Overview and Recent Developments: Kernel Self-Protection Project

Linux Security Summit EU
October 25, 2018
Edinburgh, Scotland

Kees ("Case") Cook
keescook@chromium.org
@kees_cook

https://outflux.net/slides/2018/lss-eu/kspp.pdf

Kernel Security for *this* talk is ...

- More than access control (e.g. SELinux)
- More than attack surface reduction (e.g. seccomp)
- More than bug fixing (e.g. CVEs)
- More than protecting userspace
- More than kernel integrity
- This is about *Kernel Self Protection*

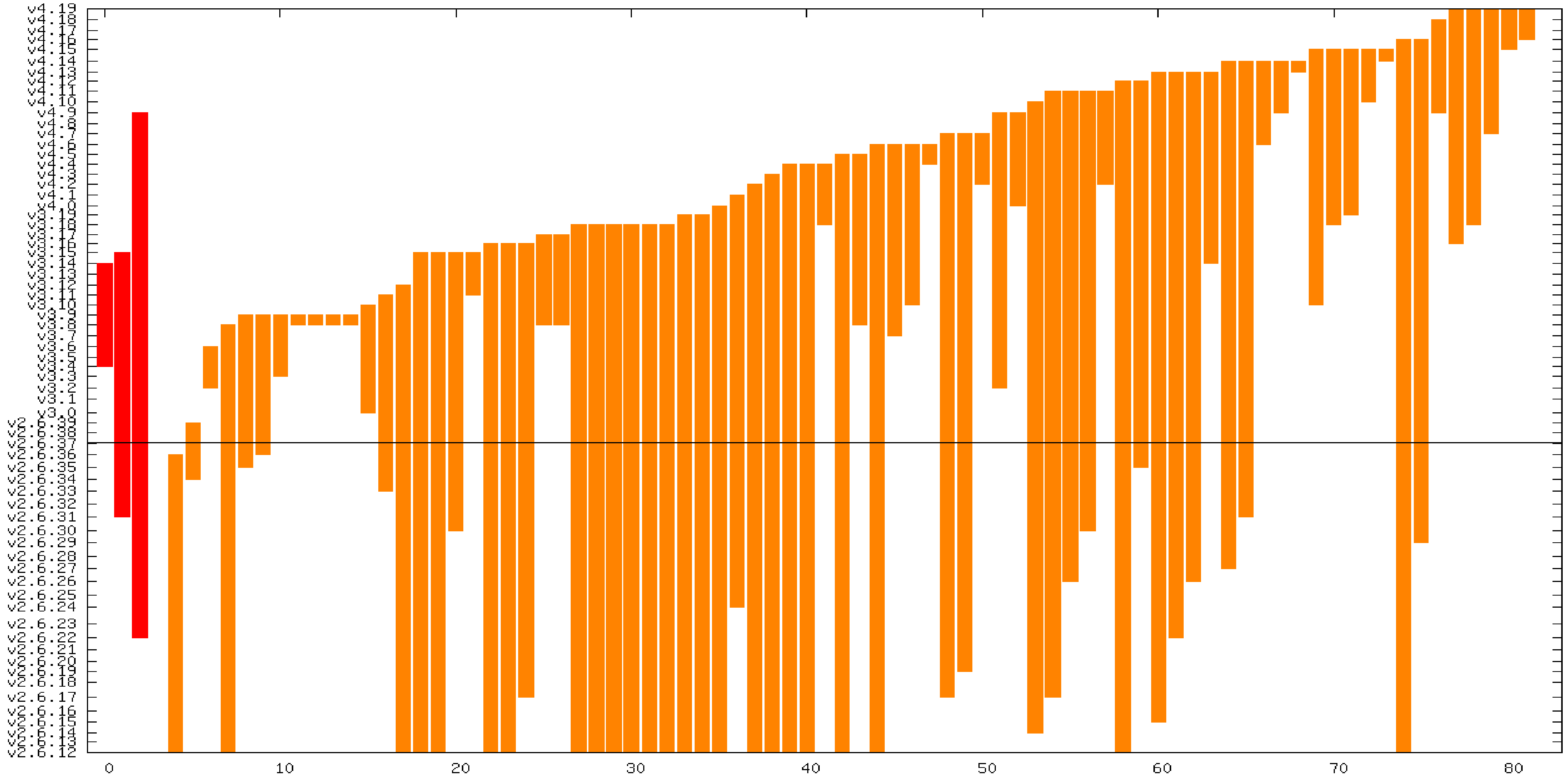
What needs securing?

- Servers, laptops, cars, phones, TVs, space stations, …
- >2,000,000,000 active Android devices in 2017
  - Majority are running v3.10 (with v3.18 slowly catching up)
- Bug lifetimes are even longer than upstream
- “Not our problem”? Even if upstream fixes every bug found, and the fixes are magically sent to devices, bug lifetimes are still huge.

Upstream Bug Lifetime

- In 2010 Jon Corbet researched security flaw fixes with CVEs, and found that the average time between introduction and fix was about 5 years.
- My analysis of Ubuntu CVE tracker for the kernel from 2011 through 2018 crept closer to 6 years for a while, but has now started to diminish:
  - Critical: 3 at 5.3 years average
  - High: 79 at 5.6 years average
  - Medium: 691 at 5.9 years average
  - Low: 349 at 6.2 years average

critical & high CVE lifetimes



Attackers are watching

- The risk is not theoretical. Attackers are watching commits, and they are better at finding bugs than we are:
  - http://seclists.org/fulldisclosure/2010/Sep/268
- Most attackers are not publicly boasting about when they found their 0-day...

