From: Daniel Borkmann <daniel@iogearbox.net>
To: ast@kernel.org
Cc: andrii@kernel.org, bpf@vger.kernel.org,
        Daniel Borkmann <daniel@iogearbox.net>,
        Hsin-Wei Hung <hsinweih@uci.edu>,
        Shung-Hsi Yu <shung-hsi.yu@suse.com>
Subject: [PATCH bpf] bpf: Don't use tnum_range on array range checking for poke descriptors
Date: Thu, 25 Aug 2022 23:26:47 +0200 [thread overview]
Message-ID: <984b37f9fdf7ac36831d2137415a4a915744c1b6.1661462653.git.daniel@iogearbox.net>
(raw)

Hsin-Wei reported a KASAN splat triggered by their BPF runtime fuzzer which
is based on a customized syzkaller:

```
  BUG: KASAN: slab-out-of-bounds in bpf_int_jit_compile+0x1257/0x13f0
  Read of size 8 at addr ffff888004e90b58 by task syz-executor.0/1489
  CPU: 1 PID: 1489 Comm: syz-executor.0 Not tainted 5.19.0 #1
  Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS
  1.13.0-1ubuntu1.1 04/01/2014
  Call Trace:
   <TASK>
   dump_stack_lvl+0x9c/0xc9
   print_address_description.constprop.0+0x1f/0x1f0
   ? bpf_int_jit_compile+0x1257/0x13f0
   kasan_report.cold+0xeb/0x197
   ? kvmalloc_node+0x170/0x200
   ? bpf_int_jit_compile+0x1257/0x13f0
   bpf_int_jit_compile+0x1257/0x13f0
   ? arch_prepare_bpf_dispatcher+0xd0/0xd0
   ? rcu_read_lock_sched_held+0x43/0x70
   bpf_prog_select_runtime+0x3e8/0x640
   ? bpf_obj_name_cpy+0x149/0x1b0
   bpf_prog_load+0x102f/0x2220
   ? __bpf_prog_put.constprop.0+0x220/0x220
   ? find_held_lock+0x2c/0x110
   ? __might_fault+0xd6/0x180
   ? lock_downgrade+0x6e0/0x6e0
   ? lock_is_held_type+0xa6/0x120
   ? __might_fault+0x147/0x180
   __sys_bpf+0x137b/0x6070
   ? bpf_perf_link_attach+0x530/0x530
   ? new_sync_read+0x600/0x600
   ? __fget_files+0x255/0x450
   ? lock_downgrade+0x6e0/0x6e0
   ? fput+0x30/0x1a0
   ? ksys_write+0x1a8/0x260
   __x64_sys_bpf+0x7a/0xc0
   ? syscall_enter_from_user_mode+0x21/0x70
   do_syscall_64+0x3b/0x90
   entry_SYSCALL_64_after_hwframe+0x63/0xcd
  RIP: 0033:0x7f917c4e2c2d
```

The problem here is that a range of tnum_range(0, map->max_entries - 1) has
limited ability to represent the concrete tight range with the tnum as the
set of resulting states from value + mask can result in a superset of the
actual intended range, and as such a tnum_in(range, reg->var_off) check may
yield true when it shouldn't, for example tnum_range(0, 2) would result in
00XX -> v = 0000, m = 0011 such that the intended set of {0, 1, 2} is here
represented by a less precise superset of {0, 1, 2, 3}. As the register is
known const scalar, really just use the concrete reg->var_off.value for the
upper index check.

```
Fixes: d2e4c1e6c294 ("bpf: Constant map key tracking for prog array pokes")
Reported-by: Hsin-Wei Hung <hsinweih@uci.edu>
Signed-off-by: Daniel Borkmann <daniel@iogearbox.net>
Cc: Shung-Hsi Yu <shung-hsi.yu@suse.com>
---
 kernel/bpf/verifier.c | 10 ++++------
 1 file changed, 4 insertions(+), 6 deletions(-)

diff --git a/kernel/bpf/verifier.c b/kernel/bpf/verifier.c
index 30c6eebce146..3eadb14e090b 100644
--- a/kernel/bpf/verifier.c
+++ b/kernel/bpf/verifier.c
@@ -7033,8 +7033,7 @@ record_func_key(struct bpf_verifier_env *env, struct bpf_call_arg_meta
*meta,
        struct bpf_insn_aux_data *aux = &env->insn_aux_data[insn_idx];
        struct bpf_reg_state *regs = cur_regs(env), *reg;
        struct bpf_map *map = meta->map_ptr;
-       struct tnum range;
-       u64 val;
+       u64 val, max;
        int err;

        if (func_id != BPF_FUNC_tail_call)
@@ -7044,10 +7043,11 @@ record_func_key(struct bpf_verifier_env *env, struct
bpf_call_arg_meta *meta,
                return -EINVAL;
        }

-       range = tnum_range(0, map->max_entries - 1);
        reg = &regs[BPF_REG_3];
+       val = reg->var_off.value;
+       max = map->max_entries;

-       if (!register_is_const(reg) || !tnum_in(range, reg->var_off)) {
+       if (!(register_is_const(reg) && val < max)) {
                bpf_map_key_store(aux, BPF_MAP_KEY_POISON);
                return 0;
        }
@@ -7055,8 +7055,6 @@ record_func_key(struct bpf_verifier_env *env, struct bpf_call_arg_meta
*meta,
        err = mark_chain_precision(env, BPF_REG_3);
        if (err)
                return err;
-
-       val = reg->var_off.value;
        if (bpf_map_key_unseen(aux))
                bpf_map_key_store(aux, val);
        else if (!bpf_map_key_poisoned(aux) &&
--
2.21.0
```

---

next reply other threads:[~2022-08-25 21:27 UTC|newest]

Thread overview: 3+ messages / expand[flat|nested] mbox.gz Atom feed top
2022-08-25 21:26 Daniel Borkmann [this message]
2022-08-25 21:53 ` [PATCH bpf] bpf: Don't use tnum_range on array range checking for poke descriptors John Fastabend
2022-08-25 22:00 ` patchwork-bot+netdevbpf

find likely ancestor, descendant, or conflicting patches for this message:
dfblob:30c6eebce14 dfblob:3eadb14e090

search (help)

Reply instructions:

You may reply publicly to this message via plain-text email
using any one of the following methods:

* Save the following mbox file, import it into your mail client,
  and reply-to-all from there: mbox

  Avoid top-posting and favor interleaved quoting:
  https://en.wikipedia.org/wiki/Posting_style#Interleaved_style

* Reply using the **--to**, **--cc**, and **--in-reply-to**
  switches of git-send-email(1):

```
  git send-email \
    --in-reply-
to=984b37f9fdf7ac36831d2137415a4a915744c1b6.1661462653.git.daniel@iogearbox.net \
    --to=daniel@iogearbox.net \
    --cc=andrii@kernel.org \
    --cc=ast@kernel.org \
    --cc=bpf@vger.kernel.org \
    --cc=hsinweih@uci.edu \
    --cc=shung-hsi.yu@suse.com \
    /path/to/YOUR_REPLY
```

  https://kernel.org/pub/software/scm/git/docs/git-send-email.html

* If your mail client supports setting the **In-Reply-To** header
  via mailto: links, try the mailto: link

Be sure your reply has a **Subject:** header at the top and a blank line before the message body.

This is a public inbox, see mirroring instructions
for how to clone and mirror all data and code used for this inbox;
as well as URLs for NNTP newsgroup(s).