Chasing Page Pool Leaks

… with drgns

Dragoș Tătulea,

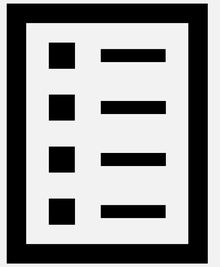
NetDev 0x19

March 2025,

Zagreb - Croatia

Agenda

- Scope and Motivation
- Introduction
- Toolbox
- Examples / Tutorial



Scope

What is this talk about?

- I have a page pool leak, is this a bug or a false positive?
- Where is it coming from?

Motivation

- Encountered quite a few of these leaks.
- Discussed on netdev mailing list as well.
- Wanted to learn *drgn*.

```
From: Jakub Kicinski <kuba@kernel.org>
To: Yonglong Liu <liuyonglong@huawei.com>
Cc: Yunsheng Lin <linyunsheng@huawei.com>, <netdev@vger.kernel.org>,
        <davem@davemloft.net>, <edumazet@google.com>, <pabeni@redhat.com>,
        <ilias.apalodimas@linaro.org>,
        Jesper Dangaard Brouer <hawk@kernel.org>,
        Alexander Duyck <alexander.duyck@gmail.com>
Subject: Re: [RFC net] net: make page pool stall netdev unregistration to avoid IOMMU crashes
Date: Wed, 14 Aug 2024 07:56:03 -0700   [thread overview]
Message-ID: <20240814075603.05f8b0f5@kernel.org> (raw)
In-Reply-To: <cec044dc-504c-47e6-8ffa-58e8c9b42713@huawei.com>

On Wed, 14 Aug 2024 18:09:59 +0800 Yonglong Liu wrote:
> On 2024/8/10 11:57, Jakub Kicinski wrote:
> > On Fri, 9 Aug 2024 14:06:02 +0800 Yonglong Liu wrote:
> >> [ 7724.272853] hns3 0000:7d:01.0: page_pool_release_retry(): eno1v0
> >> stalled pool shutdown: id 553, 82 inflight 6706 sec (hold netdev: 1855491)
> > Alright :( You gotta look around for those 82 pages somehow with drgn.
> > bpftrace+kfunc the work that does the periodic print to get the address
> > of the page pool struct and then look around for pages from that pp.. :(
>
> I spent some time to learn how to use the drgn, and found those page,
> but I think those page
>
> is allocated by the hns3 driver, how to find out who own those page now?

Scan the entire system memory looking for the pointer to this page.
Dump the memory around location which hold that pointer. If you're
lucky the page will be held by an skb, and the memory around it will
look like struct skb_shared_info. If you're less lucky the page is used
by sk_buff for the head and address will not be exact. If you're less
lucky still the page will be directly leaked by the driver, and not
pointed to by anything...

I think the last case is most likely, FWIW.
```

link

Introduction

- What is a page pool?
- What is page inflight state?
- When is a leak?
- Is could be a:
  - Bug: miscounting of resources.
  - False positive: page still referenced for a valid reason.
- *_refcount* vs *pp_ref_count*

```
struct page {
        unsigned long flags;            /* Atomic flags, some possibly
                                         * updated asynchronously */
        /*
         * Five words (20/40 bytes) are available in this union.
         * WARNING: bit 0 of the first word is used for PageTail(). That
         * means the other users of this union MUST NOT use the bit to
         * avoid collision and false-positive PageTail().
         */
        union {
```

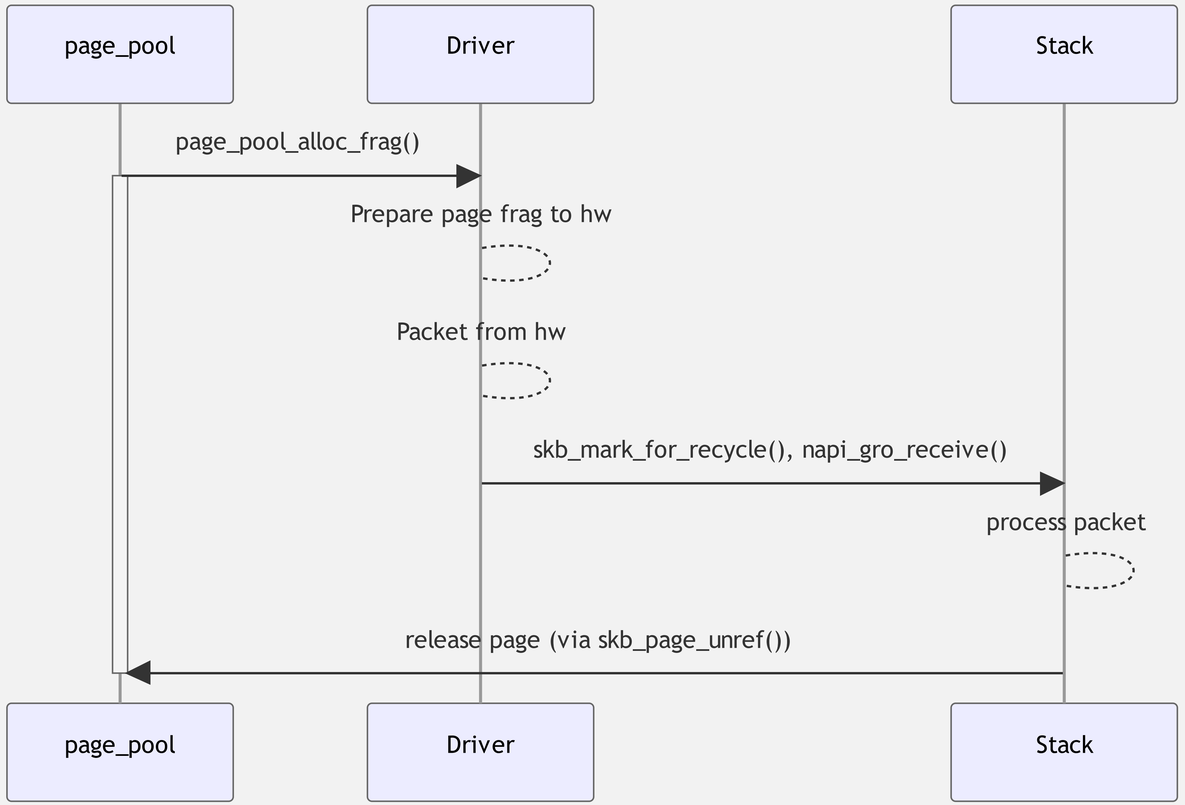
```
  struct {        /* page_pool used by netstack */
          /**
           * @pp_magic: magic value to avoid recycling non
           * page_pool allocated pages.
           */
          unsigned long pp_magic;
          struct page_pool *pp;
          unsigned long _pp_mapping_pad;
          unsigned long dma_addr;
          atomic_long_t pp_ref_count;
  };
```

```
/* Usage count. *DO NOT USE DIRECTLY*. See page_ref.h */
atomic_t _refcount;
```

