Deep Argument Inspection

Linux Plumbers Conference 2019

Kees Cook <keescook@chromium.org>
Christian Brauner <christian.brauner@ubuntu.com>

https://outflux.net/slides/2019/lpc/deep-arg-inspection.pdf

use cases

- programmatic filesystem isolation
  - to not depend on containers, mount namespaces, etc
- new syscall APIs require structure content analysis
  - clone3, openat2
- accurate passive monitoring (“what was requested?”)
  - what path was opened?
  - where is the packet going?
- meaningful active monitoring (change requests)
  - did the container just ask to install a kernel module that I need to do for it? or a mount?

background: syscall flow

- kernel entry
- ptrace entry events (blocking!)
  - may change anything, including syscall number
  - may skip syscall
- seccomp hooks
  - may kill thread or entire process
  - may skip syscall (silently or with errno)
  - may log
  - may send signal (which can be caught and continued from)
  - may notify userspace (blocking!)
  - may do ptrace event (blocking!)
    - may change anything, including syscall number...
    - re-run the seccomp filter now: goto “seccomp hooks” but stop any further recursion
- actual syscall function
  - copy userspace memory for parsing
  - parse userspace args into kernel objects
  - LSM hooks (accept/reject only!)
  - perform actions on kernel objects
- ptrace exit events (blocking!)
- return to userspace

ptrace is in the wrong place for deterministic analysis and manipulation of syscalls

- userspace memory is mutable by other threads until the arguments are copied from userspace in the body of the syscall function
- seccomp is in the wrong place too...

LSM is not syscall filtering

- LSM hook doesn’t know if called from access() or open()
  - though there is an exception now for uid/gid changes SafeSetID LSM
- in the right place for deep inspection: it mediates kernel objects
- syscall filtering really wants to be mediating kernel objects too
- there is no unprivileged LSM

should deep inspection happen via seccomp at all?

- there is a strong argument to made for doing deep inspection at the LSM level. Otherwise, there is a risk of letting userspace get sloppy: only filter open() for a path but not rename()

create an association between seccomp and LSM?

- remember that this syscall should be deeply inspected
- add seccomp LSM hook to perform seccomp inspection if indicated
  - LSM hooks are only accept/reject. Is that “sufficient” for seccomp? Unlikely: some seccomp features expect to perform manipulations before entering a (possibly new) syscall. But let’s explore what COULD be done...
- add new LSM return value for “hand back to seccomp”?
- add seccomp hook at syscall *exit*
- could handle kill, errno, fake success
- cannot do user notification (if we want to expand it to “continue” as proposed recently)
- cannot do ptrace or signal: would need to restart syscall
- jump back to seccomp entry logic? No! just introduces the memory copying race all over
- seems like an overly complex direction and a strong indication of extreme layering violations

move seccomp?

- ABI says we must run after ptrace, so moving seccomp deeper into the syscall entry stack would be okay...
- adding a hook to the body of every syscall function feels completely wrong: more laying violations, cut/pasting bugs, etc

cache userspace memory copies?

- seccomp can examine them before syscall function entry ... but that means duplicating the *parsing* logic in seccomp that the syscall function is also going to do
- which could also be a new kind of race: kernel object may change between seccomp filter and syscall function (e.g. file rename)

move argument parsing?

- what if syscalls declare their argument types more completely to have memory copied to kernel and parsed into kernel objects before syscall function entry?
- what would this look like?
  - some things are “just” structures that the syscall will act from (e.g. new mount API)
  - some things need to be resolved into kernel objects (e.g. file handles from a path string)
- some syscalls do crazy things and walk lists of structures in userspace...
- would this be a performance issue (reading everything before you know if it will be needed)?
- is moving argument parsing an information leak?
  - e.g. timing to resolve arguments between entry and seccomp filter run may leak presence of files? (I feel like this can already be done)
- but at least it could likely be done piecemeal across syscalls until everything was converted
- would give us a much more coherent set of metadata about syscall arguments that could be used by fuzzers, etc

ideas?

Kees Cook <keescook@chromium.org>

https://outflux.net/slides/2019/lpc/deep-arg-inspection.pdf