2024

Piles of assembly

PowerPC has a lot of raw assembly

- Interrupt handling
  - Saving machine state into pt_regs
- memcpy, memmove, memset
  - For speed

Interrupt handling

PowerPC has a lot of raw assembly

- Interrupt handling
  - Saving machine state into pt_regs

Solution: use `kmsan_unpoison_entry_regs()`

Mem functions

- Solution: macros

```
#ifdef CONFIG_KMSAN
void *__memcpy(void *to, const void *from, __kernel_size_t n);

#ifdef __SANITIZE_MEMORY__
#include <linux/kmsan_string.h>
#define memcpy __msan_memcpy
```

Everything
Open 2024

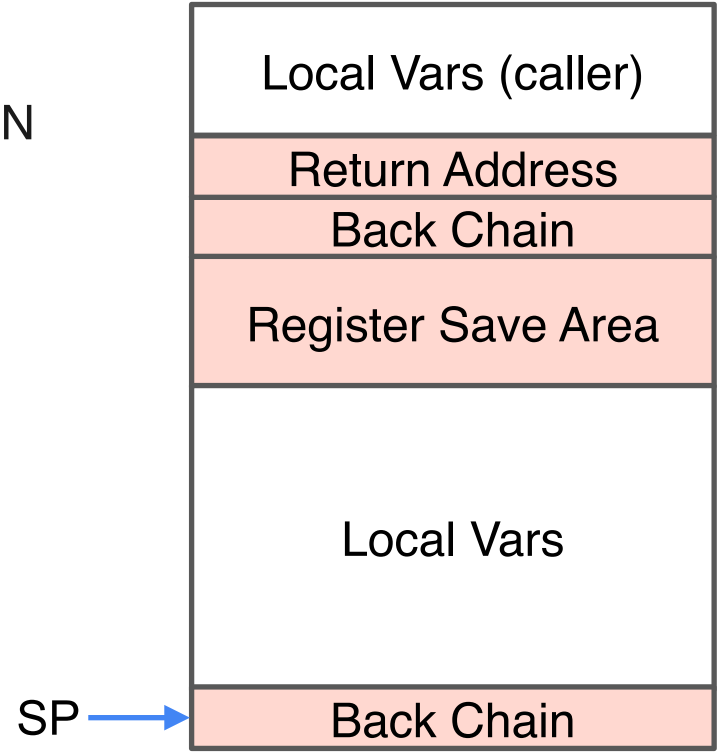
Walking the stack

- Some loads/stores are still invisible to KMSAN

```
%retval = call i32 @foo(i32 %arg)
```

- Stack frame layout is arch specific so isn’t represented in LLVM IR

- Some functions make use of arch specific parts of the stack



Walking the stack

- `perf_callchain_kernel()`
- `show_stack()`
- `arch_stack_walk()`

Solution: Mark functions as `__no_kmsan_checks`

Interfacing with other code

The kernel is not the only thing running in supervisor mode

We need to interact with the hypervisor to request resources

```
        unsigned long retbuf[PLPAR_HCALL_BUFSIZE];
        ret = plpar_hcall(H_GET_TERM_CHAR, retbuf, vtermno);
```

Solution: `kmsan_unpoison_memory(retbuf, sizeof(retbuf))`

Subject: [PATCH 00/32] kmsan: Enable on s390

- Validated the work I had done
- Revealed issues I didn’t know about
  - `DEFERRED_STRUCT_PAGE_INIT`

Everything Open 2024

What about hash?

After successfully booting the kernel without warnings on radix how hard could it be on hash?

```
[    0.000000] Booting Linux
```

…And it hangs

Static branches

Some conditional branches always go the same way e.g. MMU feature checks

Conditional branches can be replaced with static branches or nops

```
static bool mmu_has_feature(unsigned long feature) {
    ...
    i = __builtin_ctzl(feature);
    return static_branch_likely(&mmu_feature_keys[i]);
}
```

Back to hash

```
struct shadow_origin_ptr get_shadow_origin_ptr(...) {
    unsigned long ua_flags = user_access_save();
    struct shadow_origin_ptr ret;

    ret = kmsan_get_shadow_origin_ptr(...);
    user_access_restore(ua_flags);
    return ret;
}
```

```
struct shadow_origin_ptr
kmsan_get_shadow_origin_ptr(...) {
    if (!kmsan_enabled)
        goto return_dummy;
    ...
}
```

Was it worth it?

Everything Open 2024

“Bug” #1

```
static void udbg_hvc_putc(char c) {
    int count = -1;
    unsigned char bounce_buffer[16];

    do {
        bounce_buffer[0] = c;
        count = hvterm_raw_put_chars(0, bounce_buffer, 1);
    } while (count == 0 || count == -EAGAIN);
}
```

“Bug” #2

```
int rtc_read_alarm_internal(struct rtc_device *rtc, struct rtc_wkalrm *alarm) {
    int err;

    if (!rtc->ops)
        err = -ENODEV;
    else if (!test_bit(RTC_FEATURE_ALARM, rtc->features)
             || !rtc->ops->read_alarm)
        err = -EINVAL;
    else {
        alarm->enabled = 0;
        ...
        alarm->time.tm_isdst = -1;
        err = rtc->ops->read_alarm(rtc->dev.parent, alarm);
    }
    trace_rtc_read_alarm(rtc_tm_to_time64(&alarm->time));
    return err;
}
```

Next Steps

- Get it working with hash MMU
  - There are still a number of false positives
- Run it in syzkaller
  - See the talk on Thursday by Andrew Donnellan

Thank you

Everything Open 2024

Other Links

KMSAN Docs

KernelMemorySanitizer - a look under the hood

Presentation Slides:

github.com/NMiehlbradt/talks

Everything Open 2024