Kernel Self Protection Project Update

Kernel Summit, Prague
Oct 25, 2017

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

https://outflux.net/slides/2017/ks/kspp.pdf

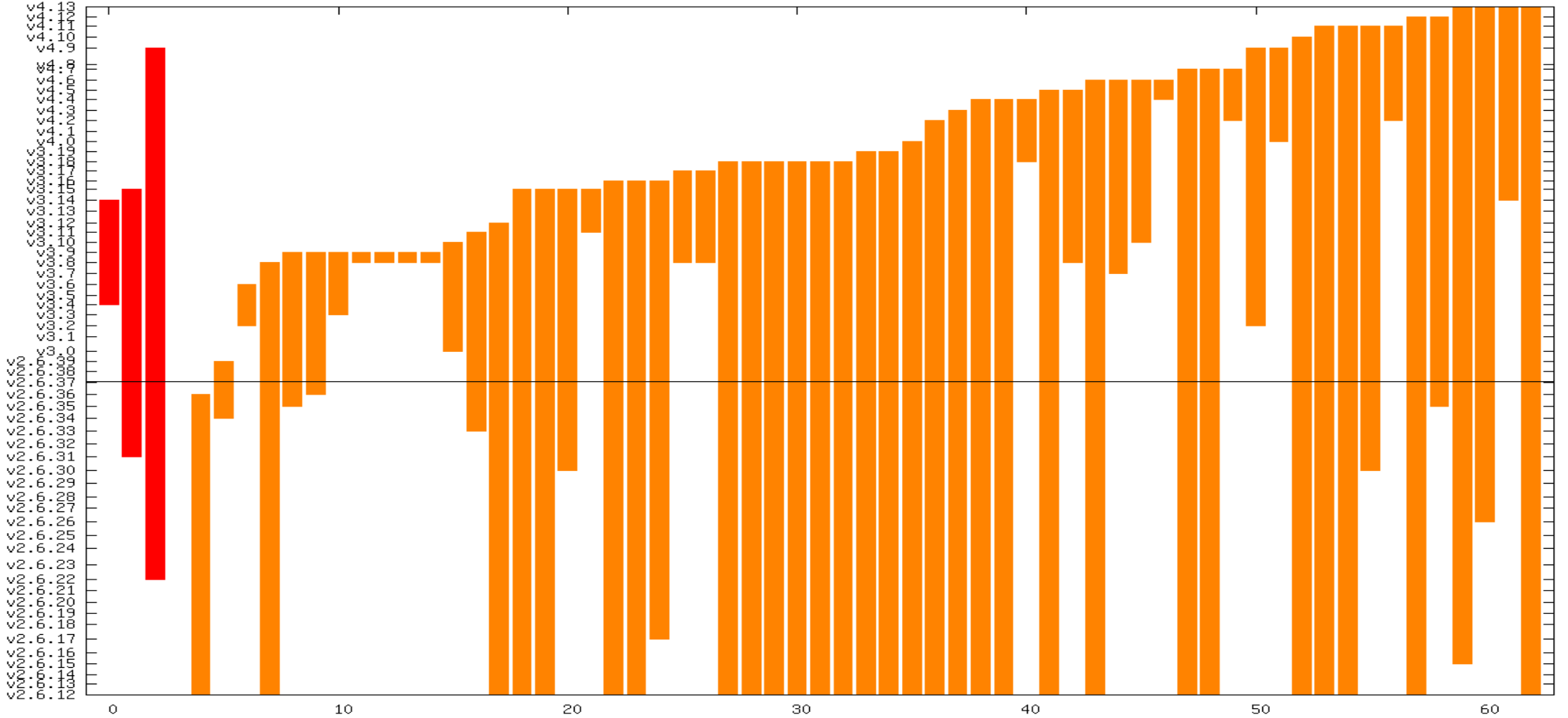
Devices using Linux

- Servers, laptops, cars, phones, …
- >2,000,000,000 active Android devices in 2017
- Vast majority are running v3.4 (with v3.10 slowly catching up)
- Bug lifetimes on devices are even longer than on upstream

