Bug 2066799 (CVE-2022-1247) - CVE-2022-1247 kernel: A race condition bug in rose_connect()

Keywords:	Security ×	**Reported:**	2022-03-22 14:06 UTC by Rohit Keshri
Status:	NEW	**Modified:**	2022-09-26 12:56 UTC (History)
Alias:	CVE-2022-1247	**CC List:**	50 users (show)
Product:	Security Response	**Fixed In Version:**	
Component:	vulnerability	**Doc Type:**	If docs needed, set a value
Version:	unspecified	**Doc Text:**	
Hardware:	All	**Clone Of:**	
OS:	Linux	**Environment:**	
Priority:	medium	**Last Closed:**	
Severity:	medium		
Target Milestone:	---		
Assignee:	Red Hat Product Security		
QA Contact:			
Docs Contact:			
URL:			
Whiteboard:			
Depends On:	2084050 2072205 2072206 2072207 2072208 2072209 2072210		
Blocks:	2066803 2072221		
TreeView+	depends on / blocked		

Attachments	(Terms of Use)
Add an attachment (proposed patch, testcase, etc.)	

Rohit Keshri 2022-03-22 14:06:33 UTC Description

```
We have found a race condition bug in rose_connect(). The rose
driver uses
rose_neigh->use to represent how many objects are using the
rose_neigh.
When a user wants to delete a rose_route via rose_ioctl(), the
rose driver
calls rose_del_node() and removes neighbours only if their
"count" and
"use" are zero:

'''
static int rose_del_node(struct rose_route_struct *rose_route,
struct net_device *dev)
{
...
spin_lock_bh(&rose_node_list_lock);
spin_lock_bh(&rose_neigh_list_lock);
...
if (rose_neigh->count == 0 && rose_neigh->use == 0) [1]
rose_remove_neigh(rose_neigh); [2]
...
out:
spin_unlock_bh(&rose_neigh_list_lock);
spin_unlock_bh(&rose_node_list_lock);
...
}
'''
```

As the code shows above, modifications on a rose_neigh should
be protected
by rose_neigh_list_lock. However, rose_neigh->use is changed
without any
locks in rose_connect():

```
'''
static int rose_connect(struct socket *sock, struct sockaddr
*uaddr, int
addr_len, int flags)
{
...
rose->neighbour = rose_get_neigh(&addr->srose_addr, &cause,
&diagnostic, 0); [3]
...
rose->neighbour->use++; [4]
...
}
'''
```

So if a user creates two threads, t1 and t2, and execute
functions in the
following order:
t1[3] - t2[1] - t2[2] - t3[4]
The rose socket will keep the pointer of the neighbour while
the neighbour
is freed, and thus lead to an uaf.

