author	Maxim Mikityanskiy <maxtram95@gmail.com>	2022-10-05 00:27:18 +0300
committer	Luiz Augusto von Dentz <luiz.von.dentz@intel.com>	2022-11-02 14:12:34 -0700
commit	3aff8aaca4e36dc8b17eaa011684881a80238966 (patch)	
tree	a7f5cac74679edf1ae891378fc935c95be9cef8d	
parent	ba9169f57090efdee6b13601fced57e123db8777 (diff)	
download	linux-3aff8aaca4e36dc8b17eaa011684881a80238966.tar.gz	

Bluetooth: L2CAP: Fix use-after-free caused by l2cap_reassemble_sdu

```
Fix the race condition between the following two flows that run in
parallel:

1. l2cap_reassemble_sdu -> chan->ops->recv (l2cap_sock_recv_cb) ->
   __sock_queue_rcv_skb.

2. bt_sock_recvmsg -> skb_recv_datagram, skb_free_datagram.

An SKB can be queued by the first flow and immediately dequeued and
freed by the second flow, therefore the callers of l2cap_reassemble_sdu
can't use the SKB after that function returns. However, some places
continue accessing struct l2cap_ctrl that resides in the SKB's CB for a
short time after l2cap_reassemble_sdu returns, leading to a
use-after-free condition (the stack trace is below, line numbers for
kernel 5.19.8).

Fix it by keeping a local copy of struct l2cap_ctrl.

BUG: KASAN: use-after-free in l2cap_rx_state_recv (net/bluetooth/l2cap_core.c:6906) bluetooth
Read of size 1 at addr ffff88812025f2f0 by task kworker/u17:3/43169

Workqueue: hci0 hci_rx_work [bluetooth]
Call Trace:
 <TASK>
 dump_stack_lvl (lib/dump_stack.c:107 (discriminator 4))
 print_report.cold (mm/kasan/report.c:314 mm/kasan/report.c:429)
 ? l2cap_rx_state_recv (net/bluetooth/l2cap_core.c:6906) bluetooth
 kasan_report (mm/kasan/report.c:162 mm/kasan/report.c:493)
 ? l2cap_rx_state_recv (net/bluetooth/l2cap_core.c:6906) bluetooth
 l2cap_rx_state_recv (net/bluetooth/l2cap_core.c:6906) bluetooth
 l2cap_rx (net/bluetooth/l2cap_core.c:7236 net/bluetooth/l2cap_core.c:7271) bluetooth
 ret_from_fork (arch/x86/entry/entry_64.S:306)
 </TASK>

Allocated by task 43169:
 kasan_save_stack (mm/kasan/common.c:39)
 __kasan_slab_alloc (mm/kasan/common.c:45 mm/kasan/common.c:436 mm/kasan/common.c:469)
 kmem_cache_alloc_node (mm/slab.h:750 mm/slub.c:3243 mm/slub.c:3293)
 __alloc_skb (net/core/skbuff.c:414)
 l2cap_recv_frag (./include/net/bluetooth/bluetooth.h:425 net/bluetooth/l2cap_core.c:8329) bluetooth
 l2cap_recv_acldata (net/bluetooth/l2cap_core.c:8442) bluetooth
 hci_rx_work (net/bluetooth/hci_core.c:3642 net/bluetooth/hci_core.c:3832) bluetooth
 process_one_work (kernel/workqueue.c:2289)
 worker_thread (./include/linux/list.h:292 kernel/workqueue.c:2437)
 kthread (kernel/kthread.c:376)
 ret_from_fork (arch/x86/entry/entry_64.S:306)

Freed by task 27920:
 kasan_save_stack (mm/kasan/common.c:39)
 kasan_set_track (mm/kasan/common.c:45)
 kasan_set_free_info (mm/kasan/generic.c:372)
 ____kasan_slab_free (mm/kasan/common.c:368 mm/kasan/common.c:328)
 slab_free_freelist_hook (mm/slub.c:1780)
 kmem_cache_free (mm/slub.c:3536 mm/slub.c:3553)
```

```
 skb_free_datagram (./include/net/sock.h:1578 ./include/net/sock.h:1639 net/core/datagram.c:323)
 bt_sock_recvmsg (net/bluetooth/af_bluetooth.c:295) bluetooth
 l2cap_sock_recvmsg (net/bluetooth/l2cap_sock.c:1212) bluetooth
 sock_read_iter (net/socket.c:1087)
 new_sync_read (./include/linux/fs.h:2052 fs/read_write.c:401)
 vfs_read (fs/read_write.c:482)
 ksys_read (fs/read_write.c:620)
 do_syscall_64 (arch/x86/entry/common.c:50 arch/x86/entry/common.c:80)
 entry_SYSCALL_64_after_hwframe (arch/x86/entry/entry_64.S:120)

Link: https://lore.kernel.org/linux-bluetooth/CAKErNvoqga1WcmoR3-0875esY6TVWFQDandbVZncSiuGPBQXLA@mail.gmail.com/T/#u
Fixes: d2a7ac5d5d3a ("Bluetooth: Add the ERTM receive state machine")
Fixes: 4b51dae96731 ("Bluetooth: Add streaming mode receive and incoming packet classifier")
Signed-off-by: Maxim Mikityanskiy <maxtram95@gmail.com>
Signed-off-by: Luiz Augusto von Dentz <luiz.von.dentz@intel.com>
```

