LINUX SECURITY SUMMIT

Kernel Self-Protection Project

September 29, 2021

Kees ("Case") Cook

keescook@chromium.org

@kees_cook

https://outflux.net/slides/2021/lss/kspp.pdf

THE LINUX FOUNDATION

“Kernel Security” for *this* talk is ...

- not just access control (e.g. SELinux)
- not just attack surface reduction (e.g. seccomp)
- not just bug fixing (e.g. CVEs)
- not just protecting userspace
- not just memory integrity

This is about *Kernel Self Protection*

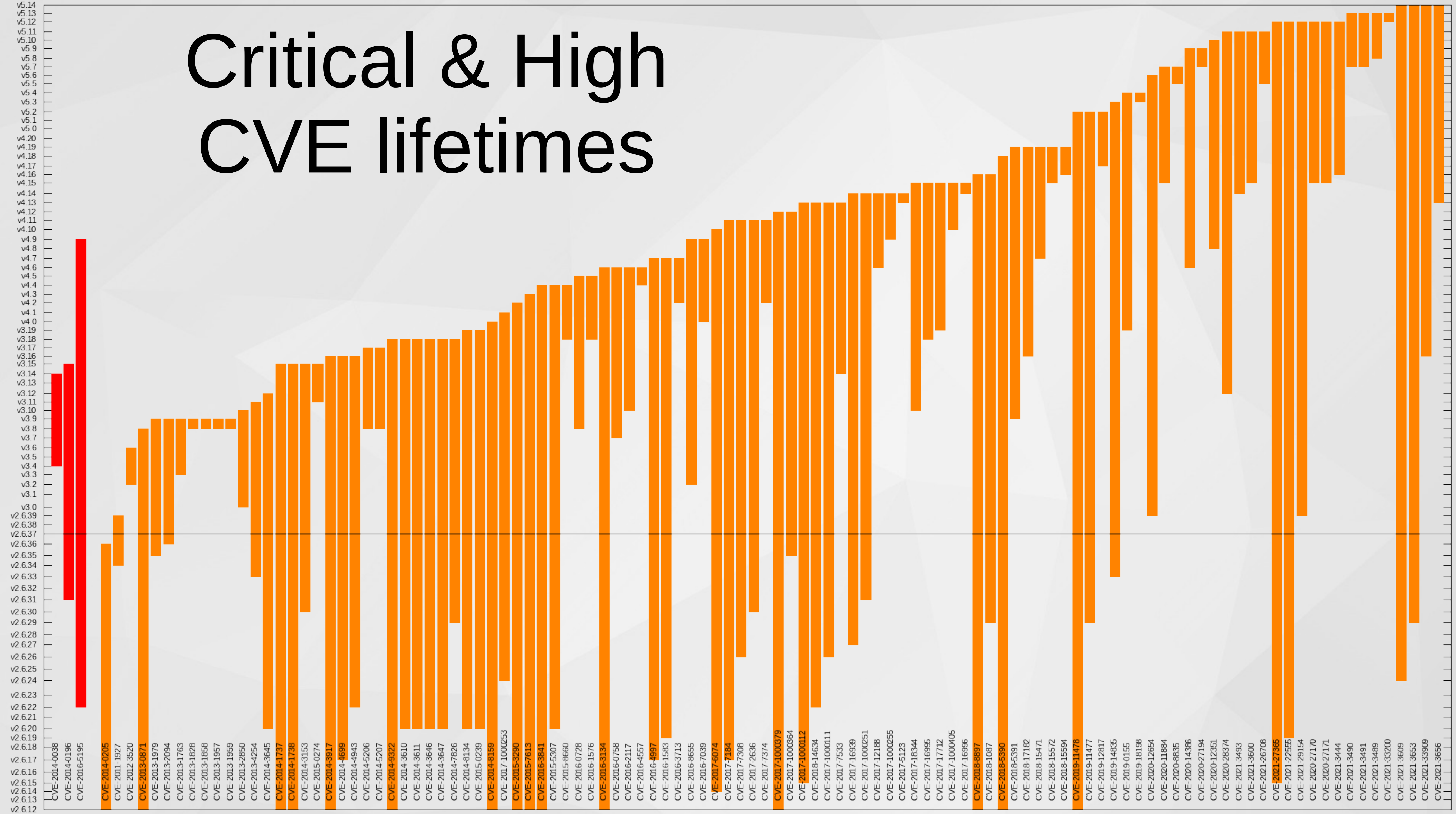
What needs protecting?

