arm

ARM v8.5 Memory Tagging Extension

Vincenzo Frascino

Agenda

- Memory Tagging Extension Introduction
- Kernel ABI and Top Byte Ignore
- MTE Enabled Kernel Interface
- How does it work?

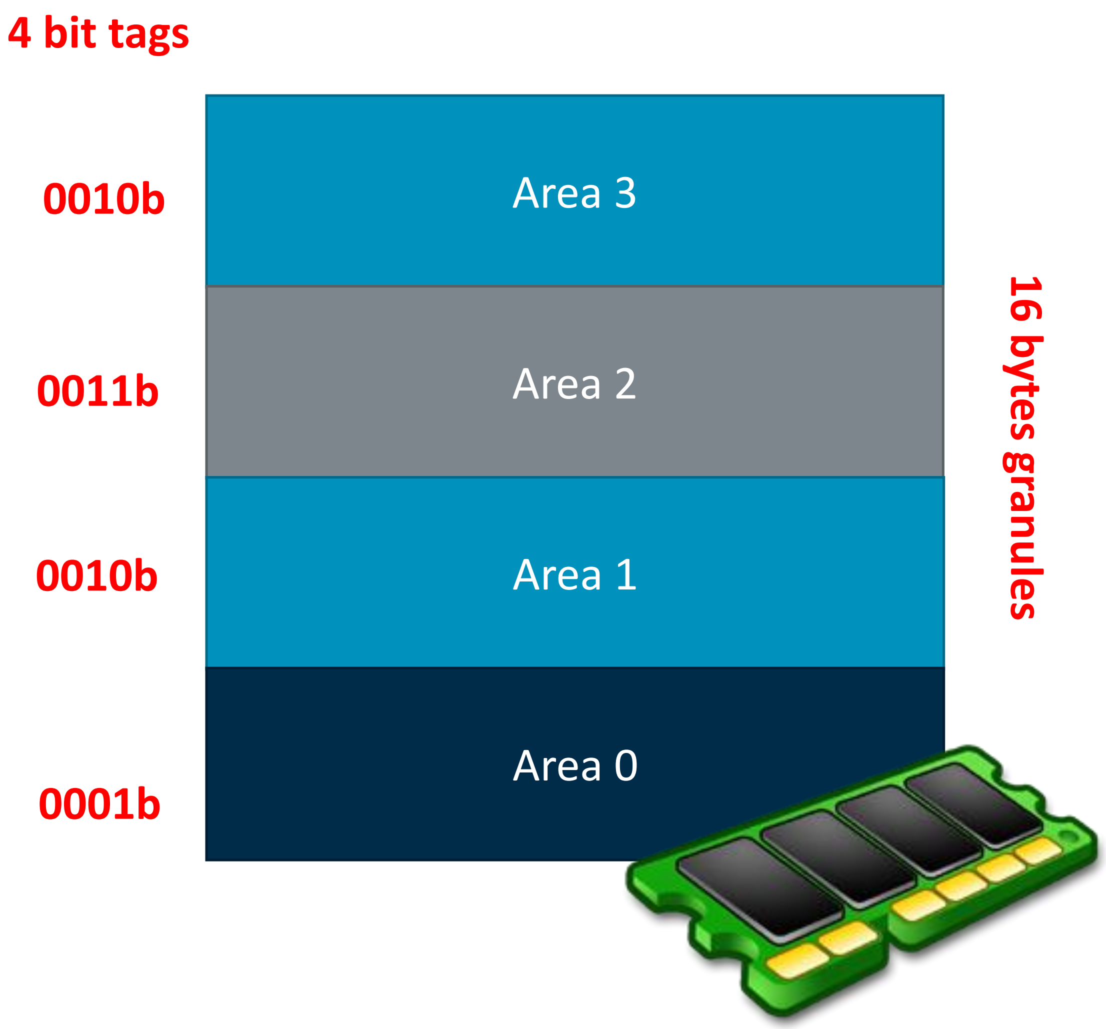
Memory Tagging Extension Introduction (1/2)

- The ARM v8.5 Memory Tagging Extension provides architectural support for run-time, always-on detection of various classes of memory error:
  - bounds violations
  - use-after-free
  - use-after-return
  - use-out-of-scope
  - use-before-initialisation
- The purpose of the extension is to aid with software debugging and to eliminate vulnerabilities before they can be exploited.
- The Memory Tagging Extension is built on top of the top-byte-ignore feature in ARMv8.0.

