author	Kuniyuki Iwashima <kuniyu@amazon.com>	2025-04-07 09:33:11 -0700
committer	Greg Kroah-Hartman <gregkh@linuxfoundation.org>	2025-04-20 10:15:48 +0200
commit	5f7f6abd92b6c8dc8f19625ef93c3a18549ede04 (patch)	
tree	8ffa2a015a1d9095e6c56839970a45e263ce2e85	
parent	fba396b79942be5432ca27ec003fa0f151235ca6 (diff)	
download	linux-5f7f6abd92b6c8dc8f19625ef93c3a18549ede04.tar.gz	

diff options

context: 3

space: include

mode: unified

net: Fix null-ptr-deref by sock_lock_init_class_and_name() and rmmod.

```
commit 0bb2f7a1ad1f11d861f58e5ee5051c8974ff9569 upstream.

When I ran the repro [0] and waited a few seconds, I observed two
LOCKDEP splats: a warning immediately followed by a null-ptr-deref. [1]

Reproduction Steps:

  1) Mount CIFS
  2) Add an iptables rule to drop incoming FIN packets for CIFS
  3) Unmount CIFS
  4) Unload the CIFS module
  5) Remove the iptables rule

At step 3), the CIFS module calls sock_release() for the underlying
TCP socket, and it returns quickly.  However, the socket remains in
FIN_WAIT_1 because incoming FIN packets are dropped.

At this point, the module's refcnt is 0 while the socket is still
alive, so the following rmmod command succeeds.

  # ss -tan
  State      Recv-Q Send-Q Local Address:Port  Peer Address:Port
  FIN-WAIT-1 0      477        10.0.2.15:51062   10.0.0.137:445

  # lsmod | grep cifs
  cifs                 1159168  0

This highlights a discrepancy between the lifetime of the CIFS module
and the underlying TCP socket.  Even after CIFS calls sock_release()
and it returns, the TCP socket does not die immediately in order to
close the connection gracefully.

While this is generally fine, it causes an issue with LOCKDEP because
CIFS assigns a different lock class to the TCP socket's sk->sk_lock
using sock_lock_init_class_and_name().

Once an incoming packet is processed for the socket or a timer fires,
sk->sk_lock is acquired.

Then, LOCKDEP checks the lock context in check_wait_context(), where
hlock_class() is called to retrieve the lock class.  However, since
the module has already been unloaded, hlock_class() logs a warning
and returns NULL, triggering the null-ptr-deref.

If LOCKDEP is enabled, we must ensure that a module calling
sock_lock_init_class_and_name() (CIFS, NFS, etc) cannot be unloaded
while such a socket is still alive to prevent this issue.

Let's hold the module reference in sock_lock_init_class_and_name()
and release it when the socket is freed in sk_prot_free().

Note that sock_lock_init() clears sk->sk_owner for svc_create_socket()
that calls sock_lock_init_class_and_name() for a listening socket,
which clones a socket by sk_clone_lock() without GFP_ZERO.

[0]:
CIFS_SERVER="10.0.0.137"
CIFS_PATH="//${CIFS_SERVER}/Users/Administrator/Desktop/CIFS_TEST"
DEV="enp0s3"
CRED="/root/WindowsCredential.txt"

MNT=$(mktemp -d /tmp/XXXXXX)
mount -t cifs ${CIFS_PATH} ${MNT} -o vers=3.0,credentials=${CRED},cache=none,echo_interval=1

iptables -A INPUT -s ${CIFS_SERVER} -j DROP

for i in $(seq 10);
do
    umount ${MNT}
    rmmod cifs
    sleep 1
done
```

```
rm -r ${MNT}

iptables -D INPUT -s ${CIFS_SERVER} -j DROP

[1]:
DEBUG_LOCKS_WARN_ON(1)
WARNING: CPU: 10 PID: 0 at kernel/locking/lockdep.c:234 hlock_class (kernel/locking/lockdep.c:234 kernel/locking/lockdep.c:223)
Modules linked in: cifs_arc4 nls_ucs2_utils cifs_md4 [last unloaded: cifs]
CPU: 10 UID: 0 PID: 0 Comm: swapper/10 Not tainted 6.14.0 #36
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.16.0-0-gd239552ce722-prebuilt.qemu.org 04/01/2014
RIP: 0010:hlock_class (kernel/locking/lockdep.c:234 kernel/locking/lockdep.c:223)
...
Call Trace:
 <IRQ>
 __lock_acquire (kernel/locking/lockdep.c:4853 kernel/locking/lockdep.c:5178)
 lock_acquire (kernel/locking/lockdep.c:469 kernel/locking/lockdep.c:5853 kernel/locking/lockdep.c:5816)
 _raw_spin_lock_nested (kernel/locking/spinlock.c:379)
 tcp_v4_rcv (./include/linux/skbuff.h:1678 ./include/net/tcp.h:2547 net/ipv4/tcp_ipv4.c:2350)
...

BUG: kernel NULL pointer dereference, address: 00000000000000c4
