Kernel Self-Protection Project

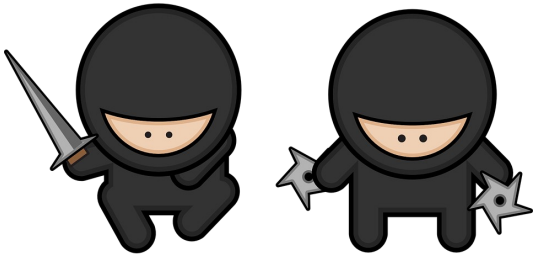
Linux Security Summit NA
August 21, 2019
San Diego, California

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

https://outflux.net/slides/2019/lss/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*



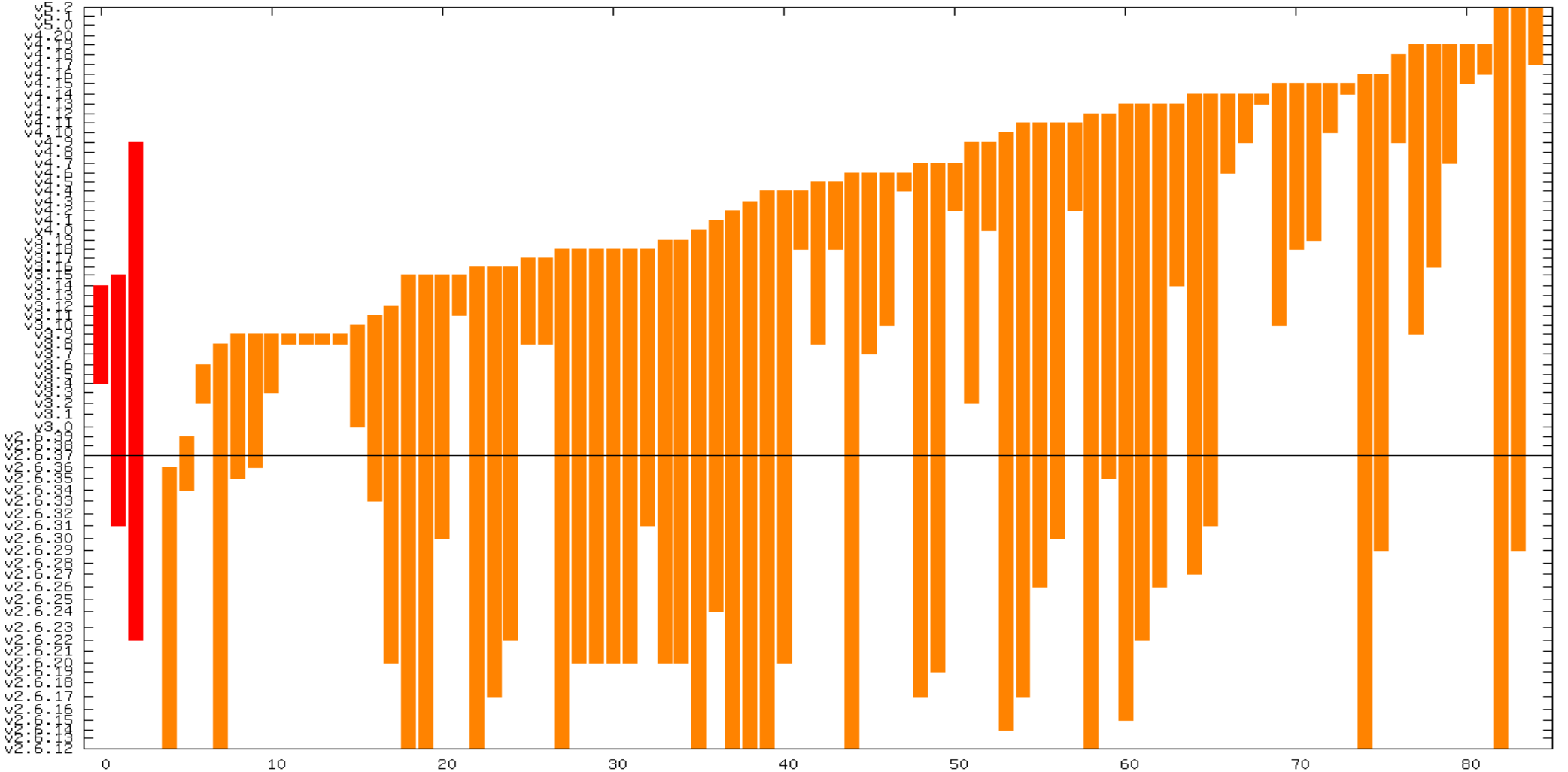
What needs securing?