Diffstat

-rw-r--r-- net/bluetooth/l2cap_core.c 48

1 files changed, 41 insertions, 7 deletions

```
diff --git a/net/bluetooth/l2cap_core.c b/net/bluetooth/l2cap_core.c
index 1f34b82ca0ec93..2283871d3f0131 100644
--- a/net/bluetooth/l2cap_core.c
+++ b/net/bluetooth/l2cap_core.c
@@ -6885,6 +6885,7 @@ static int l2cap_rx_state_recv(struct l2cap_chan *chan,
                               struct l2cap_ctrl *control,
                               struct sk_buff *skb, u8 event)
 {
+       struct l2cap_ctrl local_control;
        int err = 0;
        bool skb_in_use = false;

@@ -6909,15 +6910,32 @@ static int l2cap_rx_state_recv(struct l2cap_chan *chan,
                        chan->buffer_seq = chan->expected_tx_seq;
                        skb_in_use = true;

+                       /* l2cap_reassemble_sdu may free skb, hence invalidate
+                        * control, so make a copy in advance to use it after
+                        * l2cap_reassemble_sdu returns and to avoid the race
+                        * condition, for example:
+                        *
+                        * The current thread calls:
+                        *   l2cap_reassemble_sdu
+                        *     chan->ops->recv == l2cap_sock_recv_cb
+                        *       __sock_queue_rcv_skb
+                        * Another thread calls:
+                        *   bt_sock_recvmsg
+                        *     skb_recv_datagram
+                        *     skb_free_datagram
+                        * Then the current thread tries to access control, but
+                        * it was freed by skb_free_datagram.
+                        */
+                       local_control = *control;
                        err = l2cap_reassemble_sdu(chan, skb, control);
                        if (err)
                                break;

-                       if (control->final) {
+                       if (local_control.final) {
                                if (!test_and_clear_bit(CONN_REJ_ACT,
                                                        &chan->conn_state)) {
-                                       control->final = 0;
-                                       l2cap_retransmit_all(chan, control);
+                                       local_control.final = 0;
+                                       l2cap_retransmit_all(chan, &local_control);
                                        l2cap_ertm_send(chan);
                                }
                        }
@@ -7297,11 +7315,27 @@ static int l2cap_rx(struct l2cap_chan *chan, struct l2cap_ctrl *control,
 static int l2cap_stream_rx(struct l2cap_chan *chan, struct l2cap_ctrl *control,
                           struct sk_buff *skb)
 {
+       /* l2cap_reassemble_sdu may free skb, hence invalidate control, so store
```

```
+         * the txseq field in advance to use it after l2cap_reassemble_sdu
+         * returns and to avoid the race condition, for example:
+         *
+         * The current thread calls:
+         *   l2cap_reassemble_sdu
+         *     chan->ops->recv == l2cap_sock_recv_cb
+         *       __sock_queue_rcv_skb
+         * Another thread calls:
+         *   bt_sock_recvmsg
+         *     skb_recv_datagram
+         *     skb_free_datagram
+         * Then the current thread tries to access control, but it was freed by
+         * skb_free_datagram.
+         */
+        u16 txseq = control->txseq;
+
         BT_DBG("chan %p, control %p, skb %p, state %d", chan, control, skb,
                chan->rx_state);

-        if (l2cap_classify_txseq(chan, control->txseq) ==
-            L2CAP_TXSEQ_EXPECTED) {
+        if (l2cap_classify_txseq(chan, txseq) == L2CAP_TXSEQ_EXPECTED) {
                 l2cap_pass_to_tx(chan, control);

                 BT_DBG("buffer_seq %u->%u", chan->buffer_seq,
@@ -7324,8 +7358,8 @@ static int l2cap_stream_rx(struct l2cap_chan *chan, struct l2cap_ctrl *control,
                 }
         }

-        chan->last_acked_seq = control->txseq;
-        chan->expected_tx_seq = __next_seq(chan, control->txseq);
+        chan->last_acked_seq = txseq;
+        chan->expected_tx_seq = __next_seq(chan, txseq);

         return 0;
 }
```