Upstream Bug Lifetime

- In 2010 Jon Corbet researched security flaws, 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 2017:
  - Critical: 3 @ 5.3 years
  - High: 59 @ 6.4 years
  - Medium: 534 @ 5.6 years
  - Low: 273 @ 5.6 years

critical & high CVE lifetimes



Analogy: 1960s Car Industry

- @mricon's presentation 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 kernel
  - Lives depend on Linux

Kernel Self Protection Project

- Kill **classes** of bugs
  - Not even out-of-tree code can hit them!
  - But... we'll never kill all bug classes

- Eliminate **methods** of exploitation
  - Reduce attack surface
  - Create hostile environment for attacks
  - But... we still need to debug the kernel

http://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project

Developers under KSPP umbrella

- LF’s Core Infrastructure Initiative funded: Emese Revfy, with others pending
- Self-funded: Andy Lutomirski, Russell King, Valdis Kletnieks, Jason Cooper, Daniel Micay, David Windsor, Richard Weinberger, Richard Fellner, Daniel Gruss, Jason A. Donenfeld, Sandy Harris, Alexander Popov, Tobin Harding
- ARM: Catalin Marinas, Mark Rutland
- Canonical: Juerg Haefliger
- Cisco: Daniel Borkmann
- Docker: Tycho Andersen
- Google: Kees Cook, Thomas Garnier, Daniel Cashman, Jeff Vander Stoep, Jann Horn, Eric Biggers
- Huawei: Li Kun
- IBM: Michael Ellerman, Heiko Carstens, Christian Borntraeger
- Imagination Technologies: Matt Redfearn
- Intel: Elena Reshetova, Hans Liljestrand, Casey Schaufler, Michael Leibowitz, Dave Hansen, Peter Zijlstra
- Linaro: Ard Biesheuvel, David Brown, Arnd Bergmann
- Linux Foundation: Greg Kroah-Hartman
- Oracle: James Morris, Quentin Casasnovas, Yinghai Lu
- RedHat: Laura Abbott, Rik van Riel, Jessica Yu, Baoquan He

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 (canary value can be exposed)
  - Address Space Layout Randomization (offset can be exposed)

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)

Bug classes ...

Bug class: stack overflow and exhaustion

Exploit example:

- https://jon.oberheide.org/files/half-nelson.c

- Mitigations:
  - **stack canaries, e.g. gcc's -fstack-protector (v2.6.30) and -fstack-protector-strong (v3.14)**
  - guard pages (e.g. GRKERNSEC_KSTACKOVERFLOW)
    - **vmap stack (v4.9 x86, v4.14 arm64), removal of thread_info from stack (v4.9 x86, v4.10 arm64)**
  - *alloca checking (e.g. PAX_MEMORY_STACKLEAK): Alexander Popov*
  - shadow stacks (e.g. Clang SafeStack)

Bug class: integer over/underflow

- Exploit examples:
  - https://cyseclabs.com/page?n=02012016
  - http://perception-point.io/2016/01/14/analysis-and-exploitation-of-a-linux-kernel-vulnerability-cve-2016-0728/
- Mitigations:
  - check for refcount overflow (e.g. PAX_REFCOUNT)
    - *refcount_t: Elena Reshetova, Peter Zijlstra, Hans Liljestrand, David Windsor, Ard Biesheuvel, Li Kun*
  - compiler plugin to detect multiplication overflows at runtime (e.g. PAX_SIZE_OVERFLOW, Clang -fsanitize=integer)

Bug class: buffer overflows

- Exploit example:
  - http://blog.includesecurity.com/2014/06/exploit-walkthrough-cve-2014-0196-pty-kernel-race-condition.html
- Mitigations:
  - runtime validation of copy_{to,from}_user() buffer sizes (e.g. PAX_USERCOPY)
    - **CONFIG_HARDENED_USERCOPY (v4.8)**
    - *Usercopy whitelisting: David Windsor*
    - *Usercopy slab segregation: David Windsor*
  - metadata validation (e.g. glibc's heap protections)
    - **linked-list hardening (from grsecurity) CONFIG_DEBUG_LIST (v4.10)**
    - **heap freelist obfuscation (from grsecurity) CONFIG_SLUB_HARDENED (v4.14)**
    - *Heap canaries: Daniel Micay*
  - FORTIFY_SOURCE (inspired by glibc), check str*/mem*() buffer sizes at compile- and run-time
    - **CONFIG_FORTIFY_SOURCE (v4.13)**
    - *Intra-object checking: Daniel Micay*

