Ход анализа падения
«memory leak in drm_vma_node_allow»

Падение зафиксировано на одном из стендов syzkaller:

```
ioctl$sock_ipv6_tunnel_SIOCDELTUNNEL(r1, 0x89f2, &(0x7f0000000100)={'ip6_vti0\x00', &(0x7f00
BUG: memory leak
unreferenced object 0xffff88802f103680 (size 64):
  comm "syz-executor.2", pid 9675, jiffies 4295232585 (age 19.584s)
  hex dump (first 32 bytes):
    01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
    00 00 00 00 00 00 00 00 00 00 54 74 2b 80 88 ff ff  .........Tt+....
  backtrace:
    [<000000003d10fde5>] kmemleak_alloc_recursive include/linux/kmemleak.h:43 [inline]
    [<000000003d10fde5>] slab_post_alloc_hook mm/slab.h:534 [inline]
    [<000000003d10fde5>] slab_alloc_node mm/slub.c:2896 [inline]
    [<000000003d10fde5>] slab_alloc mm/slub.c:2904 [inline]
    [<000000003d10fde5>] kmem_cache_alloc_trace+0x10d/0x250 mm/slub.c:2921
    [<00000000a437de68>] drm_vma_node_allow+0x57/0x330 [drm]
    [<000000009d0a0f80>] drm_gem_handle_create_tail+0x143/0x3c0 [drm]
    [<00000000e01ef06b>] drm_gem_handle_create+0x58/0x70 [drm]
    [<0000000015b9f7a6>] drm_gem_vram_fill_create_dumb+0x1fa/0x4b0 [drm_vram_helper]
    [<00000000ea4eb8a1>] drm_gem_vram_driver_dumb_create+0x5d/0xc0 [drm_vram_helper]
    [<00000000d8dd2072>] drm_mode_create_dumb+0x290/0x320 [drm]
    [<0000000019a45a45>] drm_mode_create_dumb_ioctl+0x2b/0x40 [drm]
    [<00000000ba0603c9>] drm_ioctl_kernel+0x243/0x2f0 [drm]
    [<00000000f9dcda43>] drm_ioctl+0x59a/0xa90 [drm]
    [<00000000414efc5d>] vfs_ioctl fs/ioctl.c:48 [inline]
    [<00000000414efc5d>] __do_sys_ioctl fs/ioctl.c:753 [inline]
    [<00000000414efc5d>] __se_sys_ioctl fs/ioctl.c:739 [inline]
    [<00000000414efc5d>] __x64_sys_ioctl+0x19f/0x230 fs/ioctl.c:739
    [<000000001ca55010>] do_syscall_64+0x35/0x90 arch/x86/entry/common.c:46
    [<00000000294493e0>] entry_SYSCALL_64_after_hwframe+0x61/0xc6

BUG: leak checking failed
```

Рис. 1. Стек вызовов, приводящий к падению

Падение произошло на нашем стенде 2022-11-03, был получен репродюсер. В syzbot именно в этот же день был получен репродюсер, о чем syzbot сообщил, тем самым освежив историю по данной баге. Примечательно, что на нашем стенде репродюсер воспроизвелся на несколько часов раньше. В syzbot ошибка была выявлена еще в 2019 году: https://syzkaller.appspot.com/bug?extid=04639d98c75c52e41b8a. Исследователь Igor Torrente предлагал даже патч - https://syzkaller.appspot.com/text?tag=Patch&x=12ff84c4b00000, который, правда, приводил к segmentation fault (далее разберем почему).

2023-01-26 мы начали разбираться с причинами возникновения утечки. Суть кроется в неправильной работе с красно-черными деревьями. А именно – при попытке добавления нового элемента в дерево увеличивался счетчик ссылкой на этот элемент (чего делать по идее и не надо). А при