- “Downstream”: servers, laptops, phones, TVs, vehicles, space stations, Martian helicopters …
- >3 ***billion*** active Android devices in 2021
  - Majority run v4.14 (released Nov 2017) with v4.19 (Oct 2018) slowly catching up
- Most downstream bug lifetimes are even longer than upstream
  - Everyone needs to run the latest kernels and test as close to `linux-next` as possible
    https://security.googleblog.com/2021/08/linux-kernel-security-done-right.html
- Upstream devs can reasonably believe downstream bug fixing is “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: 5 ½ years

- In 2010 Jon Corbet researched security flaw fixes with assigned CVEs, and found that the average time between introduction and fix was about 5 years.
- Starting in 2015, I began a similar analysis of the Ubuntu kernel CVE tracker. This showed "critical" and "high" priority flaw lifetime was closer to 6 years for a while, then went down, and has stabilized again at 5.5:
  - Critical: 3 at 5.3 years average
    - no critical flaws since 2016's DirtyCOW (waiting for other shoe to drop)
  - High: 108 at 5.5 years average (+20   thanks Pandemic)
    - 2020: 88 @ 5.4 (+7)
    - 2019: 81 @ 5.4 (+10)
    - 2018: 71 @ 5.9 (+12)
    - 2017: 59 @ 6.4 (+25)
    - 2016: 34 @ 6.4 (+3)
    - 2015: 31 @ 6.3
  - Medium: 1038, Low: 526
    - These don't tend to get sufficiently accurate "originating commit" analysis.

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

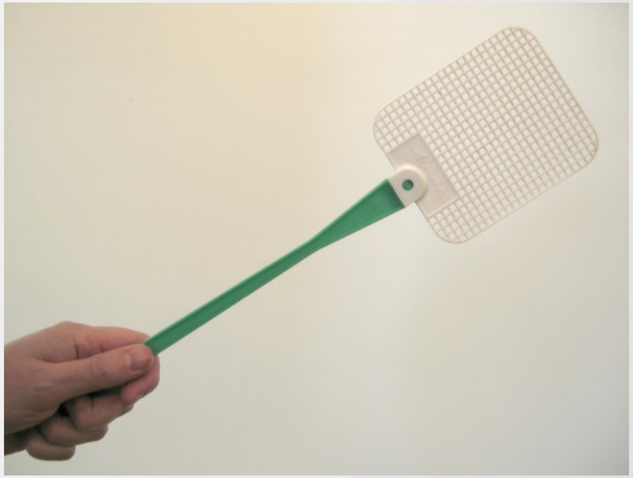
- We’re finding them
  - Static checkers: gcc, Clang, Coccinelle, Smatch, sparse, Coverity
  - Dynamic checkers: kernel, KASan-family, syzkaller, stress-ng, trinity
- We’re fixing them
  - Ask Greg KH how many patches land in -stable
- They’ll always be around
  - We keep writing them
  - They exist 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
- There isn’t a common theme to the bugs attackers use beyond “whatever they can find”
- Bug might be in out-of-tree code
  - Un-upstreamed vendor drivers
  - Not an excuse to claim “not our problem”



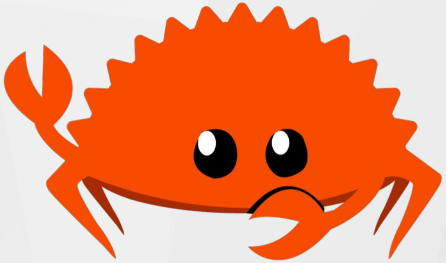
Killing bug classes is better

- If we can stop an entire kind of bug from happening, we absolutely should do so!
  - General robustness improvements beyond security
- Those bugs never happen again (not even in out-of-tree code)
- 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 has already effectively forked the C language
- Keep removing dangerous things from C ...
- ... add Rust for new stuff



https://security.googleblog.com/2021/04/rust-in-linux-kernel.html

Kernel Self-Protection Project

- KSPP focuses on the upstream Linux kernel protecting the *kernel* from attack (e.g. array bounds checking) rather than the kernel protecting *userspace* from attack (e.g. namespaces), but both (and all other) areas of related development are welcome.

