PANGU TEAM

Search

MPTCP Integer Overflow Vulnerability

int80 August 3, 2020 No Comments

In this blog, we will share an integer overflow vulnerability in the MPTCP module in the XNU kernel.

When we started to study MPTCP, we got a very brief description from the official document:

> “MPTCP is a set of extensions to the Transmission Control Protocol (TCP) specification. With MPTCP, a client can connect to the same destination host with multiple connections over different network adapters”.

Now a natural question comes into our mind: how many connections can a client connect to a host at most? With this question in mind, we created a simple test program that simply creates an MPTCP socket and connects to a host many times. Our purpose is to figure out when we cannot create new connections.

The test program ran fine. However, the surprise thing was that we triggered a kernel panic when the test program exited.

The test program was so simple that we had no clue about what triggered the panic. After analyzing the panic log, we realized that our program triggered a recursive kernel function and resulted in a kernel stack exhaustion when the MPTCP socket was closed. Note that the recursive function panic was also already fixed. You won’t be able to trigger it on iOS 13.

We continued our testing process. Now, we turned to the XNU source code. We quickly found the following data structure.

```
struct mptses {
        struct mppcb    *mpte_mppcb;            /* back ptr to multipath PCB */
        struct mptcb    *mpte_mptcb;            /* ptr to MPTCP PCB */
        TAILQ_HEAD(, mptopt) mpte_sopts;        /* list of socket options */
        TAILQ_HEAD(, mptsub) mpte_subflows;     /* list of subflows */
        uint16_t        mpte_numflows;          /* # of subflows in list */
        uint16_t        mpte_nummpcapflows;     /* # of MP_CAP subflows */
        sae_associd_t   mpte_associd;           /* MPTCP association ID */
        sae_connid_t    mpte_connid_last;       /* last used connection ID */
...
```

`mptses` represents MPTCP sessions. Every time a new connection is created between a client and a host, there will be a new `mpte_subflow` created. `mptses->mpte_numflows` records the number of subflows.

```
static void
mptcp_subflow_attach(struct mptses *mpte, struct mptsub *mpts, struct socket *sc
{
        struct socket *mp_so = mpte->mpte_mppcb->mpp_socket;
        struct tcpcb *tp = sototcpcb(so);

...
        /*
         * Insert the subflow into the list, and associate the MPTCP PCB
         * as well as the the subflow socket.  From this point on, removing
         * the subflow needs to be done via mptcp_subflow_del().
         */
        TAILQ_INSERT_TAIL(&mpte->mpte_subflows, mpts, mpts_entry);
        mpte->mpte_numflows++; //<====== no integer overflow checks
```

As we can see in function `mptcp_subflow_attach` , creating a new connection will increase `mpte->mpte_numflows` by one, but there is no integer overflow checks at all.

You may also notice that, `mpte_numflows` is in the type of uint16_t, which means its maximum value is 0xFFFF! So what if we create 0xFFFF+2 connections? The answer is that `mpte_numflows` will wrap to 1!

So far, the integer overflow doesn't cause any memory errors. We continued to check how `mpte_numflows` would be used. Just by greping mpte_numflows, we got the following sysctl handler: `mptcp_pcblist`

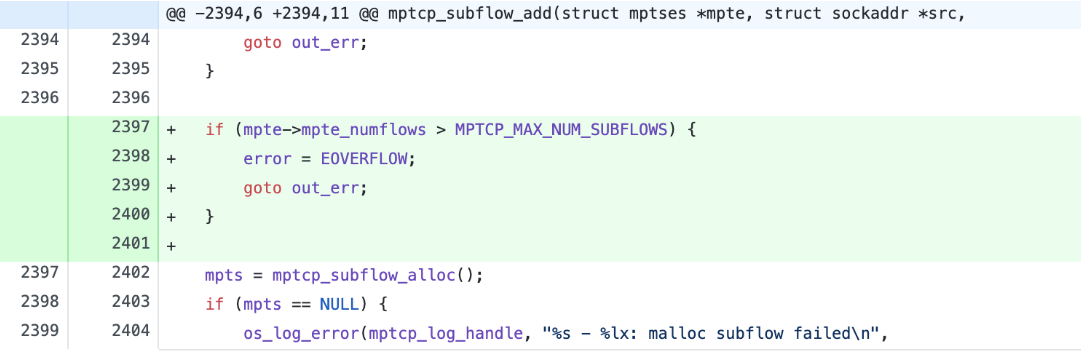
```
 static int
mptcp_pcblist SYSCTL_HANDLER_ARGS
{
...
        TAILQ_FOREACH(mpp, &mtcbinfo.mppi_pcbs, mpp_entry) {
                flows = NULL;
                socket_lock(mpp->mpp_socket, 1);
                VERIFY(mpp->mpp_flags & MPP_ATTACHED);
                mpte = mptompte(mpp);

...
                mptcpci.mptcpci_nflows = mpte->mpte_numflows;
...
                len = sizeof(*flows) * mpte->mpte_numflows;
                if (mpte->mpte_numflows != 0) {
                        flows = _MALLOC(len, M_TEMP, M_WAITOK | M_ZERO);
//<=== alloc memory according to mpte->mpte_numflows
...
                f = 0;
                TAILQ_FOREACH(mpts, &mpte->mpte_subflows, mpts_entry) {
                        so = mpts->mpts_socket;
                        fill_mptcp_subflow(so, &flows[f], mpts);
 <== dump the list into flows buffer. HEAP OVERFLOW!
                        f++;
```

In function `mptcp_pcblist` , `mpte_numflows` is used to calculate the length of a temp buffer. If we already make `mpte_numflows` wrapped to 1, the allocation site will only allocate ONE entry. However, `mptcp_pcblist` will traverse the list `mpte_subflows` and dump all the entries into the allocated buffer. Heap overflow happens!

We won't get into the exploitation phase. With partially controlled values and partially controlled length, the exploitation would be also very interesting.

Fixing the issue is quite easy. The patch is as follows. `mptcp_subflow_add` function now adds a limitation to `mpte_numflows` .



```
@@ -2394,6 +2394,11 @@ mptcp_subflow_add(struct mptses *mpte, struct sockaddr *src,
         goto out_err;
     }
 
+    if (mpte->mpte_numflows > MPTCP_MAX_NUM_SUBFLOWS) {
+        error = EOVERFLOW;
+        goto out_err;
+    }
+
     mpts = mptcp_subflow_alloc();
     if (mpts == NULL) {
         os_log_error(mptcp_log_handle, "%s - %lx: malloc subflow failed\n",
```

Do you still remember the question at the beginning? How many connections does an MPTCP socket allow? Now, we got the answer:

```
#define MPTCP_MAX_NUM_SUBFLOWS 256
```

Credit: The integer overflow was discovered and analyzed by Tao Huang and Tielei Wang of Pangu Lab.

Thanks for reading!

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

□

Save my name, email, and website in this browser for the next time I comment.

Post Comment