| | | |
|---|---|---|
| author | Ricardo Cañuelo Navarro <rcn@igalia.com> | 2025-04-04 16:53:21 +0200 |
| committer | Greg Kroah-Hartman <gregkh@linuxfoundation.org> | 2025-04-20 10:18:18 +0200 |
| commit | 5bc83bdf5f5b8010d1ca5a4555537e62413ab4e2 (patch) | |
| tree | 731aab0b09335032eb09df04b112d33316c21d7d | |
| parent | e8defb8b2082cc031342e2a4217d7aa1a36a1fe8 (diff) | |
| download | linux-5bc83bdf5f5b8010d1ca5a4555537e62413ab4e2.tar.gz | |

diff options

context: 3

space: include

mode: unified

## sctp: detect and prevent references to a freed transport in sendmsg

commit f1a69a940de58b16e8249dff26f74c8cc59b32be upstream.

sctp_sendmsg() re-uses associations and transports when possible by
doing a lookup based on the socket endpoint and the message destination
address, and then sctp_sendmsg_to_asoc() sets the selected transport in
all the message chunks to be sent.

There's a possible race condition if another thread triggers the removal
of that selected transport, for instance, by explicitly unbinding an
address with setsockopt(SCTP_SOCKOPT_BINDX_REM), after the chunks have
been set up and before the message is sent. This can happen if the send
buffer is full, during the period when the sender thread temporarily
releases the socket lock in sctp_wait_for_sndbuf().

This causes the access to the transport data in
sctp_outq_select_transport(), when the association outqueue is flushed,
to result in a use-after-free read.

This change avoids this scenario by having sctp_transport_free() signal
the freeing of the transport, tagging it as "dead". In order to do this,
the patch restores the "dead" bit in struct sctp_transport, which was
removed in
commit 47faa1e4c50e ("sctp: remove the dead field of sctp_transport").

Then, in the scenario where the sender thread has released the socket
lock in sctp_wait_for_sndbuf(), the bit is checked again after
re-acquiring the socket lock to detect the deletion. This is done while
holding a reference to the transport to prevent it from being freed in
the process.

If the transport was deleted while the socket lock was relinquished,
sctp_sendmsg_to_asoc() will return -EAGAIN to let userspace retry the
send.

The bug was found by a private syzbot instance (see the error report [1]
and the C reproducer that triggers it [2]).

Link: https://people.igalia.com/rcn/kernel_logs/20250402__KASAN_slab-use-after-free_Read_in_sctp_outq_select_transport.txt [1]
Link: https://people.igalia.com/rcn/kernel_logs/20250402__KASAN_slab-use-after-free_Read_in_sctp_outq_select_transport__repro.c [2]
Cc: stable@vger.kernel.org
Fixes: df132eff4638 ("sctp: clear the transport of some out_chunk_list chunks in sctp_assoc_rm_peer")
Suggested-by: Xin Long <lucien.xin@gmail.com>
Signed-off-by: Ricardo Cañuelo Navarro <rcn@igalia.com>
Acked-by: Xin Long <lucien.xin@gmail.com>
Link: https://patch.msgid.link/20250404-kasan_slab-use-after-free_read_in_sctp_outq_select_transport__20250404-v1-1-5ce4a0b78ef2@igalia.com
Signed-off-by: Paolo Abeni <pabeni@redhat.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>

**Diffstat**

| | | |
|---|---|---|
| -rw-r--r-- | include/net/sctp/structs.h | 3 |
| -rw-r--r-- | net/sctp/socket.c | 22 |
| -rw-r--r-- | net/sctp/transport.c | 2 |