arm

Memory Tagging Extension Introduction (2/2)

- The MTE extension introduces a set of new instructions to address various classes of memory errors.
- The extension is mainly based on the Lock/Key mechanism.
- It can make easier addressing errors related to Stack and Heap allocations.
- To use tagging with heap allocations only the allocator needs to make use of the new instructions, the rest of the code only performs standard LDR/STR.



arm

Kernel ABI and Top Byte Ignore

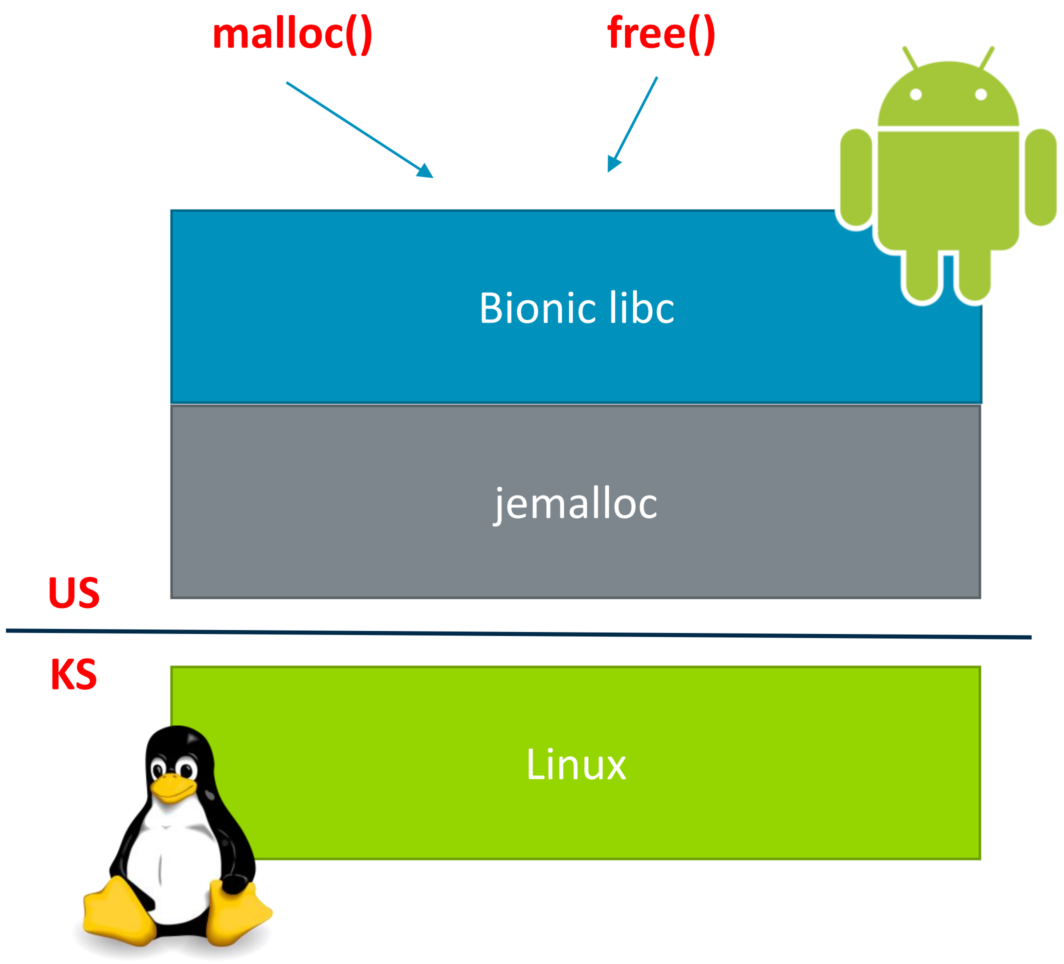
- On AArch64 the **TCR_EL1.TBI0** bit is set by default.
- When the AArch64 Tagged Address ABI is enabled for a thread, the following behaviours are guaranteed:
  - All syscalls (except prctl(), ioctl(), shmat() and shmdt()) can accept any valid tagged pointer.
  - The syscall behaviour is undefined for invalid tagged pointers: it may result in an error code being returned, a (fatal) signal being raised, or other modes of failure.
  - The syscall behaviour for a valid tagged pointer is the same as for the corresponding untagged pointer.
- For more details refer to: **Documentation/arm64/tagged-address-abi.rst**