Inflight Page Lifetime

… with variations

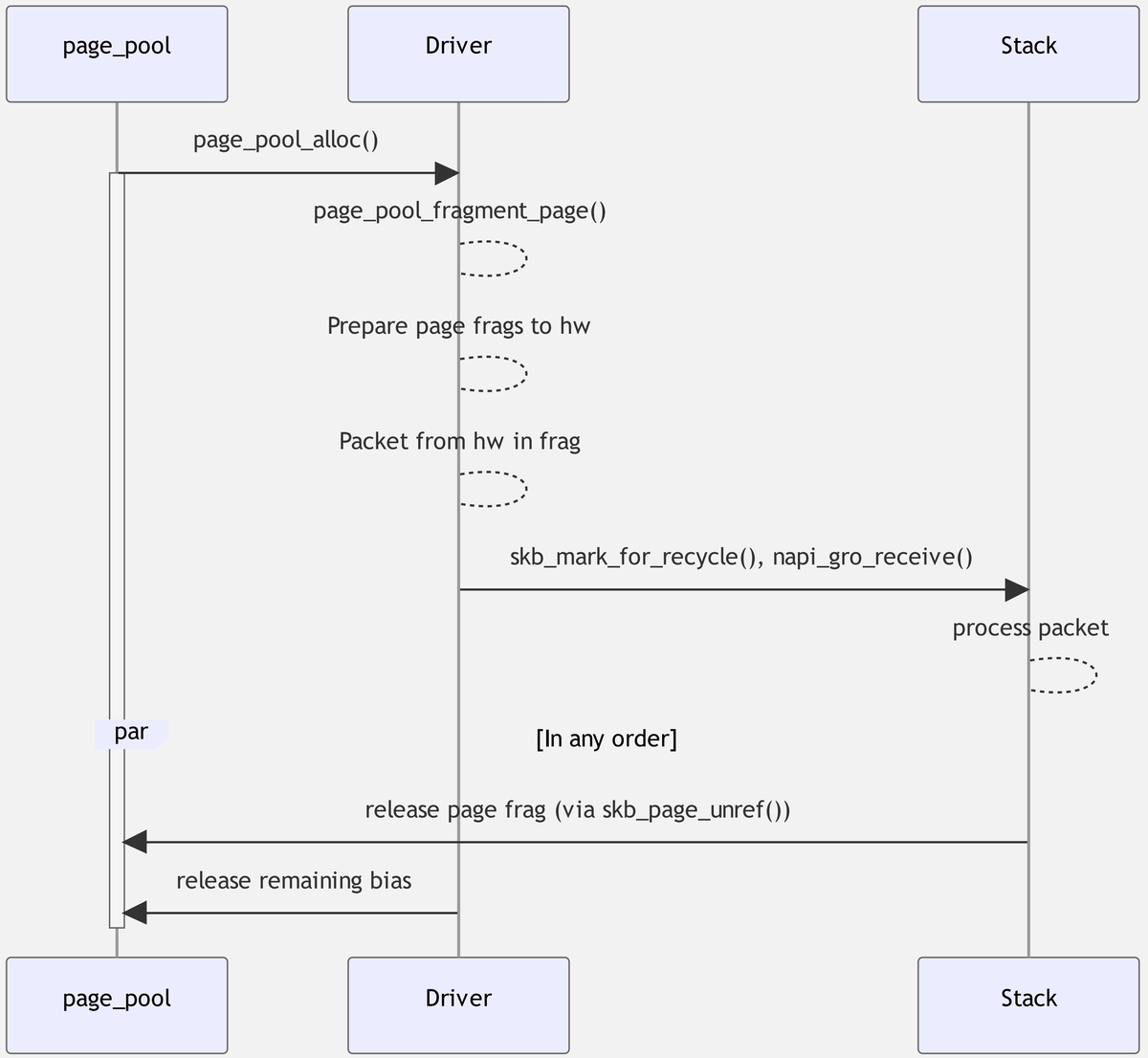
- Full page.
- Page fragment:
  - Size known in advance



Inflight Page Lifetime

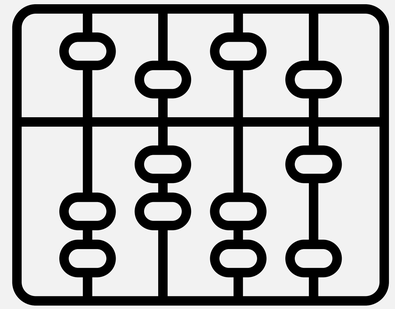
with bias

- Page fragment
  - Size not known in advance
- 2 step release
- How about queue teardown?



Types of Leaks

- Driver side
  - Incorrect refcounting at runtime
  - Incorrect refcounting at teardown
- System side
  - False positive
  - Bug



Toolbox

- The *drgn* debugger
- scripts
  - tcp_sock.py
    - Based on contrib/tcp_sock.py.
  - ls_pp_leaks.py
  - find.py
    - Based on contrib/search_kernel_memory.py

Let’s begin

Example 1

- A leak was found …
- Where does it come from?

```
+ /w/linux/tools/net/ynl/cli.py --no-schema --spec /w/linux/Documentation/netlink/specs/netdev.yaml --dump page-pool-get
[{'detach-time': 149,
  'id': 3,
  'ifindex': 3,
  'inflight': 1,
  'inflight-mem': 4096}]
```

Example 1

Variation 1

- Search sockets for SKBs with leaked page_pool pages.
- Use *tcp_sock.py*.

tcp_sock.py

```
PP_SIGNATURE = 0xdead000000000040

tcp_hashinfo = prog.object("tcp_hashinfo")

for i in range(tcp_hashinfo.ehash_mask + 1):
    head = tcp_hashinfo.ehash[i].chain
    if hlist_nulls_empty(head):
        continue

    for sk in sk_nulls_for_each(head):

        # Filter by interface:
        if ifindex > 0 and sk.sk_rx_dst_ifindex.value_() != ifindex:
            continue

        first_skb = sk.sk_receive_queue.next
        skb = first_skb
        while skb != None:

            try:
                # Check linear part of skb:
                page = virt_to_page(skb.data)
                if page.pp_magic == PP_SIGNATURE and page.pp.user.detach_time:
                    print(f"Found leaked page {hex(page)} in linear part of  skb: {hex(skb.address_of_())}")

                # Check fragments:
                shinfo = skb_shinfo(skb)
                for i in range(0, shinfo.nr_frags):
                    frag = shinfo.frags[i]
                    page = Object(prog, "struct page", address=frag.netmem)
                    if page.pp_magic == PP_SIGNATURE and page.pp.user.detach_time:
                        print(f"Found leaked page {hex(page.address_of_())} in skb frag {i} of skb: {hex(skb.address_of_())}")

            except FaultError:
                continue

            # Move to next skb:
            skb = skb.next
            if skb == first_skb:
                break
```

```
Found leaked page 0xffffea000016ab40 in linear part of  skb: 0xffff88800d7689e8
```

Example 1

Variation 1

- Check “struct page”
- Peek in page.

```
>>> Object(prog, "struct page", address=0xffffea000016ab40)
(struct page){
        .flags = (unsigned long)36028797018963968,
        .lru = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800b1ac800,
        },
        .__filler = (void *)0xdead000000000040,
        .mlock_count = (unsigned int)186304512,
        .buddy_list = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800b1ac800,
        },
        .pcp_list = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800b1ac800,
        },
        .mapping = (struct address_space *)0x0,
        .index = (unsigned long)95080448,
        .share = (unsigned long)95080448,
        .private = (unsigned long)1,
        .pp_magic = (unsigned long)16045481047390945344,
        .pp = (struct page_pool *)0xffff88800b1ac800,
        ._pp_mapping_pad = (unsigned long)0,
        .dma_addr = (unsigned long)95080448,
        .pp_ref_count = (atomic_long_t){
                .counter = (s64)1,
        },
        .compound_head = (unsigned long)16045481047390945344,
        .pgmap = (struct dev_pagemap *)0xdead000000000040,
        .zone_device_data = (void *)0xffff88800b1ac800,
        .callback_head = (struct callback_head){
                .next = (struct callback_head *)0xdead000000000040,
                .func = (void (*)(struct callback_head *))0xffff88800b1ac800,
        },
        .page_type = (unsigned int)4294967295,
        ._mapcount = (atomic_t){
                .counter = (int)-1,
        },
        ._refcount = (atomic_t){
                .counter = (int)1,
        },
        .memcg_data = (unsigned long)0,
}
```

```
>>> print_annotated_memory(page_to_virt(page.address_of_()), 128)
ADDRESS           VALUE
ffff888005aad000: 0000000000000000
ffff888005aad008: 0000000000000000
ffff888005aad010: 0000000000000000
ffff888005aad018: 0000000000000000
ffff888005aad020: 0000000000000000
ffff888005aad028: 0000000000000000
ffff888005aad030: 0000000000000000
ffff888005aad038: 0000000000000000
ffff888005aad040: 79fe573412005452
ffff888005aad048: 0045000853158c69
ffff888005aad050: 06400040e6bb5200
ffff888005aad058: 01010a010101b37a
ffff888005aad060: 1840393038d40101
ffff888005aad068: 1880944e2a81aef4
ffff888005aad070: 010100002b78f601
ffff888005aad078: 7e05558744e50a08
```

Example 1

Variation 1

- Let’s look at the skb.

```
>>> Object(prog, "struct sk_buff *", address=0xffff88800d7689e8)
*(struct sk_buff *)0xffff888004fdb000 = {
        .next = (struct sk_buff *)0xffff88800d7689e8,
        .prev = (struct sk_buff *)0xffff88800d7689e8,
        .dev = (struct net_device *)0x0,
        .dev_scratch = (unsigned long)0,
        .rbnode = (struct rb_node){
                .__rb_parent_color = (unsigned long)18446612682295904744,
                .rb_right = (struct rb_node *)0xffff88800d7689e8,
                .rb_left = (struct rb_node *)0x0,
        },
        .list = (struct list_head){
                .next = (struct list_head *)0xffff88800d7689e8,
                .prev = (struct list_head *)0xffff88800d7689e8,
        },
        .ll_node = (struct llist_node){
                .next = (struct llist_node *)0xffff88800d7689e8,
        },
        .sk = (struct sock *)0xffff88800d768940,
        .tstamp = (ktime_t)0,
        .skb_mstamp_ns = (u64)0,
        .cb = (char [48])"\xae\xf4\x18@\xcc\xf4\x18@",
        ._skb_refdst = (unsigned long)0,
        .destructor = (void (*)(struct sk_buff *))sock_rfree+0x0 = 0xffffffff81902740,
        .tcp_tsorted_anchor = (struct list_head){
                .next = (struct list_head *)0x0,
                .prev = (struct list_head *)sock_rfree+0x0 = 0xffffffff81902740,
        },
        ._sk_redir = (unsigned long)0,
        ._nfct = (unsigned long)0,
        .len = (unsigned int)30,
        .data_len = (unsigned int)0,
        .mac_len = (__u16)14,
        .hdr_len = (__u16)0,
        .queue_mapping = (__u16)1,
        .__cloned_offset = (__u8 [0]){},
        .cloned = (__u8)0,
        .nohdr = (__u8)0,
        .fclone = (__u8)0,
        .peeked = (__u8)0,
        .head_frag = (__u8)1,
        .pfmemalloc = (__u8)0,
        .pp_recycle = (__u8)1,
        .active_extensions = (__u8)0,
        .__pkt_type_offset = (__u8 [0]){},
        .pkt_type = (__u8)0,
```

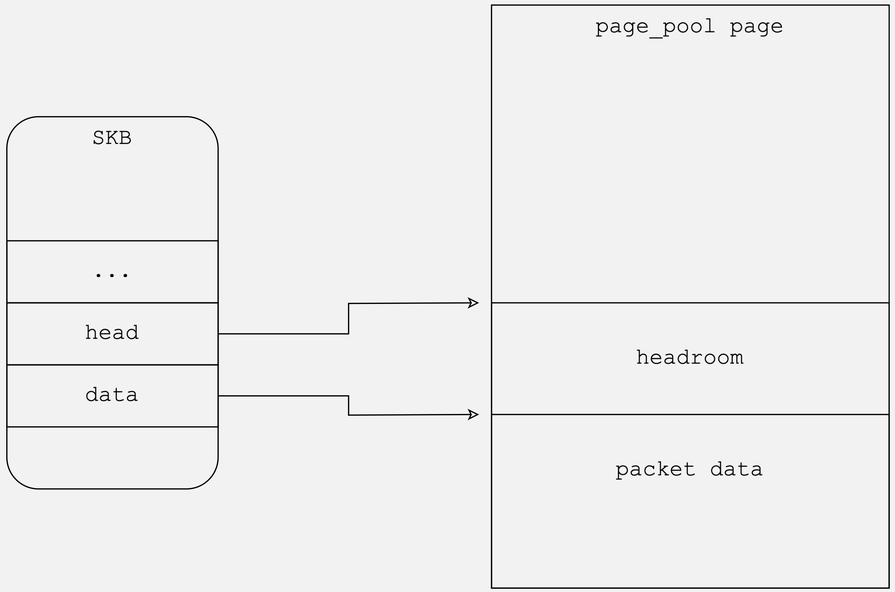
```
        .tail = (sk_buff_data_t)160,
        .end = (sk_buff_data_t)192,
        .head = (unsigned char *)0xffff888005aad000 = "",
        .data = (unsigned char *)0xffff888005aad082 = "more data.....................0",
        .truesize = (unsigned int)768,
        .users = (refcount_t){
                .refs = (atomic_t){
                        .counter = (int)1,
                },
        },
        .extensions = (struct skb_ext *)0x0,
}
```

Example 1

What was found

- Leaked page in SKB linear area.
- Direction: Socket -> SKB -> Page
- … let’s try the other way around:

  Page -> SKB -> Socket



Example 1

Variation 2

- Scan all leaked page pool pages.

ls_pp_leaks.py

```
PP_SIGNATURE = 0xdead000000000040

for page in for_each_page():
        try:
            if page.pp_magic == PP_SIGNATURE and page.pp.user.detach_time:
                print(page)
                print_annotated_memory(page_to_virt(page), 128)
        except FaultError:
            continue
```

```
Leaked page: 0xffffea000016ab40
Page content:
ADDRESS            VALUE
ffff888005aad000: 0000000000000000
ffff888005aad008: 0000000000000000
ffff888005aad010: 0000000000000000
ffff888005aad018: 0000000000000000
ffff888005aad020: 0000000000000000
ffff888005aad028: 0000000000000000
ffff888005aad030: 0000000000000000
ffff888005aad038: 0000000000000000
ffff888005aad040: 79fe573412005452
ffff888005aad048: 0045000853158c69
ffff888005aad050: 06400040e6bb5200
ffff888005aad058: 01010a010101b37a
ffff888005aad060: 1840393038d40101
```

Example 1

Variation 2

- This looks familiar...

```
>>> Object(prog, "struct page", address=0xffffea000016ab40)
(struct page){
        .flags = (unsigned long)36028797018963968,
        .lru = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800b1ac800,
        },
        .__filler = (void *)0xdead000000000040,
        .mlock_count = (unsigned int)186304512,
        .buddy_list = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800b1ac800,
        },
        .pcp_list = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800b1ac800,
        },
        .mapping = (struct address_space *)0x0,
        .index = (unsigned long)95080448,
        .share = (unsigned long)95080448,
        .private = (unsigned long)1,
        .pp_magic = (unsigned long)16045481047390945344,
        .pp = (struct page_pool *)0xffff88800b1ac800,
        ._pp_mapping_pad = (unsigned long)0,
        .dma_addr = (unsigned long)95080448,
        .pp_ref_count = (atomic_long_t){
                .counter = (s64)1,
        },
        .compound_head = (unsigned long)16045481047390945344,
        .pgmap = (struct dev_pagemap *)0xdead000000000040,
        .zone_device_data = (void *)0xffff88800b1ac800,
        .callback_head = (struct callback_head){
                .next = (struct callback_head *)0xdead000000000040,
                .func = (void (*)(struct callback_head *))0xffff88800b1ac800,
        },
        .page_type = (unsigned int)4294967295,
        ._mapcount = (atomic_t){
                .counter = (int)-1,
        },
        ._refcount = (atomic_t){
                .counter = (int)1,
        },
        .memcg_data = (unsigned long)0,
}
```

Example 1

Variation 2

- Find SKBs referencing address in page range.
- Search for most significant 6 bytes.
- Check next 4 bits.
- Once found, check that address is within page range.

snippet of find.py

```python
def search_page_reference(page):

    val = page_to_virt(page).value_()

    skip_bytes = math.ceil(PAGE_SHIFT / 8)
    ptr_size = 8

    val_endian = val.to_bytes(ptr_size, byteorder)
    if byteorder == "little":
        big_needle = val_endian[skip_bytes:ptr_size - skip_bytes]
    else:
        big_needle = val_endian[0:ptr_size - skip_bytes]

    small_needle = val >> PAGE_SHIFT

    results = []

    # Search for first 6 bytes:
    for addr in search_memory(prog, big_needle):

        if byteorder == "little":
            # Adjust address to skipped bytes:
            addr = addr - skip_bytes

        mem_bytes = prog.read(addr, ptr_size)
        mem_val = int.from_bytes(mem_bytes, byteorder)

        if mem_val >> PAGE_SHIFT == small_needle:
            results.append((addr, mem_val))

    return results
```

Example 1

Variation 2

- Found something.
  - Probably much more.
- Interpret as SKB.

```
Found reference at 0xffff888004fdb0c0. slab object: skbuff_head_cache+0xc0
ADDRESS           VALUE
ffff888004fdb000: ffff88800d7689e8 [slab object: TCP+0xa8]
ffff888004fdb008: ffff88800d7689e8 [slab object: TCP+0xa8]
ffff888004fdb010: 0000000000000000
ffff888004fdb018: ffff88800d768940 [slab object: TCP+0x0]
ffff888004fdb020: 0000000000000000
ffff888004fdb028: 4018f4cc4018f4ae
ffff888004fdb030: 0000001800000000
ffff888004fdb038: 00000000812a4e94
ffff888004fdb040: 0000000000000003
ffff888004fdb048: 0000000000000000
ffff888004fdb050: 0000000000000000
ffff888004fdb058: 0000000000000000
ffff888004fdb060: ffffffff81902740 [function symbol: sock_rfree+0x0]
ffff888004fdb068: 0000000000000000
ffff888004fdb070: 000000000000001e
ffff888004fdb078: 00a000010000000e
ffff888004fdb080: 0000000010420040
ffff888004fdb088: 000000000000aefb
ffff888004fdb090: ec80bfc300000003
ffff888004fdb098: 0000020200000000
ffff888004fdb0a0: 0000000000000000
ffff888004fdb0a8: 0000000000000000
ffff888004fdb0b0: 0040004e00620008
ffff888004fdb0b8: 000000c0000000a0
ffff888004fdb0c0: ffff888005aad000 [page offset in page 0xffffea000016ab40 page_pool id: 3, dev eth1, detached: yes]
ffff888004fdb0c8: ffff888005aad082 [page offset in page 0xffffea000016ab40 page_pool id: 3, dev eth1, detached: yes]
```

Example 1

Variation 2

- Interpret as SKB.
- This also looks familiar!

```
>>> skb = Object(prog, "struct sk_buff", address=0xffff888004fdb000)
>>> skb
(struct sk_buff){
        .next = (struct sk_buff *)0xffff88800d7689e8,
        .prev = (struct sk_buff *)0xffff88800d7689e8,
        .dev = (struct net_device *)0x0,
        .dev_scratch = (unsigned long)0,
        .rbnode = (struct rb_node){
                .__rb_parent_color = (unsigned long)18446612682295904744,
                .rb_right = (struct rb_node *)0xffff88800d7689e8,
                .rb_left = (struct rb_node *)0x0,
        },
        .list = (struct list_head){
                .next = (struct list_head *)0xffff88800d7689e8,
                .prev = (struct list_head *)0xffff88800d7689e8,
        },
        .ll_node = (struct llist_node){
                .next = (struct llist_node *)0xffff88800d7689e8,
        },
        .sk = (struct sock *)0xffff88800d768940,
        .tstamp = (ktime_t)0,
        .skb_mstamp_ns = (u64)0,
        .cb = (char [48])"\xae\xf4\x18@\xcc\xf4\x18@",
        ._skb_refdst = (unsigned long)0,
        .destructor = (void (*)(struct sk_buff *))sock_rfree+0x0 = 0xffffffff81902740,
        .tcp_tsorted_anchor = (struct list_head){
                .next = (struct list_head *)0x0,
                .prev = (struct list_head *)sock_rfree+0x0 = 0xffffffff81902740,
        },
        ._sk_redir = (unsigned long)0,
        ._nfct = (unsigned long)0,
        .len = (unsigned int)30,
        .data_len = (unsigned int)0,
        .mac_len = (__u16)14,
        .hdr_len = (__u16)0,
        .queue_mapping = (__u16)1,
        .__cloned_offset = (__u8 [0]){},
        .cloned = (__u8)0,
        .nohdr = (__u8)0,
        .fclone = (__u8)0,
        .peeked = (__u8)0,
        .head_frag = (__u8)1,
        .pfmemalloc = (__u8)0,
        .pp_recycle = (__u8)1,
        .active_extensions = (__u8)0,
        .__pkt_type_offset = (__u8 [0]){},
        .pkt_type = (__u8)0,
```

```
        .end = (sk_buff_data_t)192,
        .head = (unsigned char *)0xffff888005aad000 = "",
        .data = (unsigned char *)0xffff888005aad082 = "more data...................0",
        .truesize = (unsigned int)768,
        .users = (refcount_t){
                .refs = (atomic_t){
                        .counter = (int)1,
                },
        },
        .extensions = (struct skb_ext *)0x0,
}
```

Example 1

Conclusions

- Found leaking page.
- Was in linear part of SKB.
- pp_ref_count = 1
- Diagnosis: false positive

Example 2

- A new day, a new leak.
- Repeat steps…

```
+ /w/linux/tools/net/ynl/cli.py --no-schema --spec /w/linux/Documentation/netlink/specs/netdev.yaml --dump page-pool-get
[{'detach-time': 431,
  'id': 3,
  'ifindex': 3,
  'inflight': 2,
  'inflight-mem': 8192}]
```

Example 2

Variation 1

- Search sockets for SKBs with leaked page_pool pages.
- This time pages are frags…

From tcp_sock.py

```
Found leaked page 0xffffea0000495940 in skb frag 0 of skb: 0xffff888005dc1328
Found leaked page 0xffffea0000495900 in skb frag 1 of skb: 0xffff888005dc1328
```

Example 2

Variation 1

- Let’s peek into the SKB.
- Linear part not a page_pool page.

```
>>> skb = Object(prog, "struct sk_buff *", address=0xffff888005dc1328)
>>> skb
*(struct sk_buff *)0xffff888005ba8800 = {
        .next = (struct sk_buff *)0xffff888005dc1328,
        .prev = (struct sk_buff *)0xffff888005dc1328,
        .dev = (struct net_device *)0x0,
        .dev_scratch = (unsigned long)0,
        .rbnode = (struct rb_node){
                .__rb_parent_color = (unsigned long)18446612682168341288,
                .rb_right = (struct rb_node *)0xffff888005dc1328,
                .rb_left = (struct rb_node *)0x0,
        },
        .list = (struct list_head){
                .next = (struct list_head *)0xffff888005dc1328,
                .prev = (struct list_head *)0xffff888005dc1328,
        },
        .ll_node = (struct llist_node){
                .next = (struct llist_node *)0xffff888005dc1328,
        },
        .sk = (struct sock *)0xffff888005dc1280,
        .tstamp = (ktime_t)0,
        .skb_mstamp_ns = (u64)0,
        .cb = (char [48])"\x8a\x1d\\\x81\x1c1\\\x81",
        ._skb_refdst = (unsigned long)0,
        .destructor = (void (*)(struct sk_buff *))sock_rfree+0x0 = 0xffffffff81902740,
        .tcp_tsorted_anchor = (struct list_head){
                .next = (struct list_head *)0x0,
                .prev = (struct list_head *)sock_rfree+0x0 = 0xffffffff81902740,
        },
        ._sk_redir = (unsigned long)0,
        ._nfct = (unsigned long)0,
        .len = (unsigned int)5010,
        .data_len = (unsigned int)4820,
        .mac_len = (__u16)14,
        .hdr_len = (__u16)0,
        .queue_mapping = (__u16)1,
        .__cloned_offset = (__u8 [0]){},
        .cloned = (__u8)0,
        .nohdr = (__u8)0,
        .fclone = (__u8)0,
        .peeked = (__u8)0,
        .head_frag = (__u8)1,
        .pfmemalloc = (__u8)0,
        .pp_recycle = (__u8)1,
        .active_extensions = (__u8)0,
        .__pkt_type_offset = (__u8 [0]){},
        .pkt_type = (__u8)0,
```

```
        .tail = (sk_buff_data_t)320,
        .end = (sk_buff_data_t)704,
        .head = (unsigned char *)0xffff888005b99000 = "",
        .data = (unsigned char *)0xffff888005b99082 = "more data.....................
.....................................................................................
        .truesize = (unsigned int)6144,
        .users = (refcount_t){
                .refs = (atomic_t){
                        .counter = (int)1,
                },
        },
        .extensions = (struct skb_ext *)0x0,
}
```

Example 2

Variation 1

- Let’s look at SKB shinfo.
- 2 frags, each a leaking page.

```
>>> shinfo = skb_shinfo(skb)
>>> shinfo
*(struct skb_shared_info *)0xffff888005b992c0 = {
        .flags = (__u8)0,
        .meta_len = (__u8)0,
        .nr_frags = (__u8)2,
        .tx_flags = (__u8)0,
        .gso_size = (unsigned short)0,
        .gso_segs = (unsigned short)0,
        .frag_list = (struct sk_buff *)0x0,
        .hwtstamps = (struct skb_shared_hwtstamps){
                .hwtstamp = (ktime_t)0,
                .netdev_data = (void *)0x0,
        },
        .xsk_meta = (struct xsk_tx_metadata_compl){
                .tx_timestamp = (__u64 *)0x0,
        },
        .gso_type = (unsigned int)0,
        .tskey = (u32)0,
        .dataref = (atomic_t){
                .counter = (int)1,
        },
        .xdp_frags_size = (u32)0,
        .xdp_frags_truesize = (u32)0,
        .destructor_arg = (void *)0x0,
        .frags = (skb_frag_t [17]){
                {
                        .netmem = (netmem_ref)18446719884458547520,
                        .len = (unsigned int)1792,
                        .offset = (unsigned int)2304,
                },
                {
                        .netmem = (netmem_ref)18446719884458547456,
                        .len = (unsigned int)3028,
                        .offset = (unsigned int)0,
                },
        },
}
```

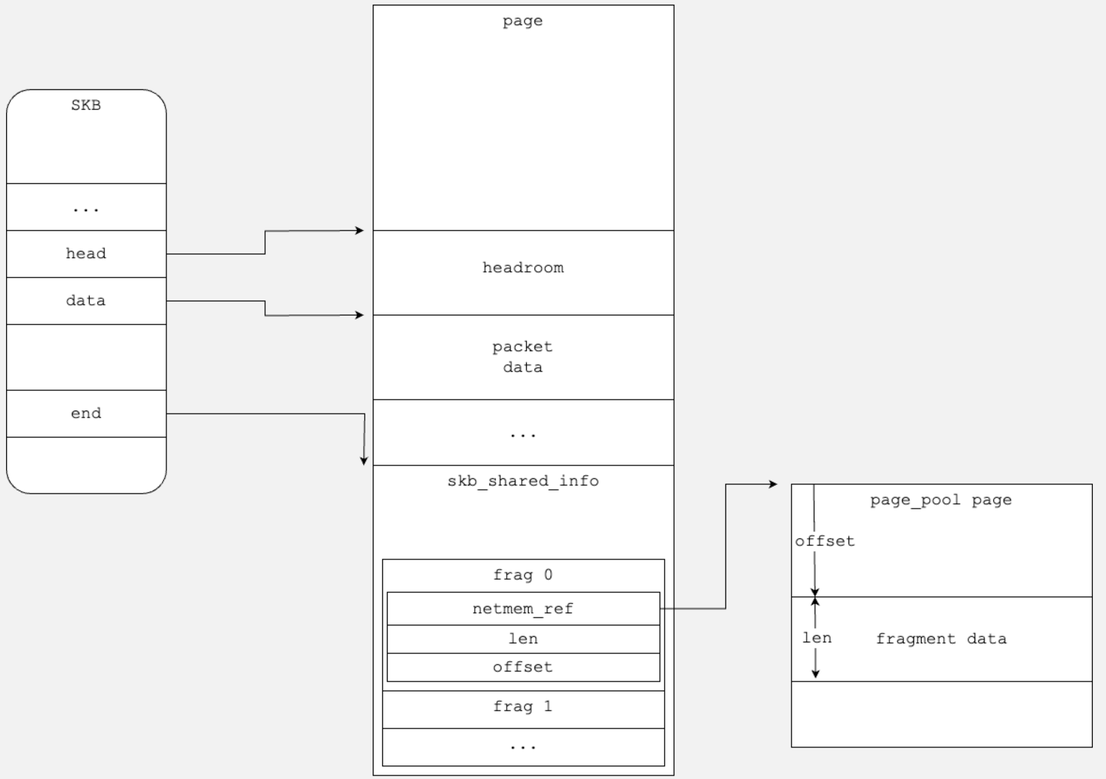
Example 2

What was found

- Leaked pages in SKB fragments.
- Direction:

  Socket -> SKB -> shinfo -> Page
- … let’s try the other way around:

  Page -> shinfo -> SKB -> Socket



Example 2

Variation 2

- Scan all pages for leaked page pool pages.
- Found 2. Showing 1.

Output of ls_pp_leaks.py

```
Leaked page: 0xffffea0000495900
Page content:
ADDRESS           VALUE
ffff888012564000: 2e2e2e2e2e2e2e2e
ffff888012564008: 2e2e2e2e2e2e2e2e
ffff888012564010: 2e2e2e2e2e2e2e2e
ffff888012564018: 2e2e2e2e2e2e2e2e
ffff888012564020: 2e2e2e2e2e2e2e2e
ffff888012564028: 2e2e2e2e2e2e2e2e
ffff888012564030: 2e2e2e2e2e2e2e2e
ffff888012564038: 2e2e2e2e2e2e2e2e
ffff888012564040: 2e2e2e2e2e2e2e2e
ffff888012564048: 2e2e2e2e2e2e2e2e
ffff888012564050: 2e2e2e2e2e2e2e2e
ffff888012564058: 2e2e2e2e2e2e2e2e
ffff888012564060: 000000002e2e2e2e
Leaked page: 0xffffea0000495940
Page content:
ADDRESS           VALUE
ffff888012565000: 70feffffffffffff
ffff888012565008: 010006089b01af02
ffff888012565010: 70fe010004060008
ffff888012565018: 0a0201019b01af02
ffff888012565020: 0101000000000000
ffff888012565028: 0000000000000102
ffff888012565030: 0000000000000000
ffff888012565038: 0000000000000000
ffff888012565040: 000000000001bdfc
ffff888012565048: 00000000000570d0
ffff888012565050: 000000240000000a
ffff888012565058: 000000000003393c
ffff888012565060: 00000000000570df
```

```
>>> page = Object(prog, "struct page", address=0xffffea0000495940)
>>> page
(struct page){
        .flags = (unsigned long)36028797018963968,
        .lru = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800e236800,
        },
        .__filler = (void *)0xdead000000000040,
        .mlock_count = (unsigned int)237201408,
        .buddy_list = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800e236800,
        },
        .pcp_list = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff88800e236800,
        },
        .mapping = (struct address_space *)0x0,
        .index = (unsigned long)307646464,
        .share = (unsigned long)307646464,
        .private = (unsigned long)1,
        .pp_magic = (unsigned long)16045481047390945344,
        .pp = (struct page_pool *)0xffff88800e236800,
        ._pp_mapping_pad = (unsigned long)0,
        .dma_addr = (unsigned long)307646464,
        .pp_ref_count = (atomic_long_t){
                .counter = (s64)1,
        },
        .compound_head = (unsigned long)16045481047390945344,
        .pgmap = (struct dev_pagemap *)0xdead000000000040,
        .zone_device_data = (void *)0xffff88800e236800,
        .callback_head = (struct callback_head){
                .next = (struct callback_head *)0xdead000000000040,
                .func = (void (*)(struct callback_head *))0xffff88800e236800,
        },
        .page_type = (unsigned int)4294967295,
        ._mapcount = (atomic_t){
                .counter = (int)-1,
        },
        ._refcount = (atomic_t){
                .counter = (int)1,
        },
        .memcg_data = (unsigned long)0,
}
```

Example 2

Variation 2

- Find page references.
  - Actual pointer.
- Does it look like SKB or shinfo?
- shinfo -> SKB

Output of find.py

```
Found reference at 0xffff888005b992f0: value 0xffffea0000495940.
ADDRESS           VALUE
ffff888005b99290: 0000000000000000
ffff888005b99298: 0000000000000000
ffff888005b992a0: 0000000000000000
ffff888005b992a8: 0000000000000000
ffff888005b992b0: 0000000000000000
ffff888005b992b8: 0000000000000000
ffff888005b992c0: 0000000000020000
ffff888005b992c8: 0000000000000000
ffff888005b992d0: 0000000000000000
ffff888005b992d8: 0000000000000000
ffff888005b992e0: 0000000000000001
ffff888005b992e8: 0000000000000000
ffff888005b992f0: ffffea0000495940 [page ref in page_pool id: 3, dev eth1, detached: yes]
ffff888005b992f8: 0000090000000700
ffff888005b99300: ffffea0000495900 [page ref in page_pool id: 3, dev eth1, detached: yes]
ffff888005b99308: 00000000000000bd4
ffff888005b99310: 0000000000000000
ffff888005b99318: 0000000000000000
```

```
(drgnenv) dtatulea@virtme-ng /x> ./find.py ffff888005b992f0 --peek-skb

Found reference at 0xffff888005ba88c0: value 0xffff888005b99000. slab object: skbuff_head_cache+0xc0
ADDRESS           VALUE
ffff888005ba8800: ffff888005dc1328 [slab object: TCP+0xa8]
ffff888005ba8808: ffff888005dc1328 [slab object: TCP+0xa8]
ffff888005ba8810: 0000000000000000
ffff888005ba8818: ffff888005dc1280 [slab object: TCP+0x0]
ffff888005ba8820: 0000000000000000
ffff888005ba8828: 815c311c815c1d8a
ffff888005ba8830: 0000001800000000
ffff888005ba8838: 00000000be00254f
ffff888005ba8840: 0000000000000003
ffff888005ba8848: 0000000000000000
ffff888005ba8850: 0000000000000000
ffff888005ba8858: 0000000000000000
ffff888005ba8860: ffffffff81902740 [function symbol: sock_rfree+0x0]
ffff888005ba8868: 0000000000000000
ffff888005ba8870: 000012d400001392
ffff888005ba8878: 00a000010000000e
ffff888005ba8880: 0000000010420040
ffff888005ba8888: 0000000000003ae6
ffff888005ba8890: 8fce042100000003
ffff888005ba8898: 0000020200000000
ffff888005ba88a0: 0000000000000000
ffff888005ba88a8: 0000000000000000
ffff888005ba88b0: 0040004e00620008
ffff888005ba88b8: 000002c000000140
ffff888005ba88c0: ffff888005b99000
ffff888005ba88c8: ffff888005b99082
```

Example 2

Variation 2

- Let’s check…
- Bingo!

```
>>> skb = Object(prog, "struct sk_buff", address=0xffff888005ba8800)
>>> skb
(struct sk_buff){
        .next = (struct sk_buff *)0xffff888005dc1328,
        .prev = (struct sk_buff *)0xffff888005dc1328,
        .dev = (struct net_device *)0x0,
        .dev_scratch = (unsigned long)0,
        .rbnode = (struct rb_node){
                .__rb_parent_color = (unsigned long)18446612682168341288,
                .rb_right = (struct rb_node *)0xffff888005dc1328,
                .rb_left = (struct rb_node *)0x0,
        },
        .list = (struct list_head){
                .next = (struct list_head *)0xffff888005dc1328,
                .prev = (struct list_head *)0xffff888005dc1328,
        },
        .ll_node = (struct llist_node){
                .next = (struct llist_node *)0xffff888005dc1328,
        },
        .sk = (struct sock *)0xffff888005dc1280,
        .tstamp = (ktime_t)0,
        .skb_mstamp_ns = (u64)0,
        .cb = (char [48])"\x8a\x1d\\\x81\x1c1\\\x81",
        ._skb_refdst = (unsigned long)0,
        .destructor = (void (*)(struct sk_buff *))sock_rfree+0x0 = 0xffffffff81902740,
        .tcp_tsorted_anchor = (struct list_head){
                .next = (struct list_head *)0x0,
                .prev = (struct list_head *)sock_rfree+0x0 = 0xffffffff81902740,
        },
        ._sk_redir = (unsigned long)0,
        ._nfct = (unsigned long)0,
        .len = (unsigned int)5010,
        .data_len = (unsigned int)4820,
        .mac_len = (__u16)14,
        .hdr_len = (__u16)0,
        .queue_mapping = (__u16)1,
        .__cloned_offset = (__u8 [0]){},
        .cloned = (__u8)0,
        .nohdr = (__u8)0,
        .fclone = (__u8)0,
        .peeked = (__u8)0,
        .head_frag = (__u8)1,
        .pfmemalloc = (__u8)0,
        .pp_recycle = (__u8)1,
        .active_extensions = (__u8)0,
        .__pkt_type_offset = (__u8 [0]){},
        .pkt_type = (__u8)0,
```

```
        .tail = (sk_buff_data_t)320,
        .end = (sk_buff_data_t)704,
        .head = (unsigned char *)0xffff888005b99000 = "",
        .data = (unsigned char *)0xffff888005b99082 = "more data.....................
.....................................................................................
        .truesize = (unsigned int)6144,
        .users = (refcount_t){
                .refs = (atomic_t){
                        .counter = (int)1,
                },
        },
        .extensions = (struct skb_ext *)0x0,
}
```

Example 2

Conclusions

- Found page leaks in SKB fragments.
- pp_ref_count = 1
- Diagnosis: false positive.

Example 3

- Last one.

```
+ /w/linux/tools/net/ynl/cli.py --no-schema --spec /w/linux/Documentation/netlink/specs/netdev.yaml --dump page-pool-get
[{'detach-time': 40,
  'id': 4,
  'ifindex': 3,
  'inflight': 1,
  'inflight-mem': 4096,
  'napi-id': 515}]
```

Example 3

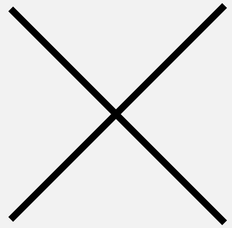
Variation 1

- Search sockets for SKBs with leaked page_pool pages.

Example 3

Variation 1

- No results!



Example 3

Variation 2

- Scan all pages for leaked page pool pages.

```
(drgnenv) dtatulea@virtme-ng /x> ./ls_pp_leaks.py
Leaked page: 0xffffea00004207c0
Page content:
ADDRESS           VALUE
ffff88801081f000: 8f11037a6bb52ca9
ffff88801081f008: abcb9c9244b629ff
ffff88801081f010: a59158bdadd9d0b9
ffff88801081f018: 37ab6a2e4a4cda09
ffff88801081f020: a015038cf21050e7
ffff88801081f028: 13854b48b3f9be10
ffff88801081f030: 412c61dd7f680af2
ffff88801081f038: 8a3f965101b1de67
ffff88801081f040: 1beeb874d7cb7f42
ffff88801081f048: beaa8d6c9fd31501
ffff88801081f050: 2e8bbf0b8452c9bc
ffff88801081f058: c7baf6a920a9b5eb
ffff88801081f060: 00000000787bcaaf
```

Example 3

Variation 2

- Let’s peek in the page.
- Large pp_ref_count.

- HW GRO: header or data page?

```
>>> page = Object(prog, "struct page", address=0xffffea00004207c0)
>>> page
(struct page){
        .flags = (unsigned long)36028797018963968,
        .lru = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff888008586000,
        },
        .__filler = (void *)0xdead000000000040,
        .mlock_count = (unsigned int)140009472,
        .buddy_list = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff888008586000,
        },
        .pcp_list = (struct list_head){
                .next = (struct list_head *)0xdead000000000040,
                .prev = (struct list_head *)0xffff888008586000,
        },
        .mapping = (struct address_space *)0x2,
        .index = (unsigned long)276951040,
        .share = (unsigned long)276951040,
        .private = (unsigned long)64,
        .pp_magic = (unsigned long)16045481047390945344,
        .pp = (struct page_pool *)0xffff888008586000,
        ._pp_mapping_pad = (unsigned long)2,
        .dma_addr = (unsigned long)276951040,
        .pp_ref_count = (atomic_long_t){
                .counter = (s64)64,
        },
        .compound_head = (unsigned long)16045481047390945344,
        .pgmap = (struct dev_pagemap *)0xdead000000000040,
        .zone_device_data = (void *)0xffff888008586000,
        .callback_head = (struct callback_head){
                .next = (struct callback_head *)0xdead000000000040,
                .func = (void (*)(struct callback_head *))0xffff888008586000,
        },
        .page_type = (unsigned int)4294967295,
        ._mapcount = (atomic_t){
                .counter = (int)-1,
        },
        ._refcount = (atomic_t){
                .counter = (int)1,
        },
        .memcg_data = (unsigned long)0,
}
```

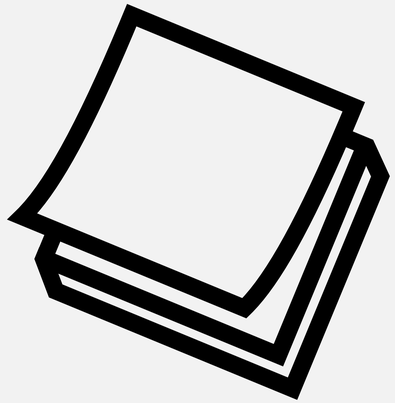
Example 3

Conclusions

- Found leaking page with high pp_ref_count
  - Driver leak.
  - Leaked page was in header page.

Ending Notes

- How about XDP?
- Examples are on a drgn fork for now.
  - Hoping to get some of it accepted in drgn.



Thank you!

Questions?