# kmem 命令

该命令主要用来显示kernel memory的使用。

常见option

## kmem <addr>

when used without any flag, the address can be a kernel virtual, or physical address; a search is made through the symbol table, the kmalloc() slab subsystem, the free list, the page\_hash\_table, the vmalloc() region subsystem, the current set of task\_structs and kernel stacks, and the mem\_map array. If found in any of those areas, the information will be dumped in the same manner as if the location-specific flags were used; if contained within a curent task\_struct or kernel stack, that task's context will be displayed.

【举例1：】

crash> kmem c3fbdb60

CACHE NAME OBJSIZE ALLOCATED TOTAL SLABS SSIZE

c8402970 blkdev\_requests 96 30208 37800 945 4k

SLAB MEMORY TOTAL ALLOCATED FREE

c3fbd020 c3fbd0e0 40 40 0

FREE / [ALLOCATED]

[c3fbdb60]

/\* 有中括号的表示该地址以及被分配，没有中括号的表示是free的地址。这里显示的信息包括memory的name，slab的分配情况，以及当前地址的分配情况等信息。 通过rd命令可以进一步确认该地址所在内存的信息。\*/

【举例2：】

crash-arm64> kmem ffffffc030e4f038

CACHE NAME OBJSIZE ALLOCATED TOTAL SLABS SSIZE

ffffffc0a3c5a780 skbuff\_head\_cache 224 18 100 4 16k

SLAB MEMORY NODE TOTAL ALLOCATED FREE

ffffffbdc0c39300 ffffffc030e4c000 0 25 9 16

FREE / [ALLOCATED]

[ffffffc030e4ef80] /\* 与命令行地址不一样，这个地址应是命令行地址所在的slab的起始地址。 \*/

PAGE PHYSICAL MAPPING INDEX CNT FLAGS

ffffffbdc0c393c0 b0e4f000 0 0 0 0

crash-arm64> rd ffffffc030e4ef80 2 /\* 读取该slab内存，发现已经free了 \*/

ffffffc030e4ef80: bbbbbbbbbbbbbbbb bbbbbbbbbbbbbbbb ................//free

## kmem -S <addr>

when used with -s or -S, searches the kmalloc() slab subsystem for the slab containing of this virtual address, showing whether it is in use or free.

【举例1：】

<http://bugzilla.spreadtrum.com/bugzilla/show_bug.cgi?id=779517>

通过现场能够获取rdev(0xeb998040)的地址，该地址是合法地址，rdev的数据结构也未见明显异常。由rdev可推导得到wdev(0xe8e2d644)，解析wdev链表，下一个wdev节点地址是0xeb998074，它的next又指向0xe8e2d644，这里应该是链表的尾部了。

但是实际现场情况是wdev地址是6b6b6b63，该地址以6b填充，属于use after free类型地址，所以怀疑是对链表进行遍历时，有其他的线程对链表节点进行删除操作，当遍历链表时，指向了已被删除的节点，导致出错，但是看代码上有加锁的操作，不清楚这里的原因

接着查看wdev的两个地址由slab分配，根据slab推出wdev两个地址的调用栈，未发现有6b或者bb填充的地址，怀疑可能是在链表在这两个地址之间跳转时，发生bit位翻转,导致跳转到某个被释放的未知地址

crash-arm32> kmem -S 0xe8e2d64c /\* 偏移8字节 \*/

CACHE NAME OBJSIZE ALLOCATED TOTAL SLABS SSIZE

ee802640 kmalloc-2048 2048 162 168 12 32k

SLAB MEMORY NODE TOTAL ALLOCATED FREE

ef516500 e8e28000 0 14 13 1

FREE / [ALLOCATED]

[e8e28000]

[e8e28900]

[e8e29200]

[e8e29b00]

[e8e2a400]

[e8e2ad00]

[e8e2b600]

[e8e2bf00]

[e8e2c800]

[e8e2d100]

[e8e2da00]

[e8e2e300]

[e8e2ec00]

[e8e2f500] /\* 这里列出了一组分配状态的slab的起始地址，看0xe8e2d64c地址落在哪个范围。 \*/

crash-arm32> p 0xe8e2d100-0xe8e2c800

$12 = 2304 /\* 计算得出这个slab共2304个字节 \*/

crash-arm32> p 2304/4 /\* 每组显示4个字节，共576组 \*/

$13 = 576

crash-arm32> rd e8e2d100 576

e8e2d100: cccccccc cccccccc cccccccc cccccccc ................

e8e2d110: cccccccc cccccccc cccccccc cccccccc ................

e8e2d120: cccccccc cccccccc cccccccc cccccccc ................

e8e2d130: cccccccc cccccccc cccccccc cccccccc ................

e8e2d140: 6e616c77 00000030 00000000 00000000 wlan0...........

e8e2d150: 00000000 ed41905c 00000000 00000000 ....\.A.........

e8e2d160: 00000000 00000000 00000000 00000000 ................

