LINUX
PLUMBERS
CONFERENCE
August 24-28, 2020

seccomp feature development

or, “soon seccomp will include an email client”

Kees (“Case”) Cook
keescook@chromium.org
@kees_cook

areas of development

https://lore.kernel.org/lkml/202005181120.971232B7B@keescook

- fd injection
- syscall bitmaps
- deep argument inspection
- changing structure sizes
- other things

fd injection (`SECCOMP_RET_USER_NOTIF`)

- general agreement:
  - use seccomp user_notif API for fd injection, not a new syscall (e.g. `pidfd_addfd()`)
  - landed, with several bugs found/fixed around fd injection (`SCM_RIGHTS`)
- discussion shifted to user_notif in general:
  - how the user_notif APIs might want to change to be more flexible (which ultimately intersected the “changing structure sizes” topic)
    - what fields are actually needed?
    - use explicitly requested statx()-style fields?
  - whether to add `read()` API (no: stay with `ioctl()` to avoid various problems)

syscall bitmaps

- general consensus: “Yes please”
- some tangential implementation ideas came up:
  - load only from read-only memory (in support of deep argument inspection)
  - jump into middle of BPF filter based on syscall nr (may not be workable due to “no changes to classic BPF filters” mandate)
- initial “constant action bitmaps” RFC published
  - significant speed-up for allow/reject-only syscalls: O(n) into O(1)
  - faster than BPF call optimization
  - argument memory read detection should be replaced with cBPF-subset emulator, which actually looks quite feasible.

deep argument inspection (goals)

- agreed: dealing with pathnames is out of scope
  - they are not stable handles, doing the plumbing here looks really problematic, and the LSM is better suited for these kinds of “kernel object” interception.
- requirement: ToCToU-safe, so a kernel copy is needed *before the syscall proper starts*:
  - doing a universal syscall argument cache means having a way to describe all syscall argument dereference methods, and *some are technically unbounded* (linked list of userspace structures).
  - if pathnames are out of scope, and a universal caching method is infeasible, then the cache needs to be syscall-specific.
- needed right now to deal with Extensible Arguments, which are effectively zero-padded/checked append-only versioned structure, described by a size
  - e.g. `syscall(..., struct something *`**`instance`**`, size_t `**`size`**`)`, like `openat2()`, `clone3()`, etc.
- conclusion: solve the problem for these kinds of syscalls in particular, and not universally

deep argument inspection (but how)

- But how to actually define the filter access (i.e. loads from `struct seccomp_data`) to the cached extensible argument copy?
  - cBPF is frozen, use eBPF?
    - Make new eBPF helper functions to access the extensible argument contents.
    - Requires pretty major overhaul of seccomp userspace libraries and tools. Is this the moment -- is this a big enough reason to make the jump?
- And how to deal with nested growing structures?
  - look at convert_ctx_accesses() to rewrite filters based on structure layouts? dynamic filter rewriting can be fragile...
  - build jump tables and do size/offset checks in the filter? Seems like pushing way too much detail into userspace...

deep argument inspection

ETOOCOMPLEX

“So I am not in the least interested in some kind of general discussion about system calls with "pointers to pointers". They exist. Deal with it. It's not in the least an interesting issue, and no, we shouldn't make seccomp and friends incredibly more complicated for it.”

-- Linus Torvalds

(not so) deep argument inspection

- So where do we stand?
  - single-level “deep” argument inspection
  - on a syscall-by-syscall basis
  - with demonstrated need to filter those contents
- Need an RFC written to:
  - add cached extensible argument region
  - add introspection of which syscalls provide deep argument inspection
  - refactor `clone3()` to use extensible argument caching
  - expand the size of `struct seccomp_data` to hold the cached copy

changing structure sizes

- extra stuff is desired in `struct seccomp_data`:
  - high 32bits of syscall nr
  - Protection Keys (or some arch-specific area?)
  - the cached extensible argument
- changes to `seccomp_data` means changes to `USER_NOTIF` structures
  - but we’re ready for this with the work done for addfd `ioctl()`
  - bump the `USER_NOTIF` API version whenever we need to

changing `seccomp_data` size (RFC)

- treat `struct seccomp_data` like a "regular" extensible argument
  - add a size as next field
  - add flags for new fields
    - this could even be populated based on filter options to avoid copying needless stuff into `seccomp_data`
  - reserve a fixed-size hole for the contents
  - put extensible argument after the hole (if any)

add extensible argument (RFC)

- how to deal with kernel/userspace size mismatches?
  - if kernel size < user size:
    - zero-fill remaining bytes in the rest of the page?
    - rewrite filter to store zeros for “out of bounds” reads?
  - if kernel size > user size:
    - argument contents beyond user size must be zero, but this requires the kernel know what size the filter expected to be the maximum
      - punt to userspace via the filter to do the check? Seems unfriendly: greater filter complexity
      - have filters declare their expected extensible argument size at filter attach time? We’ll already need changes to introspect which syscalls have deep argument inspection available...

kitchen sink API (RFC)

```
struct seccomp_data {
          int nr;
          __u32 arch;
          __u64 instruction_pointer;
          __u64 args[6];
          __u64 size;
          __u64 features;
          __u32 nr_high;
          __u32 arch_reserved[5];
          __u8  hole[1024];
          __u64 extensible_arg_size;
          __u8  extensible_arg[];
};


#define SECCOMP_APPEND_FILTER            4
struct seccomp_append_filter {
          __u64 flags;
          struct seccomp_filter *filter;
          __u64 extensible_arg_size;
};
#define SECCOMP_APPEND_FLAG_WANT_NR_HIGH (1UL << 5)
#define SECCOMP_APPEND_FLAG_WANT_ARCH    (1UL << 6)
#define SECCOMP_APPEND_FLAG_USE_EA_SIZE  (1UL << 7)


struct seccomp_append_filter append_filter = { ... };
ret = seccomp(SECCOMP_APPEND_FILTER,
              sizeof(append_filter), &append_filter);
```

- filters reading past original structure elements will fail their attach with `EINVAL`
- should userspace do size/features checks in the filter, or explicitly request the fields be made available on a per-syscall basis? (e.g. “v2 (v3?) seccomp” `SECCOMP_APPEND_FILTER`)

- discover supported EA syscalls with another new `seccomp()` op code (e.g. `SECCOMP_GET_SYSCALL_EA_SIZE`)
- declare the EA size on per-filter basis

64-bit filter registers?

- It would be so much nicer to use 64-bit registers
  - extend seccomp-BPF dialect at verification time?
    - it is already converted to eBPF on the fly...
    - will Alexei light us on fire?

intercepting `io_uring()` “syscalls”

- very different hierarchy than seccomp (i.e. process tree)
  - io_uring crosses process hierarchies (via fd passing)
    - attached to struct cred, not struct task_struct
    - multiple io_uring instances can be present in a single process
  - io_uring even crosses mm boundaries
- no io_uring()-specific syscall needed to even *use* an io_uring!
  - receive io_uring fd, mmap fd, use io_uring...
  - somewhat mitigated by such an io_uring fd needing creation-time features that require `CAP_SYS_ADMIN`
- containing io_uring seems best suited to LSMs ...
- seccomp can block `recvmsg()` or `socket(AF_UNIX, ...)`, but not just `SCM_RIGHTS` ...

Thank you; stay safe!

Thoughts? Questions?

Kees (“Case”) Cook

keescook@chromium.org
keescook@google.com
kees@outflux.net

@kees_cook