Kernel Self Protection

Kernel Summit 2016, Santa Fe

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

http://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project
http://www.openwall.com/lists/kernel-hardening/

https://outflux.net/slides/2016/ks/kspp.pdf

Agenda

- Goal review
- GCC plugins
- Probabilistic protections
- Deterministic protections

Goal review

- When I talk about kernel security, I mean more than access control and fixing bugs
- This is “kernel self-protection”
  - Eliminate security bug classes
    - lifetimes are long (5 years average), bugs are always present
  - Eliminate exploitation methods
    - reduce attack surface, create hostile environment for attacks
- Choose where to focus based on real-world exploits and low-hanging fruit

GCC plugins

- CONFIG_GCC_PLUGINS
  - tested on x86, arm, and arm64
  - let's add more!
- Want to drop “depends on !COMPILE_TEST”
  - Needs gcc 4.5 or newer
  - Many many build systems need to add the gcc plugin headers...
    - Debian/Ubuntu: gcc-$N-plugin-dev(-$arch-linux-$abi)
    - Fedora: gcc-plugin-devel (not sure about cross compiler)
  - When is the “best” time to land this kind of change?

Probabilistic protections

- Protections that derive their strength from some system state being unknown to an attacker
- Weaker than “deterministic” protections since information exposures can defeat them, though they still have real-world value
- Familiar examples:
  - stack protector (cookie value can be exposed)
  - Address Space Layout Randomization (offset can be exposed)

Probabilistic:
KASLR text base

- Randomly relocates start of kernel image at boot: CONFIG_RANDOMIZE_BASE
- x86 (v3.14), arm64 (v4.6), MIPS (v4.7)
- Lots of local exposures weaken KASLR
- Still valuable defense-in-depth
- Needs to be more than just randomized base offset
- Maybe randomize link order at boot?

Probabilistic:
KASLR memory base

- Even with text base KASLR, memory allocations for a given system may be deterministic at boot
- Randomize page table, vmap, etc areas: CONFIG_RANDOMIZE_MEMORY
- x86_64 (v4.8), arm64 (v4.6?)

Probabilistic: freelist randomization

- Makes heap spraying attacks less deterministic: CONFIG_SLAB_FREELIST_RANDOM
- SLAB (v4.7), SLUB (v4.8)

Probabilistic: struct randomization

- The most heavily targeted things in the kernel are structures containing function pointers
- “RANDSTRUCT” GCC plugin from grsecurity
- Automatically randomize structure layout for these structures and manually marked ones
- Can limit randomization within cachelines

Deterministic protections

- Protections that derive their strength from organizational system state that always blocks attackers
- Familiar examples:
  - Read-only memory (writes will fail)
  - Bounds-checking (large accesses fail)

Deterministic: Kernel memory protection

- Fundamental memory integrity protection
- Poorly named: CONFIG_DEBUG_RODATA
  - Not just a debug feature
  - Besides making .rodata read-only, ensures memory is either executable nor writable, never both
- Mandatory:
  - x86 (v4.6), arm64 (v4.9)
  - Almost: arm (v4.6 on-by-default, has corner cases)

Deterministic: Privileged userspace access blocking

- The most common attack method is to redirect execution or data dereferences into userspace memory
- Block kernel from direct userspace execution or read/write by segregating userspace/kernel memory
- In hardware (years away from real-world penetration):
  - x86 (SMEP and SMAP) since Skylake … no Xeons!
  - arm64 (PXN and PAN) since ARMv8.1 … any manufactured?
- Emulation is fundamentally important:
  - arm (v4.3): CONFIG_CPU_SW_DOMAIN_PAN
  - arm64 (v4.10...): CONFIG_ARM64_SW_TTBR0_PAN
  - x86 needed! (Implemented in PaX with PCIDs)

Deterministic: vmap stack & thread_info removal

- Common attack is intentional stack exhaustion to overwrite parts of thread_info, or reach into neighboring stacks
- vmap stack gains the vmap guard page: CONFIG_VMAP_STACK
- thread_info removal relocates attack targets to harder-to-find memory: CONFIG_THREAD_INFO_IN_TASK
- x86 (v4.9), arm64 (v4.10...), s390 (v4.10?)

Deterministic: usercopy sanity checking

- Common bug is broken bounds checking on copy_to/from_user()
- Best-effort during compile time via builtin_const checks (has existed in various forms, but most complete since v4.8)
- Runtime checks when not builtin_const (v4.8): CONFIG_HARDENED_USERCOPY
- Need to whitelist slab caches with “can be shared with userspace” flag, then create “exception” API with builtin_const bounds to bypass whitelist for known-good things like in-inode filenames, etc.