...

>e8e2d940: cccccccc e8e2da40 c08bd6d8 c0254720 ....@....... G%./\*cccc后面为调用栈信息\*/

>e8e2d950: c024ce84 c08bd6d8 bf09097c bf090c84 ..$.....|.......

>e8e2d960: bf09e88c c0549380 c0547200 c05474f0 ......T..rT..tT.

>e8e2d970: c0545328 c0546bc4 c05466c0 c054817c (ST..kT..fT.|.T.

>e8e2d980: c0549280 bf0aa018 c0101a08 00000000 ..T.............

crash-arm32> kmem -S 0xeb998074

CACHE NAME OBJSIZE ALLOCATED TOTAL SLABS SSIZE

ee802640 kmalloc-2048 2048 162 168 12 32k

SLAB MEMORY NODE TOTAL ALLOCATED FREE

ef56d300 eb998000 0 14 14 0

FREE / [ALLOCATED]

[eb998000]

[eb998900]

[eb999200]

[eb999b00]

[eb99a400]

[eb99ad00]

[eb99b600]

[eb99bf00]

[eb99c800]

[eb99d100]

[eb99da00]

[eb99e300]

[eb99ec00]

[eb99f500]

crash-arm32> p 0xeb998900-0xeb998000

$14 = 2304

crash-arm32> rd eb998000 576

eb998000: cccccccc cccccccc cccccccc cccccccc ................

eb998010: cccccccc cccccccc cccccccc cccccccc ................

eb998020: cccccccc cccccccc cccccccc cccccccc ................

eb998030: cccccccc cccccccc cccccccc cccccccc ................

eb998040: bf0a5140 c11a0ef0 c11a0ef0 00000000 @Q..............

eb998050: 00000000 c0a236b0 ec6ba840 ffffffe0 .....6..@.k.....

eb998060: eb998060 eb998060 c0a236e8 00000000 `...`....6......

eb998070: 00000000 00000000 00000000 e8e2d64c ............L...

...

>eb998840: cccccccc 00000000 c0a220c8 c0254720 ......... .. G%./\*ccccc后面为调用栈信息\*/

>eb998850: c024ce84 c0a220c8 bf094d94 bf09e874 ..$.. ...M..t...

>eb998860: c0549380 c0547200 c05474f0 c0545328 ..T..rT..tT.(ST.

>eb998870: c0546bc4 c05466c0 c054817c c0549280 .kT..fT.|.T...T.

eb998880: bf0aa018 c0101a08 c020749c 00000000 .........t .....

eb998890: 0000011d ffff96a4 bf01ddf0 c024e5c4 ..............$.

eb9988a0: bf01ddf0 bf01c5c4 c0268a50 c0268b8c ........P.&...&.

eb9988b0: c0108280 00000000 00000000 00000000 ................

eb9988c0: 00000000 00000000 00000000 00000000 ................

eb9988d0: 00000000 00000000 00000000 00000002 ................

eb9988e0: 0000019c ffff9655 5a5a5a5a 5a5a5a5a ....U...ZZZZZZZZ

eb9988f0: 5a5a5a5a 5a5a5a5a 5a5a5a5a 5a5a5a5a ZZZZZZZZZZZZZZZZ

//0xe8e2d644调用栈 /\*通过dis命令，解析调用栈\*/

\_\_\_slab\_alloc

\_\_slab\_alloc

\_\_kmalloc

alloc\_netdev\_mqs

sprdwl\_add\_iface

sprdwl\_core\_init

sprdwl\_probe

platform\_drv\_probe

driver\_probe\_device

\_\_driver\_attach

bus\_for\_each\_dev

//0xeb998074调用栈

\_\_\_slab\_alloc

\_\_slab\_alloc

\_\_kmalloc

wiphy\_new\_nm

sprdwl\_core\_create

sprdwl\_probe