Bug fighting continues

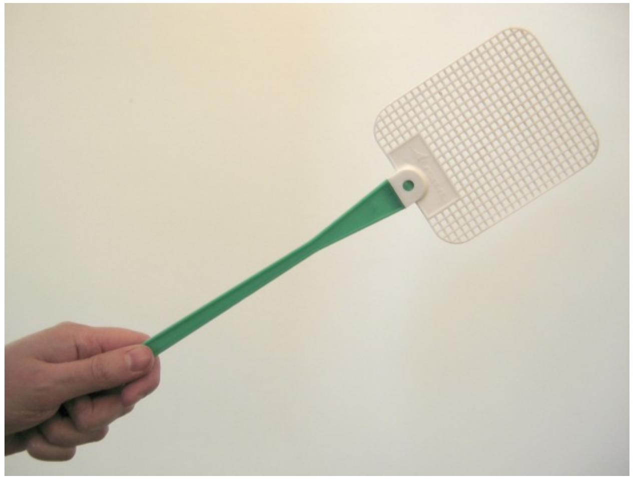
- We’re finding them
  - Static checkers: gcc, Clang, Coccinelle, Smatch, sparse, Coverity
  - Dynamic checkers: kernel, KASan-family, syzkaller, trinity
- We’re fixing them
  - Ask Greg KH how many patches land in -stable
- They’ll always be around
  - We keep writing them
  - They exist whether we’re aware of them or not
  - Whack-a-mole is not a solution

Analogy: 1960s Car Industry

- Konstantin Ryabitsev's keynote at 2015 Linux Security Summit
  - http://kernsec.org/files/lss2015/giant-bags-of-mostly-water.pdf
- Cars were designed to run, not to fail
- Linux now where the car industry was in 1960s
  - https://www.youtube.com/watch?v=fPF4fBGNK0U
- We must handle failures (attacks) safely
  - Userspace is becoming difficult to attack
  - Containers paint a target on the kernel
  - Lives depend on Linux

Killing bugs is nice

- Some truth to security bugs being “just normal bugs”
- Your security bug may not be my security bug
- We have little idea which bugs most attackers use
- Bug might be in out-of-tree code
  - Un-upstreamed vendor drivers
  - Not an excuse to claim “not our problem”



Killing bug classes is better

- If we can stop an entire kind of bug from happening, we absolutely should do so!
- Those bugs never happen again
- Not even out-of-tree code can hit them
- But we’ll never kill all bug classes

Killing exploitation is best

- We will always have bugs
- We must stop their exploitation
- Eliminate exploitation targets and methods
- Eliminate information exposures
- Eliminate anything that assists attackers
- *Even if it makes development more difficult*

Kernel Self-Protection Project

- KSPP focuses on the kernel protecting the *kernel* from attack (e.g. refcount overflow) rather than the kernel protecting *userspace* from attack (e.g. namespaces) but both and all other areas of related development are welcome
- ~12 organizations and ~10 individuals working on ~20 technologies

I used to say:

Slow and steady

but Alexander Popov suggested a better motto:

Flexible and Persistent

A year's worth of kernel releases ...

v4.14

- 3 `refcount_t` conversions (bikeshedding stall)
- randstruct plugin (automatic mode)
- SLUB freelist pointer obfuscation
- structleak plugin (by-reference mode)
- `VMAP_STACK`, arm64
- `set_fs()` removal progress
- `set_fs()` balance detection, x86, arm64, arm

v4.15

- 35 `refcount_t` conversions (32 remaining...)
- PTI, x86
- retpoline
- `struct timer_list .data` field removal
- fast refcount overflow protection, x86 (also in v4.14 -stable)
- `%p` hashing

v4.16

- 12 `refcount_t` conversions (20 more?)
- PTI, arm64
- hardened usercopy whitelisting
- `CONFIG_CC_STACKPROTECTOR_AUTO`

v4.17

- 51 VLAs removed (83 remaining...)
- Clear stack on fork
- More fixes to stack RLIMIT on exec
- `MAP_FIXED_NOREPLACE`
- Unused register clearing on syscall entry, x86
- Speculative Store Bypass Disable, x86

v4.18

- 38 VLAs removed (45 remaining...)
- Arithmetic overflow detection helpers
- Allocation overflow detection refactoring
- Speculative Store Bypass Disable, arm64

v4.19

- 33 VLAs removed (12 remaining: most in crypto API)
- Shift overflow helpers
- L1TF defenses
- Restrict `O_CREAT` for existing files and pipes in `/tmp`
- Unused register clearing on syscall entry, arm64

Expected for v4.20

- VLAs removed completely, `-Wvla` added
- stackleak gcc plugin (x86 and arm64)

Various soon and not-so-soon features

- always initialized variables
- Link-Time Optimization
- switch fallthrough marking
- memory tagging
- eXclusive Page Frame Owner
- SMAP emulation, x86
- brute force detection
- write-rarely memory
- KASLR, arm
- integer overflow detection
- Control Flow Integrity
- per-task stack canary, non-x86
- {str,mem}cpy alloc size checks
- fine-grained KASLR
- per-CPU page tables
- read-only page tables
- hardened slab allocator
- hypervisor magic :)

Challenges

Cultural: Conservatism, Responsibility, Sacrifice, Patience

Technical: Complexity, Innovation, Collaboration

Resources: Dedicated Developers, Reviewers, Testers, Backporters

Thoughts?

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

https://outflux.net/slides/2018/lss-eu/kspp.pdf

https://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project
http://www.openwall.com/lists/kernel-hardening/
`##linux-hardened` on Freenode