Bug class: format string injection

- Exploit example:
  - http://www.openwall.com/lists/oss-security/2013/06/06/13
- Mitigations:
  - **Drop %n entirely (v3.13)**
  - detect non-const format strings at compile time (e.g. gcc's -Wformat-security, or better plugin)
  - detect non-const format strings at run time (e.g. memory location checking done with glibc's -D_FORITY_SOURCE=2)
  - (Can we get rid of %p? Stay tuned...)

Bug class: kernel pointer exposure

- Exploit examples:
  - examples are legion: /proc (e.g. kallsyms, modules, slabinfo, iomem), /sys, **INET_DIAG (v4.1)**, etc
  - http://vulnfactory.org/exploits/alpha-omega.c
- Mitigations:
  - **kptr_restrict sysctl (v2.6.38)** too weak: requires dev opt-in
  - remove visibility to kernel symbols (e.g. GRKERNSEC_HIDESYM)
  - *block usage of %p or similar writes to dmesg, seq_file, or other user buffers (e.g. GRKERNSEC_HIDESYM + PAX_USERCOPY): Tobin Harding*

Bug class: uninitialized variables

- This is not just an information exposure bug!
- Exploit example:
  - https://outflux.net/slides/2011/defcon/kernel-exploitation.pdf
- Mitigations:
  - *GCC plugin, stackleak: clear kernel stack between system calls (from PAX_MEMORY_STACKLEAK): Alexander Popov*
  - **GCC plugin, structleak: instrument compiler to fully initialize all structures (from PAX_MEMORY_STRUCTLEAK): (__user v4.11, by-reference v4.14)**

Bug class: use-after-free

- Exploit example:
  - http://perception-point.io/2016/01/14/analysis-and-exploitation-of-a-linux-kernel-vulnerability-cve-2016-0728/
- Mitigations:
  - clearing memory on free can stop attacks where there is no reallocation control (e.g. PAX_MEMORY_SANITIZE)
    - **Zero poisoning (v4.6)**
  - segregating memory used by the kernel and by userspace can stop attacks where this boundary is crossed (e.g. *PAX_USERCOPY*)
  - randomizing heap allocations or using quarantines can frustrate the reallocation efforts the attack needs to perform (e.g. OpenBSD malloc)
    - **Freelist randomization (SLAB: v4.7, SLUB: v4.8)**

Exploit methods ...

Exploitation: finding the kernel

- Exploit examples (see “Kernel pointer exposure” above too):
  - https://github.com/jonoberheide/ksymhunter
- Mitigations:
  - hide symbols and kernel pointers (see “Kernel pointer exposure”)
  - kernel ASLR
    - text/modules base: **x86 (v3.14), arm64 (v4.6), MIPS (v4.7),** *ARM: Ard Biesheuvel*
    - memory: **x86 (v4.8)**
    - PIE: **arm64 (v4.6)**, *x86: Thomas Garnier*
  - runtime randomization of kernel functions
  - executable-but-not-readable memory
    - Initial support: **x86 (v4.6), arm64 (v4.9)**, needs real hardware and kernel support
  - per-build structure layout randomization (e.g. GRKERNSEC_RANDSTRUCT)
    - **manual (v4.13), automatic (v4.14)**

Exploitation: direct kernel overwrite

- How is this still a problem in the 21st century?
- Exploit examples:
  - Patch setuid to always succeed
  - http://itszn.com/blog/?p=21 Overwrite vDSO
- Mitigations:
  - Executable memory cannot be writable (CONFIG_STRICT_KERNEL_RWX)
    - **s390: forever ago**
    - **x86: v3.18**
    - **ARM: v3.19**
    - **arm64: v4.0**
    - **powerpc64: v4.13**

