# kfence

--Kernel Electric-Fence

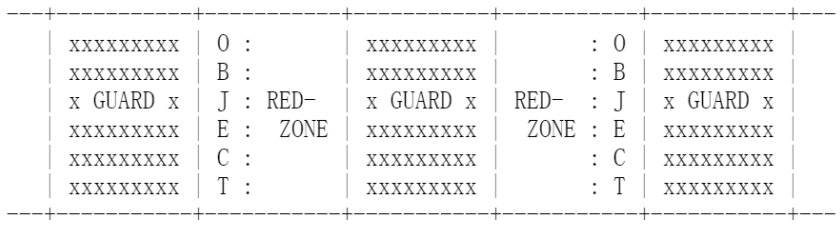
Kfence 是 kernel 5.12 引入的基于采样的低开销内存非法访问检测工具，可以检测 use-after-free，out-of-bounds access 和无效释放（double free）错误。设计的初衷是期望部署在生产环境中，对性能的影响要极小，但是相应的缺陷就是检测精度会比较低，需要大规模测试才能发现问题。

## 一、kfence 检测的原理

### 1.1__kfence_pool

Kfence 其实是在 slab 分配器内添加的一个内存池__kfence_pool。该内存池确保每个 slab object 相邻的内存页都是保护页，在 object 所在内存页的有效数据之外的地址全部设置为 redzone，以 KFENCE_CANARY_PATTERN 来填充。

CONFIG_KFENCE_NUM_OBJECT 定义了__kfence_pool 中 slab object 的数量，也就是有效数据占用的内存页数量。为了确保每个 object 相邻的页都是保护页，__kfence_pool 占用的内存页数量是(CONFIG_KFENCE_NUM_OBJECT+1)*2，其中第 0 页是额外添加的保护页，作用仅仅是因为开发者认为这样方便找到每个有效的 object（我是没理解奇数页和偶数页的查找有什么差别呢）




```
/*
 * Protect the first 2 pages. The first page is mostly unnecessary, and
 * merely serves as an extended guard page. However, adding one
 * additional page in the beginning gives us an even number of pages,
 * which simplifies the mapping of address to metadata index.
 */
```

下面介绍一下保护页和 redzone 是如何起作用的。

### 1.2 保护页

要理解保护页的作用需要先介绍内核页表的描述符。虚拟地址到物理地址的转换是 MMU 通过查找页表来实现的，48 位虚拟地址的页表描述符的具体格式可以看下图。物理页框的大小是 4KB 的话，bit[47:12]是下一级页表所在的页框号(pfn)，页框的地址就是页框号乘以 4096，也就是低 12 位全为 0。那么页表描述符低 12 位可以作为一下特殊用途，比如 bit0 为 0 时，表示无效的页表项；bit[1:0]为 01 时，表示 block 页表项，当 MMU 查找到 block 页表项或者最后一级页表项的时候就完成了页表的翻译；bit[1:0]为 11