Deterministic: memory wiping

- Stops many forms of information leaks, blocks a few use-after-free situations
- Page allocator can do zero-poisoning (v4.6)
- Slab allocator has poisoning but not zeroing
- Join us Wed in the mm break-out discussion
  - Slab poisoning cache blacklisting for better performance
  - Too many CONFIGs and cmdline arguments to enable:
    - CONFIG_DEBUG_PAGEALLOC=n, CONFIG_PAGE_POISONING=y, CONFIG_PAGE_POISONING_NO_SANITY=y, CONFIG_PAGE_POISONING_ZERO=y, CONFIG_SLUB_DEBUG=y
    - page_poison=on slub_debug=P

Deterministic: constification

- Attack surface reduction: extend what is read-only in the kernel
- Like “RANDSTRUCT”, the “CONSTIFY” GCC plugin from grsecurity targets function pointer tables and manually marked variables
- Classes of data in the kernel:
  - read/write
  - read-only
  - read-only-after-init (v4.6, needs more users)
  - read-mostly (a performance distinction...)
  - write-rarely (need to make this NOT read-only precisely during updates)

Deterministic: atomic wrap detection

- Recurring source of use-after-free flaws is wrapping atomic_t (and family) which are very commonly used as reference counters
- No measurable performance impact and an entire class of bugs goes away: CONFIG_HARDENED_ATOMIC
  - Add atomic_wrap_t for *statistical* counters, or other things that don't care
  - Switch expected-to-wrap variables to atomic_wrap_t
  - Trap overflow/underflow of atomic_t
- x86, arm, arm64 almost ready for review
- Other architectures can be similarly extracted from grsecurity

Questions / Comments / Flames

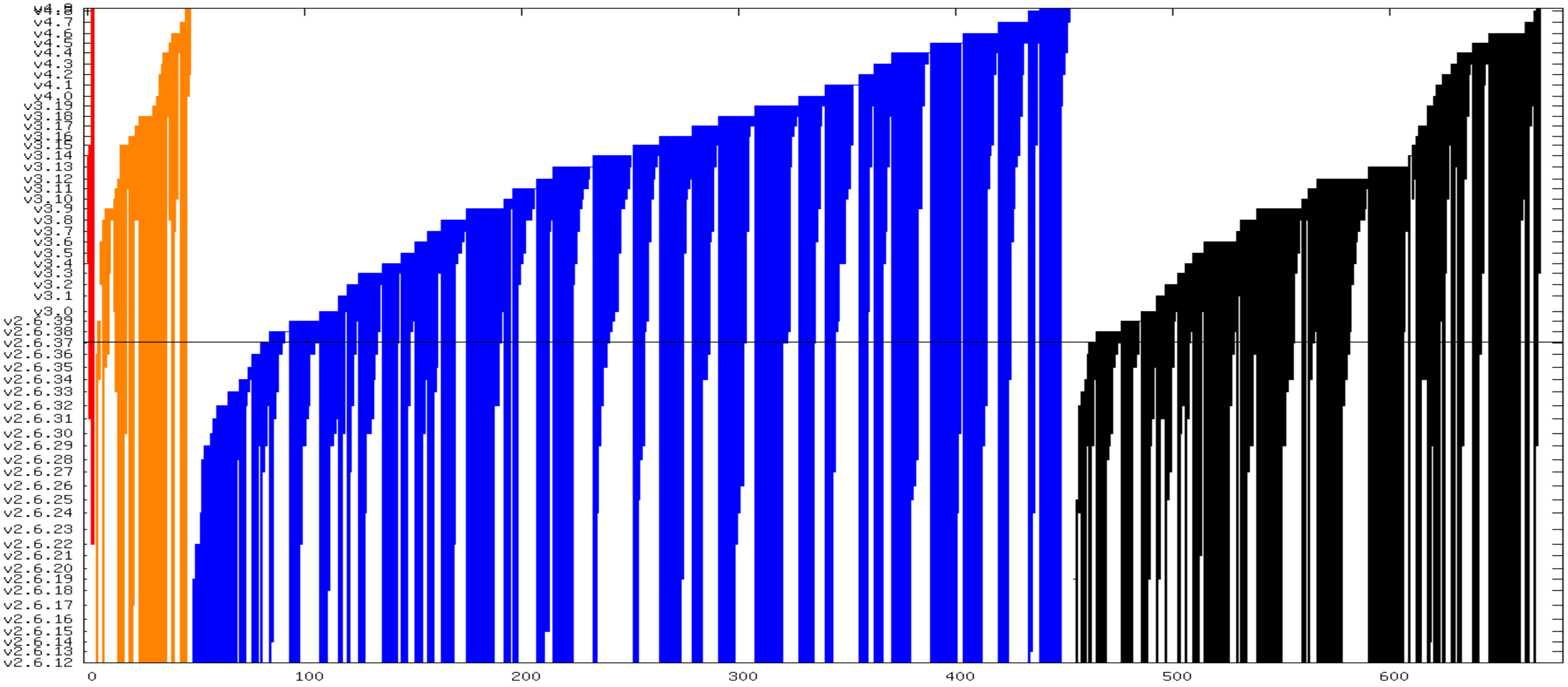
Kees (“Case”) Cook
keescook@chromium.org
keescook@google.com
kees@outflux.net
@kees_cook

http://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project
http://www.openwall.com/lists/kernel-hardening/

https://outflux.net/slides/2016/ks/kspp.pdf

bonus slides ...

CVE lifetimes



critical & high CVE lifetimes