I wrote a PoC and tested it on Linux-5.17-rc5, the result
shows as below:

```
'''
/ $ /home/pwn/poc > /dev/null
[ 44.546300]
==================================================================
[ 44.547030] BUG: KASAN: use-after-free in
rose_connect+0x11f5/0x1420
[rose]
[ 44.548981] Read of size 2 at addr ffff88800de27b2a by task
exp/305
[ 44.549048]
[ 44.549048] CPU: 7 PID: 305 Comm: exp Tainted: G E
5.17.0-rc5 #1
[ 44.549048] Hardware name: QEMU Standard PC (i440FX + PIIX,
1996), BIOS
1.10.2-1ubuntu1 04/01/2014
[ 44.549048] Call Trace:
[ 44.549048] <TASK>
[ 44.549048] dump_stack_lvl+0x4c/0x63
[ 44.549048] print_address_description.constprop.0+0x24/0x150
[ 44.549048] ? rose_connect+0x11f5/0x1420 [rose]
[ 44.549048] kasan_report.cold+0x82/0xdb
[ 44.549048] ? rose_connect+0x11f5/0x1420 [rose]
[ 44.549048] __asan_report_load2_noabort+0x14/0x20
[ 44.549048] rose_connect+0x11f5/0x1420 [rose]
[ 44.549048] ? rose_new_lci+0xd0/0xd0 [rose]
[ 44.549048] ? aa_af_perm+0x240/0x240
[ 44.549048] ? __kasan_check_write+0x14/0x20
[ 44.549048] ? read_hpet+0x152/0x200
[ 44.549048] ? tomoyo_socket_connect+0xe/0x10
[ 44.549048] __sys_connect_file+0x141/0x1a0
[ 44.549048] ? move_addr_to_kernel.part.0+0x36/0xe0
[ 44.549048] __sys_connect+0x10c/0x140
[ 44.549048] ? __sys_connect_file+0x1a0/0x1a0
[ 44.549048] ? hrtimer_interrupt+0x330/0x740
[ 44.549048] ? debug_smp_processor_id+0x17/0x20
[ 44.549048] ? fpregs_assert_state_consistent+0x4e/0xb0
[ 44.549048] __x64_sys_connect+0x72/0xb0
[ 44.549048] ? irqentry_exit+0x33/0x40
[ 44.549048] do_syscall_64+0x3b/0xc0
[ 44.549048] entry_SYSCALL_64_after_hwframe+0x44/0xae
[ 44.549048] RIP: 0033:0x405b27
[ 44.549048] Code: 44 00 00 41 54 55 41 89 d4 53 48 89 f5 89
fb 48 83 ec
10 e8 0b fb ff ff 44 89 e2 41 89 c0 48 89 ee 89 df b8 2a 00 00
00 0f 05
<48> 3d 00 f0 ff ff 77 33 44 89 c7 89 44 24 0c e8 45 fb ff ff
8b 44
[ 44.549048] RSP: 002b:00007f8454b67d20 EFLAGS: 00000293
ORIG_RAX:
000000000000002a
[ 44.549048] RAX: ffffffffffffffda RBX: 0000000000000003 RCX:
0000000000405b27
[ 44.549048] RDX: 000000000000001c RSI: 00007ffeedd3c684 RDI:
0000000000000003
[ 44.549048] RBP: 00007ffeedd3c684 R08: 0000000000000000 R09:
00007f8454b68700
[ 44.549048] R10: 00007f8454b689d0 R11: 0000000000000293 R12:
000000000000001c
[ 44.549048] R13: 0000000000000000 R14: 00007ffeedd3c680 R15:
00007ffeedd3c5c0
[ 44.549048] </TASK>
[ 44.549048]
[ 44.549048] Allocated by task 189:
[ 44.549048] kasan_save_stack+0x26/0x50
[ 44.549048] __kasan_kmalloc+0x88/0xa0
[ 44.549048] kmem_cache_alloc_trace+0xc0/0x470
[ 44.549048] rose_rt_ioctl+0x7a9/0x1a60 [rose]
[ 44.549048] rose_ioctl+0x5e7/0x6b0 [rose]
[ 44.549048] sock_do_ioctl+0xda/0x1d0
[ 44.549048] sock_ioctl+0x1b5/0x550
[ 44.549048] __x64_sys_ioctl+0x131/0x1a0
[ 44.549048] do_syscall_64+0x3b/0xc0
[ 44.549048] entry_SYSCALL_64_after_hwframe+0x44/0xae
[ 44.549048]
[ 44.549048] Freed by task 304:
[ 44.549048] kasan_save_stack+0x26/0x50
[ 44.549048] kasan_set_track+0x25/0x30
[ 44.549048] kasan_set_free_info+0x24/0x40
[ 44.549048] __kasan_slab_free+0x100/0x140
[ 44.549048] kfree+0x9a/0x2c0
[ 44.549048] rose_remove_neigh+0x1d8/0x2e0 [rose]
[ 44.549048] rose_rt_ioctl+0xf71/0x1a60 [rose]
[ 44.549048] rose_ioctl+0x5e7/0x6b0 [rose]
[ 44.549048] sock_do_ioctl+0xda/0x1d0
[ 44.549048] sock_ioctl+0x1b5/0x550
[ 44.549048] __x64_sys_ioctl+0x131/0x1a0
[ 44.549048] do_syscall_64+0x3b/0xc0
[ 44.549048] entry_SYSCALL_64_after_hwframe+0x44/0xae
[ 44.549048]
[ 44.549048] The buggy address belongs to the object at
ffff88800de27b00
[ 44.549048] which belongs to the cache kmalloc-192 of size
192
[ 44.549048] The buggy address is located 42 bytes inside of
[ 44.549048] 192-byte region [ffff88800de27b00,
ffff88800de27bc0)
[ 44.549048] The buggy address belongs to the page:
[ 44.549048] page:(____ptrval____) refcount:1 mapcount:0
mapping:0000000000000000 index:0x0 pfn:0xde26
[ 44.549048] head:(____ptrval____) order:1 compound_mapcount:0
[ 44.549048] flags:
0xfffffc0010200(slab|head|node=0|zone=1|lastcpupid=0x1fffff)
[ 44.549048] raw: 000fffffc0010200 0000000000000000
dead000000000122
ffff888005842a00
[ 44.549048] raw: 0000000000000000 0000000080200020
00000001ffffffff
0000000000000000
[ 44.549048] page dumped because: kasan: bad access detected
[ 44.549048]
[ 44.549048] Memory state around the buggy address:
[ 44.549048] ffff88800de27a00: fa fb fb fb fb fb fb fb fb fb
fb fb fb fb
fb fb
[ 44.549048] ffff88800de27a80: fb fb fb fb fb fb fb fb fc fc
fc fc fc fc
fc fc
[ 44.549048] >ffff88800de27b00: fa fb fb fb fb fb fb fb fb fb
fb fb fb fb
fb fb
[ 44.549048] ^
[ 44.549048] ffff88800de27b80: fb fb fb fb fb fb fb fb fc fc
fc fc fc fc
fc fc
[ 44.549048] ffff88800de27c00: 00 00 00 00 00 00 00 00 00 00
00 00 00 00
00 00
[ 44.549048]
==================================================================
====
[ 44.549048] Disabling lock debugging due to kernel taint
[ 46.800632] random: fast init done
/ $
'''
```

By the way, I don't think the "count" and "use" fields are well managed in rose driver, because there are many other places they are used without lock protection, such as rose_rx_call_request(), rose_state1_machine(), rose_state2_machine() and so on. These places should also be checked.

Rohit Keshri 2022-05-11 09:38:53 UTC Comment 5

```
Created kernel tracking bugs for this issue:

Affects: fedora-all [ bug 2084050 ]
```