author	Paolo Abeni <pabeni@redhat.com>	2023-03-23 18:49:01 +0100
committer	Greg Kroah-Hartman <gregkh@linuxfoundation.org>	2023-03-30 12:49:00 +0200
commit	2827f099b3fb9a59263c997400e9182f5d423e84 (patch)	
tree	7191a7abe6eea62f4caef030f16cf37d03fadb83	
parent	1516ddbc34bcba56ee09f77740b8d67a391d8f24 (diff)	
download	linux-2827f099b3fb9a59263c997400e9182f5d423e84.tar.gz	

mptcp: use the workqueue to destroy unaccepted sockets

```
[ Upstream commit b6985b9b82954caa53f862d6059d06c0526254f0 ]

  Backports notes: one simple conflict in net/mptcp/protocol.c with:

    commit a5ef058dc4d9 ("net: introduce and use custom sockopt socket flag")

  Where the two commits add a new line for different actions in the same
  context in mptcp_stream_accept().

Christoph reported a UaF at token lookup time after having
refactored the passive socket initialization part:

  BUG: KASAN: use-after-free in __token_bucket_busy+0x253/0x260
  Read of size 4 at addr ffff88810698d5b0 by task syz-executor653/3198

  CPU: 1 PID: 3198 Comm: syz-executor653 Not tainted 6.2.0-rc59af4eaa31c1f6c00c8f1e448ed99a45c66340dd5 #6
  Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.13.0-0-gf21b5a4aeb02-prebuilt.qemu.org 04/01/2014
  Call Trace:
   <TASK>
   dump_stack_lvl+0x6e/0x91
   print_report+0x16a/0x46f
   kasan_report+0xad/0x130
   __token_bucket_busy+0x253/0x260
   mptcp_token_new_connect+0x13d/0x490
   mptcp_connect+0x4ed/0x860
   __inet_stream_connect+0x80e/0xd90
   tcp_sendmsg_fastopen+0x3ce/0x710
   mptcp_sendmsg+0xff1/0x1a20
   inet_sendmsg+0x11d/0x140
   __sys_sendto+0x405/0x490
   __x64_sys_sendto+0xdc/0x1b0
   do_syscall_64+0x3b/0x90
   entry_SYSCALL_64_after_hwframe+0x72/0xdc

We need to properly clean-up all the paired MPTCP-level
resources and be sure to release the msk last, even when
the unaccepted subflow is destroyed by the TCP internals
via inet_child_forget().

We can re-use the existing MPTCP_WORK_CLOSE_SUBFLOW infra,
explicitly checking that for the critical scenario: the
closed subflow is the MPC one, the msk is not accepted and
eventually going through full cleanup.

With such change, __mptcp_destroy_sock() is always called
on msk sockets, even on accepted ones. We don't need anymore
to transiently drop one sk reference at msk clone time.

Please note this commit depends on the parent one:

  mptcp: refactor passive socket initialization

Fixes: 58b09919626b ("mptcp: create msk early")
Cc: stable@vger.kernel.org
```

Reported-and-tested-by: Christoph Paasch <cpaasch@apple.com>
Closes: https://github.com/multipath-tcp/mptcp_net-next/issues/347
Signed-off-by: Paolo Abeni <pabeni@redhat.com>
Reviewed-by: Matthieu Baerts <matthieu.baerts@tessares.net>
Signed-off-by: Matthieu Baerts <matthieu.baerts@tessares.net>
Signed-off-by: Jakub Kicinski <kuba@kernel.org>
Signed-off-by: Matthieu Baerts <matthieu.baerts@tessares.net>
Signed-off-by: Sasha Levin <sashal@kernel.org>

Diffstat

-rw-r--r-- net/mptcp/protocol.c 41
-rw-r--r-- net/mptcp/protocol.h 5
-rw-r--r-- net/mptcp/subflow.c 17