时，表示 table 页表项，会继续往下一级页表检索。

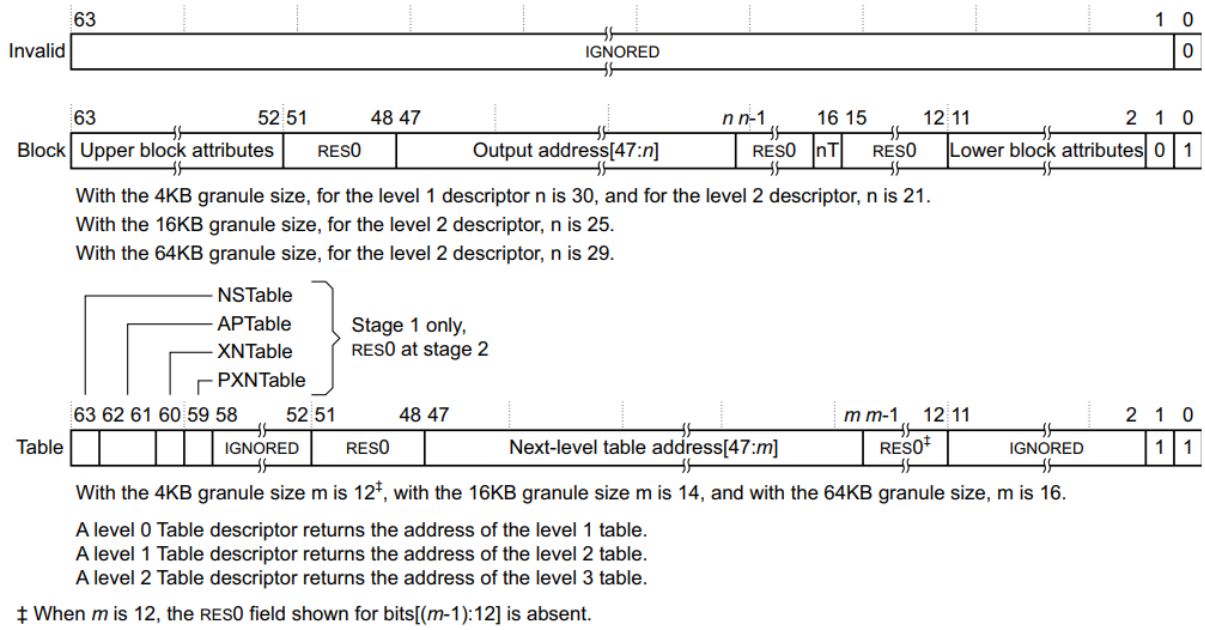



**Figure D5-15 VMSAv8-64 level 0, level 1 and level 2 descriptor formats with 48-bit OAs**

保护页就是把 pte 页表项的 bit0 置 1，这样如果发生越界访问到这一页中的地址的话，MMU 由于查找到无效的页表项，会触发缺页异常，在 kernel 的缺页异常处理流程中就可以抓到此时的调用栈。同样的对于 object 所在的内存页，释放之后会设置 pte 页表项为 invalid，申请的时候设置为 valid，这样在释放之后仍然访问的话也会被检测到。

**1.3redzone**

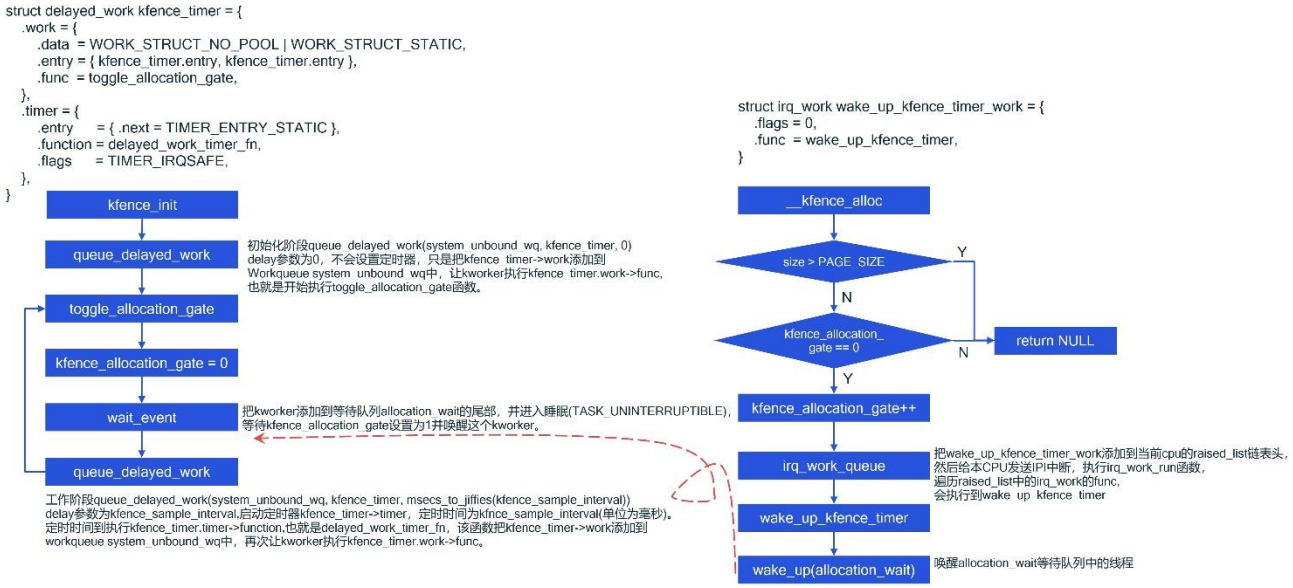
对于 object 所在的内存页，每次申请内存的时候，把有效数据长度之外的地址作为 redzone 全部填充为 KFENCE_CANARY_PATTERN(0xaa 异或地址的最低三位)。在释放内存的时候通过检查 redzone 中的值是否被破坏来判断有没有越界访问发生。如果检测到 redzone 的值被破坏，会打印出申请这段内存的调用栈，然后需要根据调用栈进一步分析使用者何时发生越界访问。但是 redzone 并不一定是被所在 page 的 object 的使用者破坏，如果某一驱动中越界的偏移地址大于 2 页，就有可能踩到了其他 object 的 redzone，而自己的 redzone 却是正常的，就无法被检测到。而其他 object 虽然检测到 redzone 被破坏，已经招不到破坏者是谁。所以 redzone 只对排查小于 1 页的越界访问有帮助。