  Mailing list archive: https://lore.kernel.org/linux-hardening/

  Issue tracker: https://github.com/KSPP/linux/issues

I used to say:

← *Slow and Steady*

but Alexander Popov suggested a better motto:

Flexible and Persistent →

Two years worth of kernel releases …

v5.3 (Sep 2019)

- building with `-Wimplicit-fallthrough` by default for GCC! (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 variable auto initialization via `init_on_{alloc,free}=1` boot parameter
- additional `kfree()` sanity checking
- KASLR enabled by default on arm64
- hardware security embargo documentation

v5.4 (Nov 2019)

- `pidfd` with `waitid()` via `P_PIDFD`
- kernel lockdown LSM
- tagged memory relaxed syscall ABI
- boot entropy improvement
- userspace writes to swap files blocked
- limit `strscpy()` sizes to `INT_MAX`
- `ld.gold` support removed
- Intel TSX disabled
- ongoing refactoring: `refcount_t`

v5.5 (Jan 2020)

- restrict `perf_event_open()` from LSM
- generic fast full `refcount_t` (and more conversions)
- linker script cleanup for exception tables
- KASLR for powerpc32
- seccomp: riscv support, `USER_NOTIF` continuation
- `EFI_RNG_PROTOCOL` for x86
- `FORTIFY_SOURCE` for MIPS
- limit `copy_{to,from}_user()` size to `INT_MAX`
- KASan support for vmap memory
- MIPS can build with GCC plugins
- `userfaultfd` requires `CAP_SYS_PTRACE` for `UFFD_FEATURE_EVENT_FORK`

v5.6 (Mar 2020)

- WireGuard
- `openat2()` syscall and `RESOLVE_{BENEATH,NO_SYMLINKS,...}` flags
- `pidfd_getfd()` syscall
- `openat()` via `io_uring`
- removal of blocking random pool
- arm64: on-chip RNG support, E0PD support (constant-time memory faults)
- `VMAP_STACK` on powerpc32
- generic Page Table dumping
- replacing 0-length and 1-element arrays with flexible arrays refactoring begins

v5.7 (May 2020)

- arm64 kernel Pointer Authentication (PAC)
- BPF LSM
- `execve()` deadlock refactoring
- slub freelist obfuscation improvements
- riscv strict kernel memory protections
- `CONFIG_UBSAN_BOUNDS` split off for run-time array index bounds checking
- fixing "appending" `snprintf()` usage with `scnprintf()` refactoring begins
- ongoing refactoring: flexible arrays, `refcount_t`

v5.8 (Aug 2020)

- arm64: Branch Target Identification, Shadow Call Stack
- Kernel Concurrency Sanitizer infrastructure added
- new capabilities: `CAP_PERFMON`, `CAP_BPF`
- network RNG improvements
- fix various kernel address exposures to non-`CAP_SYSLOG`
- riscv W^X detection
- `execve()` refactoring continues
- multiple `/proc` instances
- `set_fs()` removal preparation continues
- `READ_IMPLIES_EXEC` removed for native 64-bit architectures
- ongoing refactoring: `scnprintf()` replacement, flexible arrays, `refcount_t`

v5.9 (Oct 2020)

- seccomp: `USER_NOTIF` file descriptor injection, more architecture support: SuperH, C-SKY, xtensa
- zero-initialize stack variables with Clang
- common syscall entry/exit routines
- SLAB `kfree()` hardening
- new `CAP_CHECKPOINT_RESTORE` capability
- `debugfs` boot-time visibility restriction
- stack protector support for riscv
- new `tasklet` API
- x86: `FSGSBASE` implementation, filter x86 `MSR` writes
- `uninitialized_var()` macro removed
- ongoing refactoring: function pointer cast removals, flexible arrays, `scnprintf()`, `refcount_t`

v5.10 (Dec 2020)