3 files changed, 47 insertions, 16 deletions

```
diff --git a/net/mptcp/protocol.c b/net/mptcp/protocol.c
index 777f795246ed28..b679e8a430a83e 100644
--- a/net/mptcp/protocol.c
+++ b/net/mptcp/protocol.c
@@ -2357,7 +2357,6 @@ static void __mptcp_close_ssk(struct sock *sk, struct sock *ssk,
                goto out;
        }

-       sock_orphan(ssk);
        subflow->disposable = 1;

        /* if ssk hit tcp_done(), tcp_cleanup_ulp() cleared the related ops
@@ -2365,7 +2364,20 @@ static void __mptcp_close_ssk(struct sock *sk, struct sock *ssk,
         * reference owned by msk;
         */
        if (!inet_csk(ssk)->icsk_ulp_ops) {
+               WARN_ON_ONCE(!sock_flag(ssk, SOCK_DEAD));
                kfree_rcu(subflow, rcu);
+       } else if (msk->in_accept_queue && msk->first == ssk) {
+               /* if the first subflow moved to a close state, e.g. due to
+                * incoming reset and we reach here before inet_child_forget()
+                * the TCP stack could later try to close it via
+                * inet_csk_listen_stop(), or deliver it to the user space via
+                * accept().
+                * We can't delete the subflow - or risk a double free - nor let
+                * the msk survive - or will be leaked in the non accept scenario:
+                * fallback and let TCP cope with the subflow cleanup.
+                */
+               WARN_ON_ONCE(sock_flag(ssk, SOCK_DEAD));
+               mptcp_subflow_drop_ctx(ssk);
        } else {
                /* otherwise tcp will dispose of the ssk and subflow ctx */
                if (ssk->sk_state == TCP_LISTEN) {
@@ -2412,9 +2424,10 @@ static unsigned int mptcp_sync_mss(struct sock *sk, u32 pmtu)
        return 0;
 }

-static void __mptcp_close_subflow(struct mptcp_sock *msk)
+static void __mptcp_close_subflow(struct sock *sk)
 {
        struct mptcp_subflow_context *subflow, *tmp;
+       struct mptcp_sock *msk = mptcp_sk(sk);

        might_sleep();

@@ -2428,7 +2441,15 @@ static void __mptcp_close_subflow(struct mptcp_sock *msk)
                        continue;

-               mptcp_close_ssk((struct sock *)msk, ssk, subflow);
+               mptcp_close_ssk(sk, ssk, subflow);
+       }
+
+       /* if the MPC subflow has been closed before the msk is accepted,
+        * msk will never be accept-ed, close it now
+        */
+       if (!msk->first && msk->in_accept_queue) {
+               sock_set_flag(sk, SOCK_DEAD);
```

```
+               inet_sk_state_store(sk, TCP_CLOSE);
        }
 }

@@ -2637,6 +2658,9 @@ static void mptcp_worker(struct work_struct *work)
        __mptcp_check_send_data_fin(sk);
        mptcp_check_data_fin(sk);

+       if (test_and_clear_bit(MPTCP_WORK_CLOSE_SUBFLOW, &msk->flags))
+               __mptcp_close_subflow(sk);
+
        /* There is no point in keeping around an orphaned sk timedout or
         * closed, but we need the msk around to reply to incoming DATA_FIN,
         * even if it is orphaned and in FIN_WAIT2 state
@@ -2652,9 +2676,6 @@ static void mptcp_worker(struct work_struct *work)
                }
        }

-       if (test_and_clear_bit(MPTCP_WORK_CLOSE_SUBFLOW, &msk->flags))
-               __mptcp_close_subflow(msk);
-
        if (test_and_clear_bit(MPTCP_WORK_RTX, &msk->flags))
                __mptcp_retrans(sk);

@@ -3084,6 +3105,7 @@ struct sock *mptcp_sk_clone(const struct sock *sk,
        msk->local_key = subflow_req->local_key;
        msk->token = subflow_req->token;
        msk->subflow = NULL;
+       msk->in_accept_queue = 1;
        WRITE_ONCE(msk->fully_established, false);
        if (mp_opt->suboptions & OPTION_MPTCP_CSUMREQD)
                WRITE_ONCE(msk->csum_enabled, true);
@@ -3110,8 +3132,7 @@ struct sock *mptcp_sk_clone(const struct sock *sk,
        security_inet_csk_clone(nsk, req);
        bh_unlock_sock(nsk);

-       /* keep a single reference */
-       __sock_put(nsk);
+       /* note: the newly allocated socket refcount is 2 now */
        return nsk;
 }

@@ -3167,8 +3188,6 @@ static struct sock *mptcp_accept(struct sock *sk, int flags, int *err,
                        goto out;
                }

-               /* acquire the 2nd reference for the owning socket */
-               sock_hold(new_mptcp_sock);
                newsk = new_mptcp_sock;
                MPTCP_INC_STATS(sock_net(sk), MPTCP_MIB_MPCAPABLEPASSIVEACK);
        } else {
@@ -3726,6 +3745,8 @@ static int mptcp_stream_accept(struct socket *sock, struct socket *newsock,
                struct mptcp_subflow_context *subflow;
                struct sock *newsk = newsock->sk;

+               msk->in_accept_queue = 0;
+
                lock_sock(newsk);

                /* set ssk->sk_socket of accept()ed flows to mptcp socket.
```