Exploitation: function pointer overwrite

- Also includes e.g. vector tables, descriptor tables, etc
- Exploit examples:
  - https://outflux.net/blog/archives/2010/10/19/cve-2010-2963-v4l-compat-exploit/
  - https://blogs.oracle.com/ksplice/entry/anatomy_of_an_exploit_cve
- Mitigations:
  - read-only function tables (e.g. PAX_CONSTIFY_PLUGIN)
  - make sensitive targets that need one-time or occasional updates only writable during updates (e.g. PAX_KERNEXEC):
    - **__ro_after_init (v4.6)**
  - *struct timer_list .data field removal*

Exploitation: userspace execution

- Exploit example:
  - See almost all previous examples
- Mitigations:
  - hardware segregation: **SMEP (x86), PXN (ARM, arm64)**
  - emulated memory segregation via page table swap, PCID, etc (e.g. PAX_MEMORY_UDEREF):
    - **Domains (ARM: v4.3)**
    - **TTBR0 (arm64: v4.10)**
    - *PCID (x86): Andy Lutomirski*
  - compiler instrumentation to set high bit on function calls

Exploitation: userspace data

- Exploit examples:
  - https://github.com/geekben/towelroot/blob/master/towelroot.c
  - http://labs.bromium.com/2015/02/02/exploiting-badiret-vulnerability-cve-2014-9322-linux-kernel-privilege-escalation/
- Mitigations:
  - hardware segregation: **SMAP (x86), PAN (ARM, arm64)**
  - emulated memory segregation via page table swap, PCID, etc (e.g. PAX_MEMORY_UDEREF):
    - **Domains (ARM: v4.3)**
    - **TTBR0 (arm64: v4.10)**
    - *PCID (x86): Andy Lutomirski*
  - *eXclusive Page Frame Ownership: Tycho Andersen, Juerg Haefliger*

Exploitation: reused code chunks

- Also known as Return Oriented Programming (ROP), Jump Oriented Programming (JOP), etc
- Exploit example:
  - http://vulnfactory.org/research/h2hc-remote.pdf
- Mitigations:
  - JIT obfuscation (e.g. BPF_HARDEN):
    - **eBPF JIT hardening (v4.7)**
  - compiler instrumentation for Control Flow Integrity (CFI):
    - Clang CFI https://clang.llvm.org/docs/ControlFlowIntegrity.html
    - kCFI https://github.com/kcfi/docs
    - GCC plugin: Return Address Protection, Indirect Control Transfer Protection (e.g. RAP) https://pax.grsecurity.net/docs/PaXTeam-H2HC15-RAP-RIP-ROP.pdf

A year's worth of kernel releases ...

Added in v4.10

- PAN emulation, arm64
- thread_info relocated off stack, arm64
- Linked list hardening
- RNG seeding from UEFI, arm64
- W^X detection, arm64

Added in v4.11

- refcount_t infrastructure
- 2 refcount_t conversions
- read-only usermodehelper
- structleak plugin (__user mode)

Added in v4.12

- 57 refcount_t conversions
- read-only and fixed-location GDT, x86
- usercopy consolidation
- read-only LSM structures
- KASLR enabled by default, x86
- stack canary expanded to bit-width of host
- stack/heap gap expanded

Added in v4.13

- 65 refcount_t conversions
- CONFIG_REFCOUNT_FULL
- CONFIG_FORTIFY_SOURCE
- randstruct plugin (manual mode)
- ELF_ET_DYN_BASE lowered

Added in v4.14

- 3 refcount_t conversions (67 remaining; 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

Maybe in v4.15

- 67 refcount_t conversions!
- usercopy whitelisting
- struct timer_list .data field removal

Various soon and not-so-soon features

- stackleak plugin
- eXclusive Page Frame Owner
- KASLR, arm
- SMAP emulation, x86
- %p output hashing
- brute force detection
- write-rarely memory
- Clang plugins
- Control Flow Integrity
- integer overflow detection
- VLA removal (-Werror=vla)
- per-task stack canary, non-x86
- 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/2017/ks/kspp.pdf

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