mach_header 64bit and __PAGEZERO segment 64bit

Asked 10 years, 8 months ago Modified 10 years, 8 months ago Viewed 3k times

Part of Mobile Development Collective

1

```
const struct mach_header *mach = _dyld_get_image_header(0);
struct load_command *lc;
struct segment_command_64 *sc64;
struct segment_command *sc;

if (mach->magic == MH_MAGIC_64) {
    lc = (struct load_command *)((unsigned char *)mach + sizeof(struct
mach_header_64));
    printf("[+] detected 64bit ARM binary in memory.\n");
} else {
    lc = (struct load_command *)((unsigned char *)mach + sizeof(struct
mach_header));
    printf("[+] detected 32bit ARM binary in memory.\n");
}

for (int i = 0; i < mach->ncmds; i++) {

    if (lc->cmd == LC_SEGMENT) {
        sc = (struct segment_command *)lc;
        NSLog(@"32Bit: %s (%x - 0x%x)",sc->segname,sc->vmaddr,sc->vmsize);
    } else if (lc->cmd == LC_SEGMENT_64) {
        sc64 = (struct segment_command_64 *)lc;
        NSLog(@"64Bit: %s (%llx - 0x%llx)",sc64->segname,sc64->vmaddr,sc64-
>vmsize);
    }
    lc = (struct load_command *)((unsigned char *)lc+lc->cmdsize);
}
```

When I run this code in 32Bit I get normal outputs:

```
__PAGEZERO (0 - 0x1000)
But on 64Bit: __PAGEZERO (0 - 0x100000000)
```

`__PAGEZERO goes from 0x1000 to over 0x100000000` in size, is there any fix for it or any solution why this occurs?

MD c++ ios objective-c c

Share

Improve this question

Follow

edited Apr 20, 2014 at 22:59

Larme

26k ● 6 ● 58 ● 86

asked Apr 20, 2014 at 22:37

Janosch Hübner

1,694 ● 1 ● 27 ● 49

Why is that a problem that needs to be fixed? – stark Apr 20, 2014 at 22:43

well 0x100000000 is about 4.29 GB. and the file I run this on is about 107kb, which means there must be a wrong count somewhere. – Janosch Hübner Apr 20, 2014 at 22:47

or a wrong understanding of virtual memory addresses – stark Apr 20, 2014 at 23:03

It's exactly 4GB. But given that the total memory is 64-bits (in actual fact, it probably isn't the full 64-bit, but regardless), and even LARGE machines these days don't have much more than 1TB of RAM, wasting the first 4GB of address space (not physical memory, I believe) as "catch NULL pointer accesses") would be fine. – Mats Petersson Apr 20, 2014 at 23:03

1 Answer

Sorted by: Highest score (default)

5

Making a big `__PAGEZERO` in a 64-bit architecture makes a whole lot of sense. The address range of a 64-bit system, even when the upper 16 bits are "cropped off" like that of x86_64, allows for a huge amount of memory (the 48-bit address space of x86_64 is 256TB of memory address space). It is highly likely that this will be thought of as "small" at some point in the future, but right now, the biggest servers have 1-4TB, so there's plenty of room to grow, and more ordinary machines have 16-32GB.

Note also that no memory is actually OCCUPIED. It's just "reserved virtual space" (that is, "it will never be used"). It takes up absolutely zero resources, because it's not mapped in the page-table, it's not there physically. It's just an entry in the file, which tells the loader to reserve this space to it can never be used, and thus "safeguarded". The actual "data" of this section is zero in size, since, again, there's actually nothing there, just a "make sure this is not used". So your actual file size won't be any larger or smaller if this section is changed in size. It would be a few bytes smaller (the size of the section description) if it didn't exist at all. But that's really the only what it would make any difference at all.

The purpose of a `__PAGEZERO` is to catch NULL pointer dereferences. By reserving a large section of memory at the beginning of memory, any access through a NULL pointer will be caught and the application aborted. In a 32-bit architecture, something like:

```
int *p = NULL;
int x = p[0x100000];
```

is likely to succeed, because at 0x400000 (4MB) the code-space starts (trying to write to such a location is likely to crash, but reading will work - assuming of course the code-space actually starts there and not someplace else in the address range.

Edit:

This presentation shows that ARM, the latest entrant into the 64-bit processor sapce, is also using 48-bit virtual address space, and enforces canonical addresses (top 16 bits need to all be the same value) so it can be expanded in the future. In other words, the virtual space available on a 64-bit ARM processor is also 256TB.

Share

Improve this answer

Follow

edited Apr 20, 2014 at 23:20

answered Apr 20, 2014 at 23:13

Mats Petersson

129k ● 14 ● 145 ● 232

3 In addition to catching `NULL` dereferences, using a size of `0x100000000` in 64-bit means that no 32-bit pointer is valid. This helps catch buggy software that has been compiled for 64-bit but is not 64-bit safe. For example, it copies a pointer to an `int` and then back to a pointer variable, truncating it. – Ken Thomases Apr 21, 2014 at 4:11