diff --git a/net/mptcp/protocol.h b/net/mptcp/protocol.h
index 6f22ae13c98482..2cddd5b52e8fab 100644
--- a/net/mptcp/protocol.h
+++ b/net/mptcp/protocol.h

```
@@ -286,7 +286,8 @@ struct mptcp_sock {
        u8              recvmsg_inq:1,
                        cork:1,
                        nodelay:1,
-                       fastopening:1;
+                       fastopening:1,
+                       in_accept_queue:1;
        int             connect_flags;
        struct work_struct work;
```

```
        struct sk_buff  *ooo_last_skb;
@@ -651,6 +652,8 @@ void mptcp_subflow_set_active(struct mptcp_subflow_context *subflow);

 bool mptcp_subflow_active(struct mptcp_subflow_context *subflow);

+void mptcp_subflow_drop_ctx(struct sock *ssk);
+
 static inline void mptcp_subflow_tcp_fallback(struct sock *sk,
                                               struct mptcp_subflow_context *ctx)
 {
```

diff --git a/net/mptcp/subflow.c b/net/mptcp/subflow.c
index fe815103060c66..459621a0410cda 100644
--- a/net/mptcp/subflow.c
+++ b/net/mptcp/subflow.c

```
@@ -636,9 +636,10 @@ static bool subflow_hmac_valid(const struct request_sock *req,

 static void mptcp_force_close(struct sock *sk)
 {
-       /* the msk is not yet exposed to user-space */
+       /* the msk is not yet exposed to user-space, and refcount is 2 */
        inet_sk_state_store(sk, TCP_CLOSE);
        sk_common_release(sk);
+       sock_put(sk);
 }

 static void subflow_ulp_fallback(struct sock *sk,
@@ -654,7 +655,7 @@ static void subflow_ulp_fallback(struct sock *sk,
        mptcp_subflow_ops_undo_override(sk);
 }

-static void subflow_drop_ctx(struct sock *ssk)
+void mptcp_subflow_drop_ctx(struct sock *ssk)
 {
        struct mptcp_subflow_context *ctx = mptcp_subflow_ctx(ssk);

@@ -758,7 +759,7 @@ create_child:

                        if (new_msk)
                                mptcp_copy_inaddrs(new_msk, child);
-                       subflow_drop_ctx(child);
+                       mptcp_subflow_drop_ctx(child);
                        goto out;
                }

@@ -849,7 +850,7 @@ out:
        return child;

 dispose_child:
-       subflow_drop_ctx(child);
+       mptcp_subflow_drop_ctx(child);
        tcp_rsk(req)->drop_req = true;
        inet_csk_prepare_for_destroy_sock(child);
        tcp_done(child);
@@ -1804,7 +1805,6 @@ void mptcp_subflow_queue_clean(struct sock *listener_sk, struct sock *listener_s
                struct sock *sk = (struct sock *)msk;
                bool do_cancel_work;

-               sock_hold(sk);
                lock_sock_nested(sk, SINGLE_DEPTH_NESTING);
                next = msk->dl_next;
                msk->first = NULL;
@@ -1892,6 +1892,13 @@ static void subflow_ulp_release(struct sock *ssk)
                 * when the subflow is still unaccepted
                 */
                release = ctx->disposable || list_empty(&ctx->node);
+
+               /* inet_child_forget() does not call sk_state_change(),
+                * explicitly trigger the socket close machinery
+                */
+               if (!release && !test_and_set_bit(MPTCP_WORK_CLOSE_SUBFLOW,
+                                                 &mptcp_sk(sk)->flags))
+                       mptcp_schedule_work(sk);
                sock_put(sk);
```

```
}
```