высвобождении памяти из-под элементов дерева – ее реальная очистка производилась только при счетчике == 1. Таким образом при добавлении двух и более элементов мы получаем утечку в размере занятого объема памяти первым элементом (т.к. для дубликатов память высвобождалась при переходе по goto unlock).

```
279
280         while (likely(*iter)) {
281                 parent = *iter;
282                 entry = rb_entry(*iter, struct drm_vma_offset_file, vm_rb);
283
284                 if (tag == entry->vm_tag) {
285                         entry->vm_count++;
286                         goto unlock;
287                 } else if (tag > entry->vm_tag) {
288                         iter = &(*iter)->rb_right;
289                 } else {
290                         iter = &(*iter)->rb_left;
291                 }
292         }
293
294         if (!new) {
295                 ret = -ENOMEM;
296                 goto unlock;
297         }
298
299         new->vm_tag = tag;
300         new->vm_count = 1;
301         rb_link_node(&new->vm_rb, parent, iter);
302         rb_insert_color(&new->vm_rb, &node->vm_files);
303         new = NULL;
304
305 unlock:
306         write_unlock(&node->vm_lock);
307         kfree(new);
308         return ret;
309 }
310 EXPORT_SYMBOL(drm_vma_node_allow);
311
312 /**
313  * drm_vma_node_revoke - Remove open-file from list of allowed users
314  * @node: Node to modify
315  * @tag: Tag of file to remove
316  *
317  * Decrement the ref-count of @tag in the list of allowed open-files on @node.
318  * If the ref-count drops to zero, remove @tag from the list. You must call
319  * this once for every drm_vma_node_allow() on @tag.
320  *
321  * This is locked against concurrent access internally.
322  *
323  * If @tag is not on the list, nothing is done.
324  */
325 void drm_vma_node_revoke(struct drm_vma_offset_node *node,
326                          struct drm_file *tag)
327 {
328         struct drm_vma_offset_file *entry;
329         struct rb_node *iter;
330
331         write_lock(&node->vm_lock);
332
333         iter = node->vm_files.rb_node;
334         while (likely(iter)) {
335                 entry = rb_entry(iter, struct drm_vma_offset_file, vm_rb);
336                 if (tag == entry->vm_tag) {
337                         if (!--entry->vm_count) {
338                                 rb_erase(&entry->vm_rb, &node->vm_files);
339                                 kfree(entry);
340                         }
341                         break;
342                 } else if (tag > entry->vm_tag) {
```

linux v5.10.142

Рис. 2. Исходный код анализируемой функции

Разберем теперь, почему же при предложенном Исследователем варианте патча, мы получаем segmentation fault. Если проанализировать начальный код, то он уже выглядит странным – сначала зануляем указатель

new, а двумя строчками ниже вызываем для него kfree. Не является напрямую ошибкой, kfree безопасная с этой точки зрения функция и сама проверит переданный ей указатель на NULL. Возможно, в этом и была задумка разработчика, чтобы меньше реализовывать меток и переходов по ним goto –функция kfree «сама решит, когда освобождать память». А когда этого не надо делать, просто зануляется указатель (ведь он действительно нам и не нужен, в дальнейшем выборка элементов из дерева осуществляется посредством макроса container_of, обернутого в rb_entry – подробнее можно почитать тут http://rflinux.blogspot.com/2011/10/red-black-trees-linux_16.html).

```
diff --git a/drivers/gpu/drm/drm_vma_manager.c b/drivers/gpu/drm/drm_vma_manager.c
index 7de37f8c68fd..f20f38ffb635 100644
--- a/drivers/gpu/drm/drm_vma_manager.c
+++ b/drivers/gpu/drm/drm_vma_manager.c
@@ -300,11 +300,11 @@ int drm_vma_node_allow(struct drm_vma_offset_node *node, struct drm_file *tag)
         new->vm_count = 1;
         rb_link_node(&new->vm_rb, parent, iter);
         rb_insert_color(&new->vm_rb, &node->vm_files);
-        new = NULL;
-
+
 unlock:
         write_unlock(&node->vm_lock);
         kfree(new);
+        new = NULL;
         return ret;
 }
 EXPORT_SYMBOL(drm_vma_node_allow);
```

Рис. 3. Вариант некорректного патча

Рассмотрим почему предложенный Исследователем вариант патча приводил к другой ошибке. Предложенная замена местами зануления и kfree возникает безусловное освобождение памяти из под только что добавленного элемента в дерево! Тем самым вызывая при попытке высвобождения памяти из под него (уже штатным образом) обращение по нулевому указателю (стр. 336 на Рис. 1).