- Servers, laptops, cars, phones, TVs, space stations, …
- >2,500,000,000 active Android devices in 2019
  - Majority are running v3.18 (with v4.4 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 the Ubuntu CVE tracker for the kernel from 2011 through 2019 crept closer to 6 years for a while, but has now started to diminish:
  - Critical: 3 at 5.3 years average
  - High: 81 at 5.4 years average
  - Medium: 749 at 6.0 years average
  - Low: 368 at 6.5 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

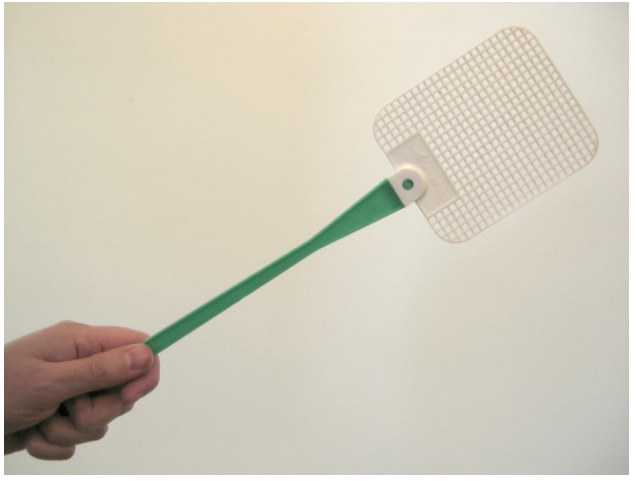
- We’re finding them
  - Static checkers: gcc, Clang, Coccinelle, Smatch, sparse, Coverity
  - Dynamic checkers: kernel, KASan-family, syzkaller, trinity
- We’re fixing them
  - Ask Greg KH how many patches land in -stable
- They’ll always be around
  - We keep writing them
  - They exist whether we’re 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*


REMOVE
BUGS
REMOVE
BUG
CLASSES
REMOVE
EXPLOIT
METHODS
REMOVE
C

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 ...

v4.19

- 33 VLAs removed (12 remaining: most in crypto API)
- 3 `refcount_t` conversions (3 bugs found via `refcount_t`)
- 129 implicit fallthroughs marked (3 missing breaks found)
- Shift overflow helpers
- L1TF mitigations
- Restrict `O_CREAT` for existing files and pipes in `/tmp`
- Unused register clearing on syscall entry, arm64

v4.20

- All VLAs removed! Building with `-Wvla` by default
- 7 `refcount_t` conversions (2 bugs found via `refcount_t`)
- 59 implicit fallthroughs marked (2 missing breaks found)
- stackleak plugin
- per-task stack canary, powerpc
- jump labels read-only after init
- STIBP mitigations
- raise `copy_{to,from}_user()` kernel address faults

v5.0

- 2 `refcount_t` conversions (5 bugs found via `refcount_t`)
- 56 implicit fallthroughs marked (3 missing breaks found)
- read-only linear mapping, arm64
- per-task canary, arm & arm64
- kernel top byte ignore, arm64
- userspace PAC, arm64
- kernel-only platform keyring

v5.1

- 13 `refcount_t` conversions (6 bugs found via `refcount_t`)
- 100 implicit fallthroughs marked (10 missing breaks found)
- `pidfd` from `/proc`, `pidfd_send_signal()` for ... sending signals
- heap mapping validations (2 bugs immediately found)
- LSM stacking, shared security blobs
- SafeSetID LSM
- stack variable auto-init GCC plugin now covers scalars

v5.2

- 1 `refcount_t` conversion
- 71 implicit fallthroughs marked (6 missing breaks found)
- `pidfd` from `clone()` via `CLONE_PIDFD`
- page allocator freelist randomization
- stack variable auto-initialization with Clang
- KUAP on powerpc (like SMAP on x86)
- MDS mitigations
- `userfaultfd` sysctl knob
- temporary mm for kernel text poking

Expected for v5.3

- Building with `-Wimplicit-fallthrough` by default! (last 69 marked and 7 missing breaks found)
- 2 `refcount_t` conversions (1 bug found via `refcount_t`)
- `pidfd` from `pidfd_open()`
- cr4, cr0 pinning on x86
- heap auto initialization
- additional `kfree()` sanity checking

Planned for v5.4

- `pidfd` with `waitid()` via `P_PIDFD`
- kernel lockdown LSM
- `stracpy()` for `char` arrays
- `strscpy()` `INT_MAX` test

Various soon and not-so-soon features

- O_BENEATH and friends
- Link-Time Optimization
- memory tagging
- eXclusive Page Frame Owner
- SMAP emulation, x86
- brute force detection
- write-rarely memory
- KASLR, arm
- integer overflow detection
- Control Flow Integrity
- {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

@kees_cook

https://outflux.net/slides/2019/lss/kspp.pdf

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