 PF: supervisor read access in kernel mode
 PF: error_code(0x0000) - not-present page
PGD 0
Oops: Oops: 0000 [#1] PREEMPT SMP NOPTI
CPU: 10 UID: 0 PID: 0 Comm: swapper/10 Tainted: G        W          6.14.0 #36
Tainted: [W]=WARN
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.16.0-0-gd239552ce722-prebuilt.qemu.org 04/01/2014
RIP: 0010:__lock_acquire (kernel/locking/lockdep.c:4852 kernel/locking/lockdep.c:5178)
Code: 15 41 09 c7 41 8b 44 24 20 25 ff 1f 00 00 41 09 c7 8b 84 24 a0 00 00 00 45 89 7c 24 20 41 89 44 24 24 e8 e1 bc ff ff 4c 89 e7 <44> 0f b6 b8 c4
RSP: 0018:ffa0000000468a10 EFLAGS: 00010046
RAX: 0000000000000000 RBX: ff1100010091cc38 RCX: 0000000000000027
RDX: ff1100081f09ca48 RSI: 0000000000000001 RDI: ff1100010091cc88
RBP: ff1100010091c200 R08: ff1100083fe6e228 R09: 00000000ffffbfff
R10: ff1100081eca0000 R11: ff1100083fe10dc0 R12: ff1100010091cc88
R13: 0000000000000001 R14: 0000000000000000 R15: 00000000000424b1
FS:  0000000000000000(0000) GS:ff1100081f080000(0000) knlGS:0000000000000000
CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: 00000000000000c4 CR3: 0000000002c4a003 CR4: 0000000000771ef0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000fffe07f0 DR7: 0000000000000400
PKRU: 55555554
Call Trace:
 <IRQ>
 lock_acquire (kernel/locking/lockdep.c:469 kernel/locking/lockdep.c:5853 kernel/locking/lockdep.c:5816)
 _raw_spin_lock_nested (kernel/locking/spinlock.c:379)
 tcp_v4_rcv (./include/linux/skbuff.h:1678 ./include/net/tcp.h:2547 net/ipv4/tcp_ipv4.c:2350)
 ip_protocol_deliver_rcu (net/ipv4/ip_input.c:205 (discriminator 1))
 ip_local_deliver_finish (./include/linux/rcupdate.h:878 net/ipv4/ip_input.c:234)
 ip_sublist_rcv_finish (net/ipv4/ip_input.c:576)
 ip_list_rcv_finish (net/ipv4/ip_input.c:628)
 ip_list_rcv (net/ipv4/ip_input.c:670)
 __netif_receive_skb_list_core (net/core/dev.c:5939 net/core/dev.c:5986)
 netif_receive_skb_list_internal (net/core/dev.c:6040 net/core/dev.c:6129)
 napi_complete_done (./include/linux/list.h:37 ./include/net/gro.h:519 ./include/net/gro.h:514 net/core/dev.c:6496)
 e1000_clean (drivers/net/ethernet/intel/e1000/e1000_main.c:3815)
 __napi_poll.constprop.0 (net/core/dev.c:7191)
 net_rx_action (net/core/dev.c:7262 net/core/dev.c:7382)
 handle_softirqs (kernel/softirq.c:561)
 __irq_exit_rcu (kernel/softirq.c:596 kernel/softirq.c:435 kernel/softirq.c:662)
 irq_exit_rcu (kernel/softirq.c:680)
 common_interrupt (arch/x86/kernel/irq.c:280 (discriminator 14))
  </IRQ>
 <TASK>
 asm_common_interrupt (./arch/x86/include/asm/idtentry.h:693)
RIP: 0010:default_idle (./arch/x86/include/asm/irqflags.h:37 ./arch/x86/include/asm/irqflags.h:92 arch/x86/kernel/process.c:744)
Code: 4c 01 c7 4c 29 c2 e9 72 ff ff ff 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 f3 0f 1e fa eb 07 0f 00 2d c3 2b 15 00 fb f4 <fa> c3 cc cc cc
RSP: 0018:ffa00000000ffee8 EFLAGS: 00000202
RAX: 000000000000640b RBX: ff1100010091c200 RCX: 0000000000061aa4
RDX: 0000000000000000 RSI: 0000000000000000 RDI: ffffffff812f30c5
RBP: 000000000000000a R08: 0000000000000001 R09: 0000000000000000
R10: 0000000000000001 R11: 0000000000000002 R12: 0000000000000000
R13: 0000000000000000 R14: 0000000000000000 R15: 0000000000000000
 ? do_idle (kernel/sched/idle.c:186 kernel/sched/idle.c:325)
 default_idle_call (./include/linux/cpuidle.h:143 kernel/sched/idle.c:118)
 do_idle (kernel/sched/idle.c:186 kernel/sched/idle.c:325)
 cpu_startup_entry (kernel/sched/idle.c:422 (discriminator 1))
 start_secondary (arch/x86/kernel/smpboot.c:315)
 common_startup_64 (arch/x86/kernel/head_64.S:421)
 </TASK>
Modules linked in: cifs_arc4 nls_ucs2_utils cifs_md4 [last unloaded: cifs]
CR2: 00000000000000c4

Fixes: ed07536ed673 ("[PATCH] lockdep: annotate nfs/nfsd in-kernel sockets")
Signed-off-by: Kuniyuki Iwashima <kuniyu@amazon.com>
Cc: stable@vger.kernel.org
Link: https://patch.msgid.link/20250407163313.22682-1-kuniyu@amazon.com
Signed-off-by: Jakub Kicinski <kuba@kernel.org>
Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
```