Патч, устраняющий ошибку

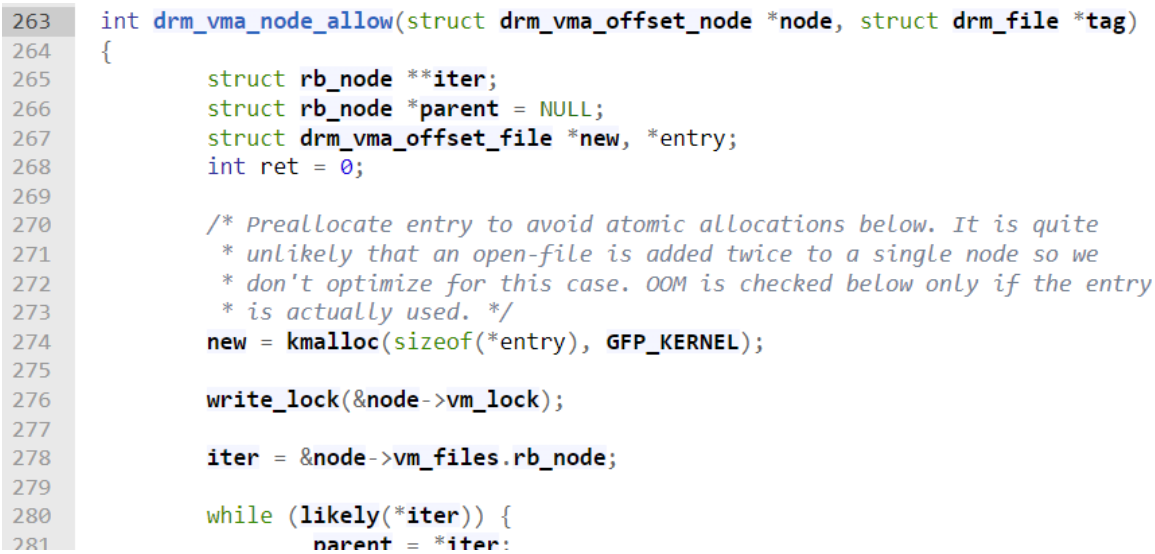
С ошибкой разобрались, но перед тем как готовить патч (а времени с начала анализа ошибки может пройти прилично), желательно еще раз посмотреть – не было ли, все-таки, какой-то активности по этому направлению, патчей для этой подсистемы, в этой или соседних функциях.

Поиски дают свои плоды - https://cgit.freedesktop.org/drm/drm/commit/?id=d23db89883962d9b4cb3ad03dfd02e525ed2cc03. Что мы видим – 2023-01-27 (вспоминаем, что мы начали разбираться с багой 2023-01-26, а первое упоминание о ней было аж в 2019 году) мантейнер подсистемы drm внес патч, устраняющий утечку памяти (правда до нее они дошли не через syzbot, а через тестирование отдельной составляющей подсистемы i915). Чуть-чуть мы не успели, но зато подтверждаются все наши предположения о месте ошибки, вызывающей утечку памяти.

Таким образом, остается проследить, чтобы этот патч вошел в ветку ядра, и не только в апстрим, но и 5.10, 5.15. Здесь надо проработать вопрос – какие действия нам стоит предпринять, какую тактику выбрать?