## 二、采样

为了减小对性能的影响，kfence 引入了采样的机制，也就是不会对每次内存申请都进行保护。采样时间(t)可以通过 CONFIG_KFENCE_SAMPLE_INTERVAL 选项进行设置。如果 kfence 内存池还有空闲内存的话，距离上一次从 kfence 的内存池中分配内存间隔时间 t 之后，通过 slab 分配器分配的内存才会再次有机会从 kfence 内存池中获取内存。这

是在牺牲精度的基础上来换取极低的性能和额外内存开销，只有通过大规模测试才会发现内存问题。

采样的原理是基于内核的 delay work 机制，调用 queue_delayed_work 接口的时候（delay 参数不为 0）其实是启动定时器 kfence_timer->timer，定时时间为采样时间 t。定时时间到之后会执行 delayed_work_timer_fn，把 kfence_timer.work 添加到 workqueue events_unbound 中，然后 kworker 执行 kfence_timer.work->func，也就是 toggle_allocation_gate。函数 toggle_allocation_gate 首先会清零 kfence_allocation_gate 变量，表示 kfence 内存池可以使用，然后进入睡眠状态(TASK_UNINTERRUPTIBLE)，等到有线程申请 kfence 内存池的时候会设置 toggle_allocation_gate 为 1 并且唤醒 kworker 继续执行。被唤醒之后就再次调用 queue_delayed_work 设置定时器，进入循环。