3 files changed, 18 insertions, 9 deletions

```
diff --git a/include/net/sctp/structs.h b/include/net/sctp/structs.h
index 31248cfdfb235f..dcd288fa1bb6fb 100644
--- a/include/net/sctp/structs.h
+++ b/include/net/sctp/structs.h
@@ -775,6 +775,7 @@ struct sctp_transport {

        /* Reference counting. */
        refcount_t refcnt;
+       __u32   dead:1,
                /* RTO-Pending : A flag used to track if one of the DATA
                 *              chunks sent to this address is currently being
                 *              used to compute a RTT. If this flag is 0,
```

```
@@ -784,7 +785,7 @@ struct sctp_transport {
                  *              calculation completes (i.e. the DATA chunk
                  *              is SACK'd) clear this flag.
                  */
-       __u32   rto_pending:1,
+               rto_pending:1,

                /*
                 * hb_sent : a flag that signals that we have a pending
```

```
diff --git a/net/sctp/socket.c b/net/sctp/socket.c
index 36ee34f483d703..53725ee7ba06d7 100644
--- a/net/sctp/socket.c
+++ b/net/sctp/socket.c
@@ -72,8 +72,9 @@
 /* Forward declarations for internal helper functions. */
 static bool sctp_writeable(const struct sock *sk);
 static void sctp_wfree(struct sk_buff *skb);
-static int sctp_wait_for_sndbuf(struct sctp_association *asoc, long *timeo_p,
-                                size_t msg_len);
+static int sctp_wait_for_sndbuf(struct sctp_association *asoc,
+                                struct sctp_transport *transport,
+                                long *timeo_p, size_t msg_len);
 static int sctp_wait_for_packet(struct sock *sk, int *err, long *timeo_p);
 static int sctp_wait_for_connect(struct sctp_association *, long *timeo_p);
 static int sctp_wait_for_accept(struct sock *sk, long timeo);
@@ -1828,7 +1829,7 @@ static int sctp_sendmsg_to_asoc(struct sctp_association *asoc,

        if (sctp_wspace(asoc) <= 0 || !sk_wmem_schedule(sk, msg_len)) {
                timeo = sock_sndtimeo(sk, msg->msg_flags & MSG_DONTWAIT);
-               err = sctp_wait_for_sndbuf(asoc, &timeo, msg_len);
+               err = sctp_wait_for_sndbuf(asoc, transport, &timeo, msg_len);
                if (err)
                        goto err;
                if (unlikely(sinfo->sinfo_stream >= asoc->stream.outcnt)) {
@@ -9214,8 +9215,9 @@ void sctp_sock_rfree(struct sk_buff *skb)


 /* Helper function to wait for space in the sndbuf.  */
-static int sctp_wait_for_sndbuf(struct sctp_association *asoc, long *timeo_p,
-                                size_t msg_len)
+static int sctp_wait_for_sndbuf(struct sctp_association *asoc,
+                                struct sctp_transport *transport,
+                                long *timeo_p, size_t msg_len)
 {
        struct sock *sk = asoc->base.sk;
        long current_timeo = *timeo_p;
@@ -9225,7 +9227,9 @@ static int sctp_wait_for_sndbuf(struct sctp_association *asoc, long *timeo_p,
        pr_debug("%s: asoc:%p, timeo:%ld, msg_len:%zu\n", __func__, asoc,
                 *timeo_p, msg_len);

-       /* Increment the association's refcnt.  */
+       /* Increment the transport and association's refcnt. */
+       if (transport)
+               sctp_transport_hold(transport);
        sctp_association_hold(asoc);

        /* Wait on the association specific sndbuf space. */
@@ -9234,7 +9238,7 @@ static int sctp_wait_for_sndbuf(struct sctp_association *asoc, long *timeo_p,
                                          TASK_INTERRUPTIBLE);
                if (asoc->base.dead)
                        goto do_dead;
-               if (!*timeo_p)
+               if ((!*timeo_p) || (transport && transport->dead))
                        goto do_nonblock;
                if (sk->sk_err || asoc->state >= SCTP_STATE_SHUTDOWN_PENDING)
                        goto do_error;
@@ -9259,7 +9263,9 @@ static int sctp_wait_for_sndbuf(struct sctp_association *asoc, long *timeo_p,
 out:
        finish_wait(&asoc->wait, &wait);

-       /* Release the association's refcnt.  */
+       /* Release the transport and association's refcnt. */
+       if (transport)
+               sctp_transport_put(transport);
        sctp_association_put(asoc);

        return err;
```

```
diff --git a/net/sctp/transport.c b/net/sctp/transport.c
index 2abe45af98e7c6..31eca29b6cfbfb 100644
--- a/net/sctp/transport.c
+++ b/net/sctp/transport.c
@@ -117,6 +117,8 @@ fail:
  */
 void sctp_transport_free(struct sctp_transport *transport)
 {
+       transport->dead = 1;
```

```
+
        /* Try to delete the heartbeat timer.  */
        if (del_timer(&transport->hb_timer))
                sctp_transport_put(transport);
```