Translation table TVLV functions not thread safe

There is the possibility that batadv_tt_prepare_tvlv_global_data/batadv_tt_prepare_tvlv_local_data and any function operating on the buffer is causing a buffer overflow

Here some details from IRC

<ecsv_> hm, isn't the batadv_tt_prepare_tvlv_global_data buggy?
<ecsv_> it traverses the vlan_list twice in hope that it returns the same number of entries
<ecsv_> this doesn't have to be true because it is not locked with a mutex/spinlock
<ecsv_> let us assume that something adds a vlan to vlan_list while batadv_tt_prepare_tvlv_global_data is running
<ecsv_> and this happens on a core so that the current core first sees the new entry in the list after the kmalloc
<ecsv_> the second loop will traverse the list again and then at the end write an entry more than it is expected to write to the buffer
<ecsv_> or is tt_buff_lock also locked around the vlan/tt entry del/add functions?
<ecsv_> ah, no. it isn't. for example the non-full table doesn't use it in batadv_send_other_tt_response

Potential fix is:

```diff--git a/net/batman-adv/translation-table.c b/net/batman-adv/translation-table.c
index 0225616d..7fa3a0a0 100644
--- a/net/batman-adv/translation-table.c
+++ b/net/batman-adv/translation-table.c
@@ -862,7 +862,7 @@ batadv_tt_prepare_tvlv_global_data(struct batadv_orig_node *orig_node,
 struct batadv_orig_node_vlan *vlan;
 u8 *tt_change_ptr;
-   rcu_read_lock();
+   spin_lock_bh(&orig_node->vlan_list_lock);
     hlist_for_each_entry_rcu(vlan, &orig_node->vlan_list, list) {
         num_vlan++;
         num_entries += atomic_read(&vlan->tt.num_entries);
@@ -900,7 +900,7 @@ batadv_tt_prepare_tvlv_global_data(struct batadv_orig_node *orig_node,
     *tt_change = (struct batadv_tvlv_tt_change *)tt_change_ptr;
 }
@@ -936,7 +936,7 @@ batadv_tt_prepare_tvlv_local_data(struct batadv_priv *bat_priv,
 u8 *tt_change_ptr;
     int change_offset;
-   rcu_read_lock();
+   spin_lock_bh(&bat_priv->softif_vlan_list_lock);
     hlist_for_each_entry_rcu(vlan, &bat_priv->softif_vlan_list, list) {
```

02/22/2020
num_vlan++;
num_entries += atomic_read(&vlan->tt.num_entries);
@@ -974,7 +974,7 @@ batadv_tt_prepare_tvlv_local_data(struct batadv_priv *bat_priv,
 *tt_change = (struct batadv_tvlv_tt_change *)tt_change_ptr;
@@
out:
-rcu_read_unlock();
+spin_unlock_bh(&bat_priv->softif_vlan_list_lock);
    return tvlv_len;
}

A patch (untested) can be found at https://patchwork.open-mesh.org/patch/17350/

#3 - 05/10/2018 03:25 AM - Antonio Quartulli

isn't the rcu_lock/unlock enough to guarantee that the objects accessed in that timeframe won't be altered by any list_add_rcu (or similar _rcu name) function?

#4 - 05/10/2018 08:18 AM - Sven Eckelmann

No, rcu provides only very weak guarantees (existence guarantees). Used correctly, it would just give you (for this list):

- the iteration over this list will not point to a non existing object at any time (which would cause a segfault otherwise)
- each object, which is in this rcu protected list and which you access in this rcu-lock, will not just suddenly disappear to exist as object in memory
  - this doesn't mean that the list is not modified - just that you can operate "freely" on this list without suddenly seeing a lot of segfaults because the RCU mechanism decided that the object from the list can now be kfree'd and thus end its existence

I simplified it here but you can find the details in the perfbook. But the LWN RCU article also summarizes it very well:

The effect is that if any RCU-protected data element is accessed within an RCU read-side critical section, that data element is guaranteed to remain in existence for the duration of that RCU read-side critical section.

#5 - 05/10/2018 04:13 PM - Sven Eckelmann

- Target version set to 2018.2
- Status changed from In Progress to Resolved

Patch was applied in https://git.open-mesh.org/batman-adv.git/commit/286be89a33497ba9000aa5c2960f1f14114953522 and queued for net.git

02/22/2020 2/3
batman-adv 2018.2 was just released