# Cgroup-bpf Production Surprises

Martin Lau
Kernel Software Engineer

Meta

# cgroup g/setsockopt (a bpf prog)

- bpf prog is only interested to a few optnames

```
#define SO_INTERNAL_OPT_XYZ 9876

SEC("cgroup/getsockopt")
int interal_optname(struct bpf_sockopt* ctx) {
   __u8 *storage, *optval = ctx->optval;

   if (ctx->level == SOL_SOCKET && ctx->optname == SO_INTERNAL_OPT_XYZ) {
      storage = bpf_sk_storage_get(&sk_map, ctx->sk, NULL, 0);
      if (!storage || optval + 1 > ctx->optval_end)
         return 0;
      *optval = *storage;
      ctx->optlen = 1;
      ctx->retval = 0;
   }

   /* Use kernel getsockopt for other optnames */

   return 1;
}
```

# cgroup g/setsockopt (oops)

- All works well until one day a random service did this:

```
getsockopt(4, SOL_IPV6, IP6T_SO_GET_ENTRIES, 0x561e7b280680, [8192]) = -1
EFAULT (Bad address)
```

- EFAULT only happens on optlen > PAGE_SIZE (4096).

# cgroup g/setsockopt (a fixed bpf prog)

```
#define SO_INTERNAL_OPT_XYZ 9876

SEC("cgroup/getsockopt")
int interal_optname(struct bpf_sockopt* ctx) {
   __u8 *storage, *optval = ctx->optval;

   if (ctx->level == SOL_SOCKET && ctx->optname == SO_INTERNAL_OPT_XYZ) {
      storage = bpf_sk_storage_get(&sk_map, ctx->sk, NULL, 0);
      if (!storage || optval + 1 > ctx->optval_end)
         return 0;
      *optval = *storage;
      ctx->optlen = 1;
      ctx->retval = 0;
   } else {
      /* Only reset optlen > PAGE_SIZE such that other bpf prog
       * has a chance to look at optlen (and optval).
       */
      if (ctx->optlen > PAGE_SIZE)
        ctx->optlen = 0;
   }
   return 1;
}
```

# cgroup g/setsockopt (kernel details)

```
int __cgroup_bpf_run_filter_getsockopt(struct sock *sk, int level,
                                       int optname, char __user *optval,
                                       int __user *optlen, int max_optlen,
                                       int retval)
{
   /* ctx.optlen at 8192 */
   ctx.optlen = max_optlen;

   /* Allocate kernel memory for bpf prog to read and write.
    * The alloc size to PAGE_SIZE.
    */
   /* max_optlen at 4096 */
   max_optlen = sockopt_alloc_buf(&ctx, max_optlen, &buf);

   ret = bpf_prog_run_array_cg(..., &ctx, ...);

   /* ctx.optlen (8192) > max_optlen (4096) */
   if (optval && (ctx.optlen > max_optlen || ctx.optlen < 0)) {
      ret = -EFAULT;
      goto out;
   }
}
```

# cgroup g/setsockopt (a relief fix)

- Do not -EFAULT if the original optlen > PAGE_SIZE: https://lore.kernel.org/bpf/20230504184349.3632259-1-sdf@google.com/

# cgroup getsockopt/setsockopt (Better UX)

- Why kmalloc? For non-sleepable cgroup-bpf to read/write the __user optval.
- PAGE_SIZE to limit the alloc (and memcpy)
- Can this alloc (and memcpy) be avoided?
- Have the bpf prog directly read from the __user optval
  - Made cgroup-bpf sleepable. The newer lsm-cgroup is sleepable.

# cgroup g/setsockopt (Better UX)

- What if bpf needs to change the optval?
- For getsockopt, directly write to __user optval?
- For setsockopt, what if the bpf prog wants to write optval longer than the __user optval? A kmalloc is still needed.
- Does it make sense to do all this as the dynptr API?

# cgroup sockops

- bpf hooks in the tcp stack
- tp->bpf_sock_ops_cb_flags to control if the bpf prog needs to be called or not

# cgroup sockops

```
enum {
        BPF_SOCK_OPS_RTO_CB_FLAG          = (1<<0),
        BPF_SOCK_OPS_RETRANS_CB_FLAG      = (1<<1),
        BPF_SOCK_OPS_STATE_CB_FLAG        = (1<<2),
        BPF_SOCK_OPS_RTT_CB_FLAG          = (1<<3),
        BPF_SOCK_OPS_PARSE_ALL_HDR_OPT_CB_FLAG   = (1<<4),
        BPF_SOCK_OPS_PARSE_UNKNOWN_HDR_OPT_CB_FLAG = (1<<5),
        BPF_SOCK_OPS_WRITE_HDR_OPT_CB_FLAG = (1<<6),
};
```

# cgroup sockops

```
/* Enable write header option */
bpf_sock_ops_cb_flags_set(sockops,
            sockops->bpf_sock_ops_cb_flags |
            BPF_SOCK_OPS_WRITE_HDR_OPT_CB_FLAG)

/* Disable write header option */
bpf_sock_ops_cb_flags_set(sockops,
            sockops->bpf_sock_ops_cb_flags &
            ~BPF_SOCK_OPS_WRITE_HDR_OPT_CB_FLAG)
```

# Two sockops programs

```
SEC("sockops")
int prog_a(struct bpf_sock_ops *skops)
{
    switch (skops->op) {
    case BPF_SOCK_OPS_TCP_LISTEN_CB:
        /* turn on WRITE_HDR_OPT_CB_FLAG */
        break;
    case BPF_SOCK_OPS_PASSIVE_ESTABLISHED_CB:
        /* Contd to keep WRITE_HDR_OPT_CB_FLAG */
    }
}
```

```
SEC("sockops")
int prog_b(struct bpf_sock_ops *skops)
{
    switch (skops->op) {
    case BPF_SOCK_OPS_TCP_LISTEN_CB:
        /* turn on WRITE_HDR_OPT_CB_FLAG */
        break;
    case BPF_SOCK_OPS_PASSIVE_ESTABLISHED_CB:
        /* Turn off WRITE_HDR_OPT_CB_FLAG
         * Oops. prog_a will no longer be
         * able to write hdr.
         */
    }
}
```

# cgroup sockops (workaround in bpf prog)

- Once a cb_flags is turned on, it is left on forever. Need bpf progs to behave.
- The bpf prog stores a bool in its bpf_sk_storage to flag if it needs to process a cb or just return.
- Ideas?


Meta