- improved `prandom()` (e.g. network) entropy
- SafeSetID LSM gained gid awareness
- LSM kernel file reading hooks
- `set_fs()` removed from x86, riscv, powerpc
- `sysfs_emit()` added as work-around for `snprintf() usage`
- `nosymfollow` mount option
- AMD SEV register encryption
- arm64 Memory Tagging Extension support
- static calls API for replacing global function pointers
- implicit-fallthrough vs Clang refactoring begins
- ongoing refactoring: flexible arrays, `scnprintf()`, `refcount_t`

v5.11 (Feb 2021)

- split `CONFIG_UBSAN_MISC` for other inexpensive run-time checks (e.g. shift overflow)
- arm32: signal page poisoning, KAsan support
- arm64: `CONFIG_KASAN_HW_TAGS`, `set_fs()` removed
- intra-object overflow in fortified string functions
- `unprivileged_userfaultfd` sysctl
- `CONFIG_PAGE_POISONING_{ZERO,NO_SANITY}` removed
- Syscall User Dispatch
- seccomp constant-time bitmaps
- replacing `strcpy()`, `strlcpy()`, and `strncpy()` with `strscpy()` refactoring begins
- ongoing refactoring: Clang implicit-fallthrough, flexible arrays, `scnprintf()`, `refcount_t`

v5.12 (Apr 2021)

- UBSAN integer overflow checks removed due to GCC 8+ breakage
- KFENCE implemented for x86 and arm64 (heap OOB, UaF)
- `kcmp()` more generally available
- more network RNG improvements
- `MOUNT_ATTR_IDMAP` & `mount_setattr()` for user-namespace aware mounts
- per-task stack canaries on riscv
- Clang Link Time Optimization (LTO) build support
- ongoing refactoring: Clang implicit-fallthrough, flexible arrays, `scnprintf()`, `strscpy()`

v5.13 (Jun 2021)

- Landlock LSM
- Clang Control Flow Integrity (CFI) for arm64
- per-syscall kernel stack offset randomization
- check `/proc/$pid/attr/` writes against file opener
- `/dev/kmem` removed
- `set_fs()` removed from MIPS
- eXecute-Only Memory (XOM) for arm64 under EPAN (ARMv8.7-A)
- FORTIFY_SOURCE enabled for riscv
- x86_32 stack protector support removed for GCC < 8.1
- ongoing refactoring: Clang implicit-fallthrough, flexible arrays, `scnprintf()`, `strscpy()`

v5.14 (Aug 2021)

- network RNG improvements (replace Jenkins with SipHash)
- `memfd_secret()` syscall to create "secret" memory areas
- `VMAP_STACK` for riscv
- seccomp: atomic "`NOTIF_ADDFD` + send reply"
- `memcpy()` overflow refactoring begins
- ongoing refactoring: Clang implicit-fallthrough, flexible arrays, `scnprintf()`, `strscpy()`, `refcount_t`

Expected for v5.15 (Oct 2021?)

- another push for replacing open-coded size arithmetic with `struct_size()` and related helpers begins
- `kvmalloc()` limited to `INT_MAX`
- UBSAN available on riscv
- `set_fs()` removed from arm32
- L1D flushing API added
- call-used register clearing (GCC 11’s `-fzero-call-used-regs=used-gpr`)
- ongoing refactoring: Clang implicit-fallthrough, flexible arrays, `scnprintf()`, `strscpy()`, `refcount_t`, `memcpy()`

Planned for v5.16 (Jan 2022?)

- `-Wimplicit-fallthrough` enabled for Clang
- `__alloc_size` attribute
- `DECLARE_FLEX_ARRAY()` and removal of really weird remaining flexible arrays
- `struct_group()`, `memset_after()`, and `memset_startat()` for dealing with struct-member-spanning `memcpy()`/`memset()`
- `THREAD_INFO_IN_TASK` for arm32
- ongoing refactoring: size arithmetic, `scnprintf()`, `strscpy()`, `refcount_t`, `memcpy()`

Various soon and not-so-soon features

- more hardware memory tagging
- x86 CET/IBT
- x86 SMAP emulation
- `execve()` brute force detection
- write-rarely memory
- arm32 feature parity
- eXclusive Page Frame Owner
- arithmetic overflow detection
- `memcpy()` bounds checks
- Function Granular KASLR
- eXecute Only Memory
- read-only page tables
- type-aware 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/2021/lss/kspp.pdf

https://kernsec.org/wiki/index.php/KSPP

`#linux-hardening` on Libera