Сопутствующие дополнения

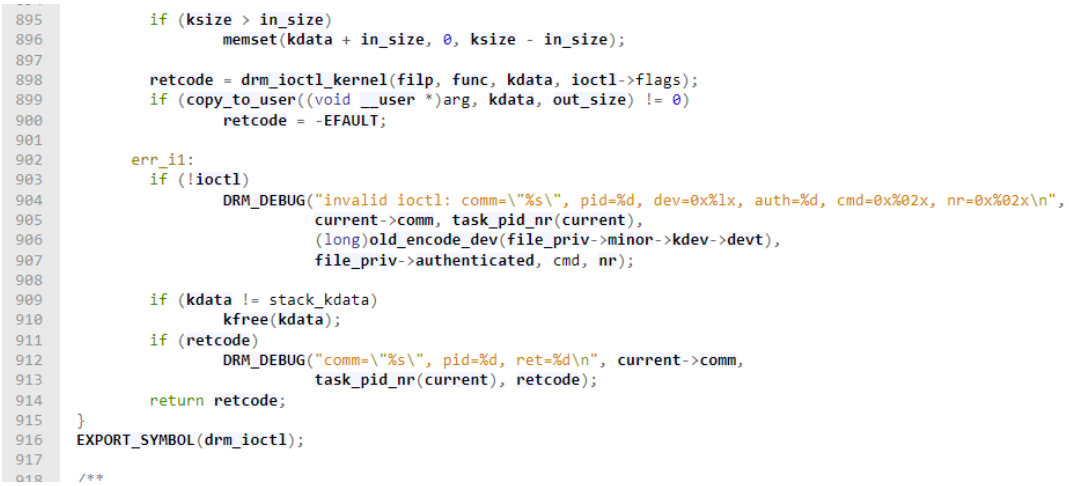
Дополнение 1. Но и на этом, как кажется, не все. Мы же за чистоту кода – видится целесообразным предложить патч с устранением «красивой» ситуации с NULL и kfree. Ведь по факту kfree реально нужен лишь для высвобождения памяти, в случае, если элемент уже найден в списке. Т.е правильнее выделение памяти перенести ниже успешного завершения цикла while, что будет означать необходимость добавления нового элемента в дерево. Тогда и комментариев на строках 270-273 (см. рис. 4) не надо будет делать, да и код будет выглядеть логичнее – когда проверка на корректность выделения памяти идет сразу после выполнения соответствующей операции. Да, придется добавить еще один маркер unlock_free, заменить new=NULL на return ret (ну примерно так) после разблокировки узла.



```
263  int drm_vma_node_allow(struct drm_vma_offset_node *node, struct drm_file *tag)
264  {
265          struct rb_node **iter;
266          struct rb_node *parent = NULL;
267          struct drm_vma_offset_file *new, *entry;
268          int ret = 0;
269
270          /* Preallocate entry to avoid atomic allocations below. It is quite
271           * unlikely that an open-file is added twice to a single node so we
272           * don't optimize for this case. OOM is checked below only if the entry
273           * is actually used. */
274          new = kmalloc(sizeof(*entry), GFP_KERNEL);
275
276          write_lock(&node->vm_lock);
277
278          iter = &node->vm_files.rb_node;
279
280          while (likely(*iter)) {
281                  parent = *iter;
```

Рис. 4. Комментарий о нецелесообразности проверки корректности выделенной памяти сразу после выполнения соответствующей функции

Дополнение 2. Трасса вызовов, приводящая к анализируемой ошибке, содержит вызов drm_ioctl, где в строке 898 вызывается drm_ioctl_kernel, в которой при успехе и добавляется новый элемент в дерево. Но вспоминаем, что вызываемые функции могут выполниться и с ошибкой, например из-за отсутствия памяти. Да и сама функция drm_ioctl_kernel предполагает возврат кода ошибки, проверка которого благополучно игнорируется. Это может привести к тому, что в стр. 900 код ошибки перезатрется или останется равным EINVAL (как инициализирован изначально) и вместо ENOMEM вернется EINVAL, что будет записано в логи на стр. 912. Не критично, но диагностику ошибки точно затруднит!



```
895          if (ksize > in_size)
896                  memset(kdata + in_size, 0, ksize - in_size);
897
898          retcode = drm_ioctl_kernel(filp, func, kdata, ioctl->flags);
899          if (copy_to_user((void __user *)arg, kdata, out_size) != 0)
900                  retcode = -EFAULT;
901
902        err_i1:
903          if (!ioctl)
904                  DRM_DEBUG("invalid ioctl: comm=\"%s\", pid=%d, dev=0x%lx, auth=%d, cmd=0x%02x, nr=0x%02x\n",
905                            current->comm, task_pid_nr(current),
906                            (long)old_encode_dev(file_priv->minor->kdev->devt),
907                            file_priv->authenticated, cmd, nr);
908
909          if (kdata != stack_kdata)
910                  kfree(kdata);
911          if (retcode)
912                  DRM_DEBUG("comm=\"%s\", pid=%d, ret=%d\n", current->comm,
913                            task_pid_nr(current), retcode);
914          return retcode;
915  }
916  EXPORT_SYMBOL(drm_ioctl);
917
```