Diffstat

-rw-r--r-- include/net/sock.h	40

-rw-r--r-- net/core/sock.c 5

2 files changed, 43 insertions, 2 deletions

```
diff --git a/include/net/sock.h b/include/net/sock.h
index fa055cf1785efd..fa9b9dadbe1709 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -338,6 +338,8 @@ struct sk_filter;
   *    @sk_txtime_unused: unused txtime flags
   *    @ns_tracker: tracker for netns reference
   *    @sk_user_frags: xarray of pages the user is holding a reference on.
+  *    @sk_owner: reference to the real owner of the socket that calls
+  *               sock_lock_init_class_and_name().
   */
 struct sock {
        /*
@@ -544,6 +546,10 @@ struct sock {
        struct rcu_head         sk_rcu;
        netns_tracker           ns_tracker;
        struct xarray           sk_user_frags;
+
+#if IS_ENABLED(CONFIG_PROVE_LOCKING) && IS_ENABLED(CONFIG_MODULES)
+       struct module           *sk_owner;
+#endif
 };

 struct sock_bh_locked {
@@ -1585,6 +1591,35 @@ static inline void sk_mem_uncharge(struct sock *sk, int size)
        sk_mem_reclaim(sk);
 }

+#if IS_ENABLED(CONFIG_PROVE_LOCKING) && IS_ENABLED(CONFIG_MODULES)
+static inline void sk_owner_set(struct sock *sk, struct module *owner)
+{
+       __module_get(owner);
+       sk->sk_owner = owner;
+}
+
+static inline void sk_owner_clear(struct sock *sk)
+{
+       sk->sk_owner = NULL;
+}
+
+static inline void sk_owner_put(struct sock *sk)
+{
+       module_put(sk->sk_owner);
+}
+#else
+static inline void sk_owner_set(struct sock *sk, struct module *owner)
+{
+}
+
+static inline void sk_owner_clear(struct sock *sk)
+{
+}
+
+static inline void sk_owner_put(struct sock *sk)
+{
+}
+#endif
 /*
  * Macro so as to not evaluate some arguments when
  * lockdep is not enabled.
@@ -1594,13 +1629,14 @@ static inline void sk_mem_uncharge(struct sock *sk, int size)
  */
 #define sock_lock_init_class_and_name(sk, sname, skey, name, key)      \
 do {                                                                   \
+       sk_owner_set(sk, THIS_MODULE);                                  \
        sk->sk_lock.owned = 0;                                          \
        init_waitqueue_head(&sk->sk_lock.wq);                           \
        spin_lock_init(&(sk)->sk_lock.slock);                           \
        debug_check_no_locks_freed((void *)&(sk)->sk_lock,              \
-                       sizeof((sk)->sk_lock));                         \
+                                  sizeof((sk)->sk_lock));              \
        lockdep_set_class_and_name(&(sk)->sk_lock.slock,                \
-                               (skey), (sname));                       \
+                                  (skey), (sname));                    \
        lockdep_init_map(&(sk)->sk_lock.dep_map, (name), (key), 0);     \
 } while (0)


diff --git a/net/core/sock.c b/net/core/sock.c
index a83f64a1d96a29..0842dc9189bf80 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -2107,6 +2107,8 @@ lenout:
  */
 static inline void sock_lock_init(struct sock *sk)
 {
+       sk_owner_clear(sk);
+
        if (sk->sk_kern_sock)
                sock_lock_init_class_and_name(
```

```
                        sk,
@@ -2203,6 +2205,9 @@ static void sk_prot_free(struct proto *prot, struct sock *sk)
        cgroup_sk_free(&sk->sk_cgrp_data);
        mem_cgroup_sk_free(sk);
        security_sk_free(sk);
+
+       sk_owner_put(sk);
+
        if (slab != NULL)
                kmem_cache_free(slab, sk);
        else
```