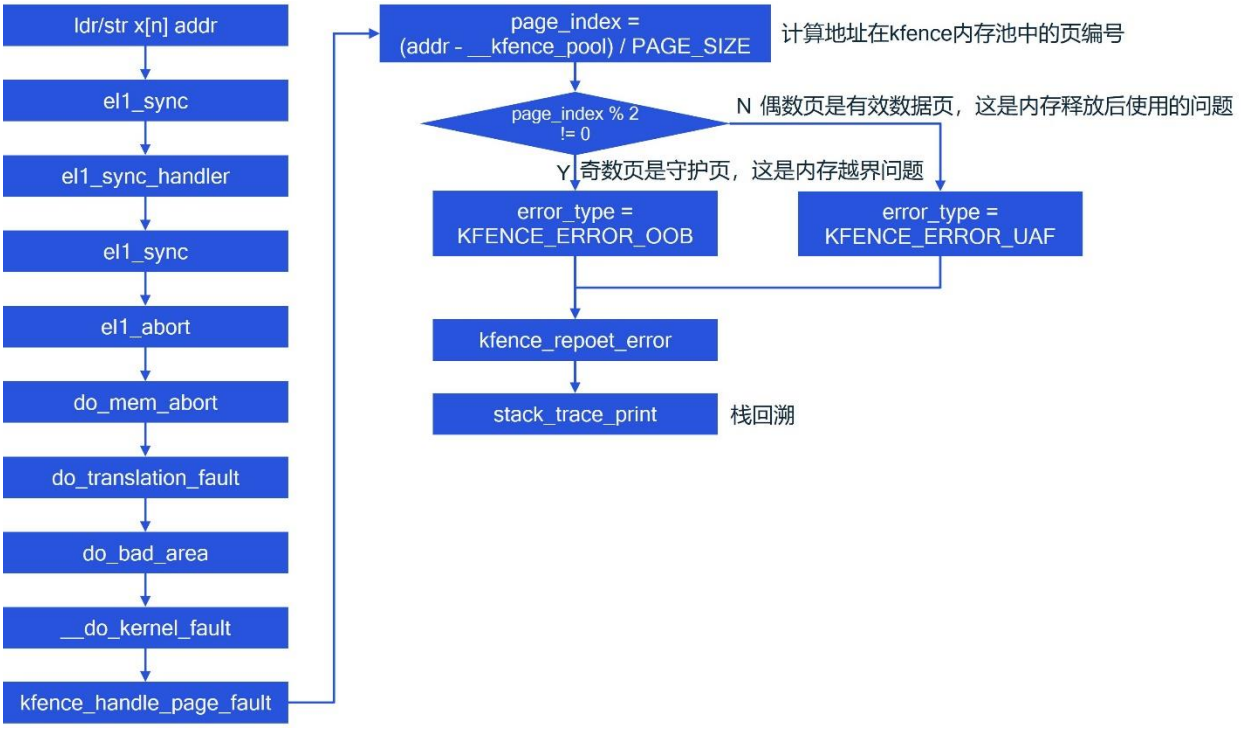

采样过程

## 三、报告 bug

如前所述，报告 bug 会通过两种方式，一是在访问到 guard page 的时候触发缺页异常，二是在释放内存的时候检测 redzone 是否被破坏。

### 3.1 缺页异常

对于守护页和已经释放的内存页，pte 页表项是无效的，当访问到其中的地址时会触发缺页异常。如果是守护页，说明是内存越界访问导致的；如果是已经释放的内存页，说明是释放后使用的问题。具体的流程见下图：




### 3.2redzone 被破坏

在每次释放 kfence 内存的时候会检测 redzone 的区域每一个地址上的值是否还是申请内存时候设置的 KFENCE_CANARY_PATTERN，如果不是，说明有内存越界访问发生。


## 四、示例

Use-after-free accesses are reported as:

```
==================================================================

BUG: KFENCE: use-after-free read in test_use_after_free_read+0xb3/0x143


Use-after-free read at 0xffffffffb673dfe0 (in kfence-#24):
 test_use_after_free_read+0xb3/0x143
 kunit_try_run_case+0x51/0x85
```

kunit_generic_run_threadfn_adapter+0x16/0x30
kthread+0x137/0x160
ret_from_fork+0x22/0x30

kfence-#24 [0xffffffffb673dfe0-0xffffffffb673dfff, size=32, cache=kmalloc-32] allocated by task 507:
test_alloc+0xf3/0x25b
test_use_after_free_read+0x76/0x143
kunit_try_run_case+0x51/0x85
kunit_generic_run_threadfn_adapter+0x16/0x30
kthread+0x137/0x160
ret_from_fork+0x22/0x30

freed by task 507:
test_use_after_free_read+0xa8/0x143
kunit_try_run_case+0x51/0x85
kunit_generic_run_threadfn_adapter+0x16/0x30
kthread+0x137/0x160
ret_from_fork+0x22/0x30

CPU: 4 PID: 109 Comm: kunit_try_catch Tainted: G W 5.8.0-rc6+ #7
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.13.0-1 04/01/2014

Reference:

https://opengrok.qualcomm.com/source/xref/KERNEL.PLATFORM.1.0/kernel_platform/common/mm/kfence/

https://opengrok.qualcomm.com/source/xref/KERNEL.PLATFORM.1.0/kernel_platform/msm-kernel/Documentation/dev-tools/kfence.rst