MTE Enabled Kernel Interface

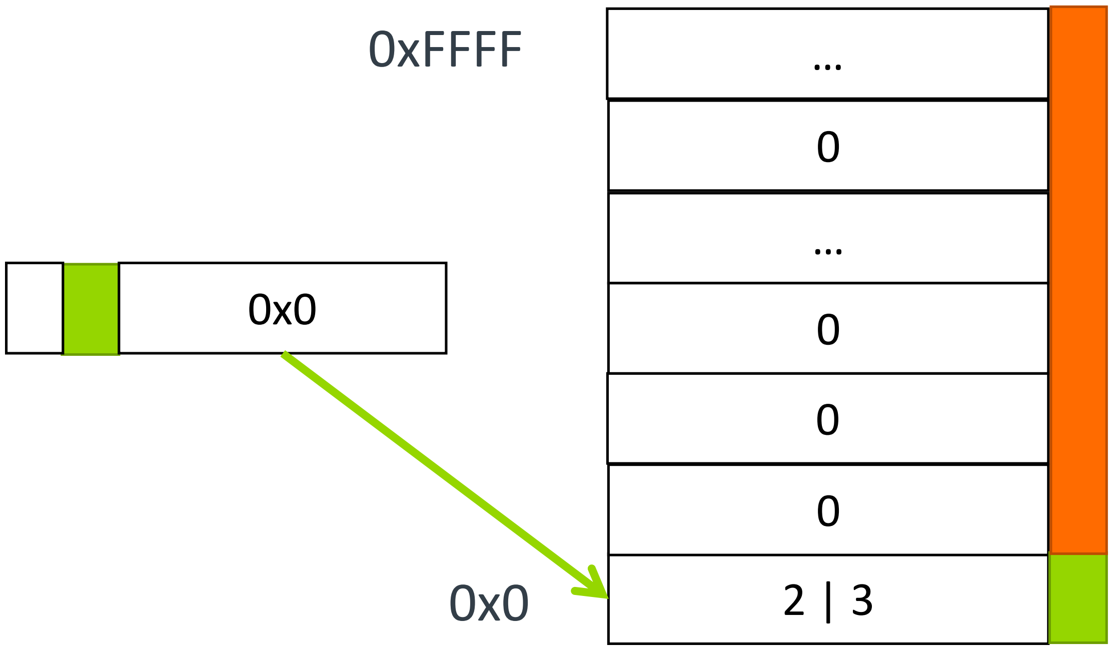
- MTE Kernel interface is built on top of the newly introduced Aarch64 Tagged Address ABI.
- The Memory Tagging Extension is enabled by default by the Kernel.
  - The Kernel exposes a new mmap()/mprotect() flag: PROT_MTE.
- The Kernel supports both the exception types: Precise and Imprecise.
- The default mode is controlled via sysctl.
- The user applications can always select Precise mode through prctl().

How does it work?

- The userspace allocates memory via malloc().
- A malloc() call is handled by the memory allocator, which ultimately invokes mmap() to reserve memory for the process.
- If mmap() is invoked with a special flag, **PROT_MTE,** the reserved memory has tagging effects enabled.
- In this case, the allocator tags the memory and returns to the application a tagged pointer.



How does it work? (Example)



Tag Size = 4 bits

Granule Size = 16 bytes

```
int main()
{
    unsigned long *a;
    unsigned long page_sz = getpagesize();

    a = mmap(0, page_sz, PROT_READ | PROT_WRITE | PROT_MTE,
             MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (a == MAP_FAILED) {
        printf("Could not mmap with PROT_MTE");
        return -1;
    }

    a[0] = 1;
    a[1] = 2;

    a = (unsigned long *)insert_random_tag((void *)a);
    set_tag((void *)a);

    printf("%p\n", a);
    a[0] = 3;
    printf("a[0] = %lu a[1] = %lu\n", a[0], a[1]);

    a[256] = 0xdead;

    return 0;
}
```

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה