

[about](#) [summary](#) [refs](#) [log](#) [tree](#) [commit](#) [diff](#) [stats](#)log msg

author Jiri Benc <jbenc@redhat.com> 2022-11-02 17:53:25 +0100
committer Greg Kroah-Hartman <gregkh@linuxfoundation.org> 2022-11-25 17:36:44 +0100
commit [0a9f56e525ea871d3950b90076912f5c7494f00f](#) (patch)
tree [ab6cf0c175bb25d237726bc64ae9efa198a9c28b](#)
parent [249b743801c00542e9324f87b380032e957a43e8](#) (diff)
download [linux-0a9f56e525ea871d3950b90076912f5c7494f00f.tar.gz](#)

diff options

context:
space:
mode:

net: gso: fix panic on frag_list with mixed head alloc types

[Upstream commit 9e4b7a99a03aefd37ba7bb1f022c8efab5019165]

Since commit 3dcbdb134f32 ("net: gso: Fix skb_segment splat when splitting gso_size mangled skb having linear-headed frag_list"), it is allowed to change gso_size of a GRO packet. However, that commit assumes that "checking the first list_skb member suffices; i.e if either of the list_skb members have non head_frag head, then the first one has too".

It turns out this assumption does not hold. We've seen BUG_ON being hit in skb_segment when skbs on the frag_list had differing head_frag with the vmxnet3 driver. This happens because __netdev_alloc_skb and __napi_alloc_skb can return a skb that is page backed or kmalloced depending on the requested size. As the result, the last small skb in the GRO packet can be kmalloced.

There are three different locations where this can be fixed:

- (1) We could check head_frag in GRO and not allow GROing skbs with different head_frag. However, that would lead to performance regression on normal forward paths with unmodified gso_size, where !head_frag in the last packet is not a problem.
- (2) Set a flag in bpf_skb_net_grow and bpf_skb_net_shrink indicating that NETIF_F_SG is undesirable. That would need to eat a bit in sk_buff. Furthermore, that flag can be unset when all skbs on the frag_list are page backed. To retain good performance, bpf_skb_net_grow/shrink would have to walk the frag_list.
- (3) Walk the frag_list in skb_segment when determining whether NETIF_F_SG should be cleared. This of course slows things down.

This patch implements (3). To limit the performance impact in skb_segment, the list is walked only for skbs with SKB_GSO_DODGY set that have gso_size changed. Normal paths thus will not hit it.

We could check only the last skb but since we need to walk the whole list anyway, let's stay on the safe side.

Fixes: 3dcbdb134f32 ("net: gso: Fix skb_segment splat when splitting gso_size mangled skb having linear-headed frag_list")
Signed-off-by: Jiri Benc <jbenc@redhat.com>
Reviewed-by: Willem de Bruijn <willemdebruijn@google.com>
Link: <https://lore.kernel.org/r/e04426a6a91baf4d1081e1b478c82b5de25fdf21.1667407944.git.jbenc@redhat.com>
Signed-off-by: Jakub Kicinski <kuba@kernel.org>
Signed-off-by: Sasha Levin <sashal@kernel.org>

Diffstat

```
-rw-r--r-- net/core/skbuff.c 36
```

1 files changed, 19 insertions, 17 deletions

diff --git a/net/core/skbuff.c b/net/core/skbuff.c

index 11d0ffc51c24a5..0b672d71447f86 100644

--- a/net/core/skbuff.c

+++ b/net/core/skbuff.c

```
@@ -3545,23 +3545,25 @@ struct skb_segment(struct sk_buff *head_skb,
    int pos;
    int dummy;
```

```

- if (list_skb && !list_skb->head_frag && skb_headlen(list_skb) &&
-     (skb_shinfo(head_skb)->gso_type & SKB_GSO_DODGY)) {
-     /* gso_size is untrusted, and we have a frag_list with a linear
-      * non head_frag head.
-      *
-      * (we assume checking the first list_skb member suffices;
-      * i.e if either of the list_skb members have non head_frag
-      * head, then the first one has too).
-      *
-      * If head_skb's headlen does not fit requested gso_size, it
-      * means that the frag_list members do NOT terminate on exact
-      * gso_size boundaries. Hence we cannot perform skb_frag_t page
-      * sharing. Therefore we must fallback to copying the frag_list
-      * skbs; we do so by disabling SG.
-      */
-     if (mss != GSO_BY_FRAGS && mss != skb_headlen(head_skb))
-         features &= ~NETIF_F_SG;
+ if ((skb_shinfo(head_skb)->gso_type & SKB_GSO_DODGY) &&
+     mss != GSO_BY_FRAGS && mss != skb_headlen(head_skb)) {
+     struct sk_buff *check_skb;
+
+     for (check_skb = list_skb; check_skb; check_skb = check_skb->next) {
+         if (skb_headlen(check_skb) && !check_skb->head_frag) {
+             /* gso_size is untrusted, and we have a frag_list with
+              * a linear non head_frag item.
+              *
+              * If head_skb's headlen does not fit requested gso_size,
+              * it means that the frag_list members do NOT terminate
+              * on exact gso_size boundaries. Hence we cannot perform
+              * skb_frag_t page sharing. Therefore we must fallback to
+              * copying the frag_list skbs; we do so by disabling SG.
+              */
+             features &= ~NETIF_F_SG;
+             break;
+         }
+     }
+ }
+
+ __skb_push(head_skb, doffset);

```