author	Tung Nguyen <tung.quang.nguyen@est.tech>	2025-04-17 14:47:15 +0700
committer	Greg Kroah-Hartman <gregkh@linuxfoundation.org>	2025-05-02 07:41:15 +0200
commit	e6613b6d41f4010c4d484cbc7bfca690d8d522a2 (patch)	
tree	12bceecb94964eeb201333d807d628899baa236a	
parent	95bed65cc0eb2a610550abf849a8b94374da80a7 (diff)	
download	linux-e6613b6d41f4010c4d484cbc7bfca690d8d522a2.tar.gz	

tipc: fix NULL pointer dereference in tipc_mon_reinit_self()

```
[ Upstream commit d63527e109e811ef11abb1c2985048fdb528b4cb ]

syzbot reported:

tipc: Node number set to 1055423674
Oops: general protection fault, probably for non-canonical address 0xdfffc0000000000: 0000 [#1] SMP KASAN NOPTI
KASAN: null-ptr-deref in range [0x0000000000000000-0x0000000000000007]
CPU: 3 UID: 0 PID: 6017 Comm: kworker/3:5 Not tainted 6.15.0-rc1-syzkaller-00246-g900241a5cc15 #0 PREEMPT(full)
Hardware name: QEMU Standard PC (Q35 + ICH9, 2009), BIOS 1.16.3-debian-1.16.3-2~bpo12+1 04/01/2014
Workqueue: events tipc_net_finalize_work
RIP: 0010:tipc_mon_reinit_self+0x11c/0x210 net/tipc/monitor.c:719
...
RSP: 0018:ffffc9000356fb68 EFLAGS: 00010246
RAX: 0000000000000000 RBX: 0000000000000000 RCX: 000000003ee87cba
RDX: 0000000000000000 RSI: ffffffff8dbc56a7 RDI: ffff88804c2cc010
RBP: dfffffc0000000000 R08: 0000000000000001 R09: 0000000000000000
R10: 0000000000000001 R11: 0000000000000000 R12: 0000000000000007
R13: fffffbfff2111097 R14: ffff88804ead8000 R15: ffff88804ead9010
FS:  0000000000000000(0000) GS:ffff888097ab9000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 00000000f720eb00 CR3: 000000000e182000 CR4: 0000000000352ef0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400
Call Trace:
 <TASK>
 tipc_net_finalize+0x10b/0x180 net/tipc/net.c:140
 process_one_work+0x9cc/0x1b70 kernel/workqueue.c:3238
 process_scheduled_works kernel/workqueue.c:3319 [inline]
 worker_thread+0x6c8/0xf10 kernel/workqueue.c:3400
 kthread+0x3c2/0x780 kernel/kthread.c:464
 ret_from_fork+0x45/0x80 arch/x86/kernel/process.c:153
 ret_from_fork_asm+0x1a/0x30 arch/x86/entry/entry_64.S:245
 </TASK>
...
RIP: 0010:tipc_mon_reinit_self+0x11c/0x210 net/tipc/monitor.c:719
...
RSP: 0018:ffffc9000356fb68 EFLAGS: 00010246
RAX: 0000000000000000 RBX: 0000000000000000 RCX: 000000003ee87cba
RDX: 0000000000000000 RSI: ffffffff8dbc56a7 RDI: ffff88804c2cc010
RBP: dfffffc0000000000 R08: 0000000000000001 R09: 0000000000000000
R10: 0000000000000001 R11: 0000000000000000 R12: 0000000000000007
R13: fffffbfff2111097 R14: ffff88804ead8000 R15: ffff88804ead9010
FS:  0000000000000000(0000) GS:ffff888097ab9000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 00000000f720eb00 CR3: 000000000e182000 CR4: 0000000000352ef0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400

There is a racing condition between workqueue created when enabling
bearer and another thread created when disabling bearer right after
that as follow:
```

```
enabling_bearer                                 | disabling_bearer
---------------                                 | ----------------
tipc_disc_timeout()                             |
{                                               | bearer_disable()
 ...                                            | {
 schedule_work(&tn->work);                      |  tipc_mon_delete()
 ...                                            |  {
}                                               |   ...
                                                |   write_lock_bh(&mon->lock);
                                                |   mon->self = NULL;
                                                |   write_unlock_bh(&mon->lock);
                                                |   ...
                                                |  }
tipc_net_finalize_work()                        | }
{                                               |
 ...                                            |
 tipc_net_finalize()                            |
 {                                              |
  ...                                           |
  tipc_mon_reinit_self()                        |
  {                                             |
   ...                                          |
   write_lock_bh(&mon->lock);                   |
   mon->self->addr = tipc_own_addr(net);        |
   write_unlock_bh(&mon->lock);                 |
   ...                                          |
  }                                             |
  ...                                           |
 }                                              |
 ...                                            |
}                                               |
```

'mon->self' is set to NULL in disabling_bearer thread and dereferenced
later in enabling_bearer thread.

This commit fixes this issue by validating 'mon->self' before assigning
node address to it.

Reported-by: syzbot+ed60da8d686dc709164c@syzkaller.appspotmail.com
Fixes: 46cb01eeeb86 ("tipc: update mon's self addr when node addr generated")
Signed-off-by: Tung Nguyen <tung.quang.nguyen@est.tech>
Reviewed-by: Simon Horman <horms@kernel.org>
Link: https://patch.msgid.link/20250417074826.578115-1-tung.quang.nguyen@est.tech
Signed-off-by: Jakub Kicinski <kuba@kernel.org>
Signed-off-by: Sasha Levin <sashal@kernel.org>

Diffstat

-rw-r--r-- net/tipc/monitor.c 3

1 files changed, 2 insertions, 1 deletions

```
diff --git a/net/tipc/monitor.c b/net/tipc/monitor.c
index 1d90f39129ca07..ba0a308d41d85e 100644
--- a/net/tipc/monitor.c
+++ b/net/tipc/monitor.c
@@ -685,7 +685,8 @@ void tipc_mon_reinit_self(struct net *net)
                if (!mon)
                        continue;
                write_lock_bh(&mon->lock);
-               mon->self->addr = tipc_own_addr(net);
+               if (mon->self)
+                       mon->self->addr = tipc_own_addr(net);
                write_unlock_bh(&mon->lock);
        }
 }
```