The State of Kernel Self Protection

Linux Conf AU, Sydney
Jan 26, 2018

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

https://outflux.net/slides/2018/lca/kspp.pdf

Agenda

- Background
  - Why we need to change what we’re doing
  - Just fixing bugs isn’t sufficient
  - Upstream development model
  - Applicability of forks
- Kernel Self Protection Project
  - Who we are
  - Bug classes
  - Exploit methods
  - How you can help

I. Lietuvių Susivažiavimas Vilniuje.

Lapkričio 21—22 (Gruodžio 4—5) d. 1905.

PROGRAMA:

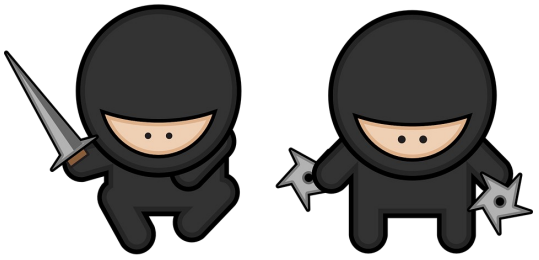
1. *Lietuvystės pradžia ir dabartis.*
2. *Lietuviai, latviai ir įvairios tautos Lietuvoje gyvenančios.*
3. *Caro manifestas nuo Spalinio 17 (30) dienos.*
4. *Lietuvos autonomija ir prikergimas Suvalkų gubernijos prie autonomiškosios Lietuvos.*
5. *Rinkimai į Viešpatystės seimą.*
6. *Lietuvos sodiečiai po rusų valdžia.*
7. *Lietuvos mokyklos.*
8. *Apie įvairius mokesčius.*
9. *Apie žemiečių įstaigas.*
10. *Apie išeivystę (emigraciją).*
11. *Apie kareiviavimą.*

Gali buti įnešami ir kiloki klausimai.

Organizacijos Komitetas.

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*



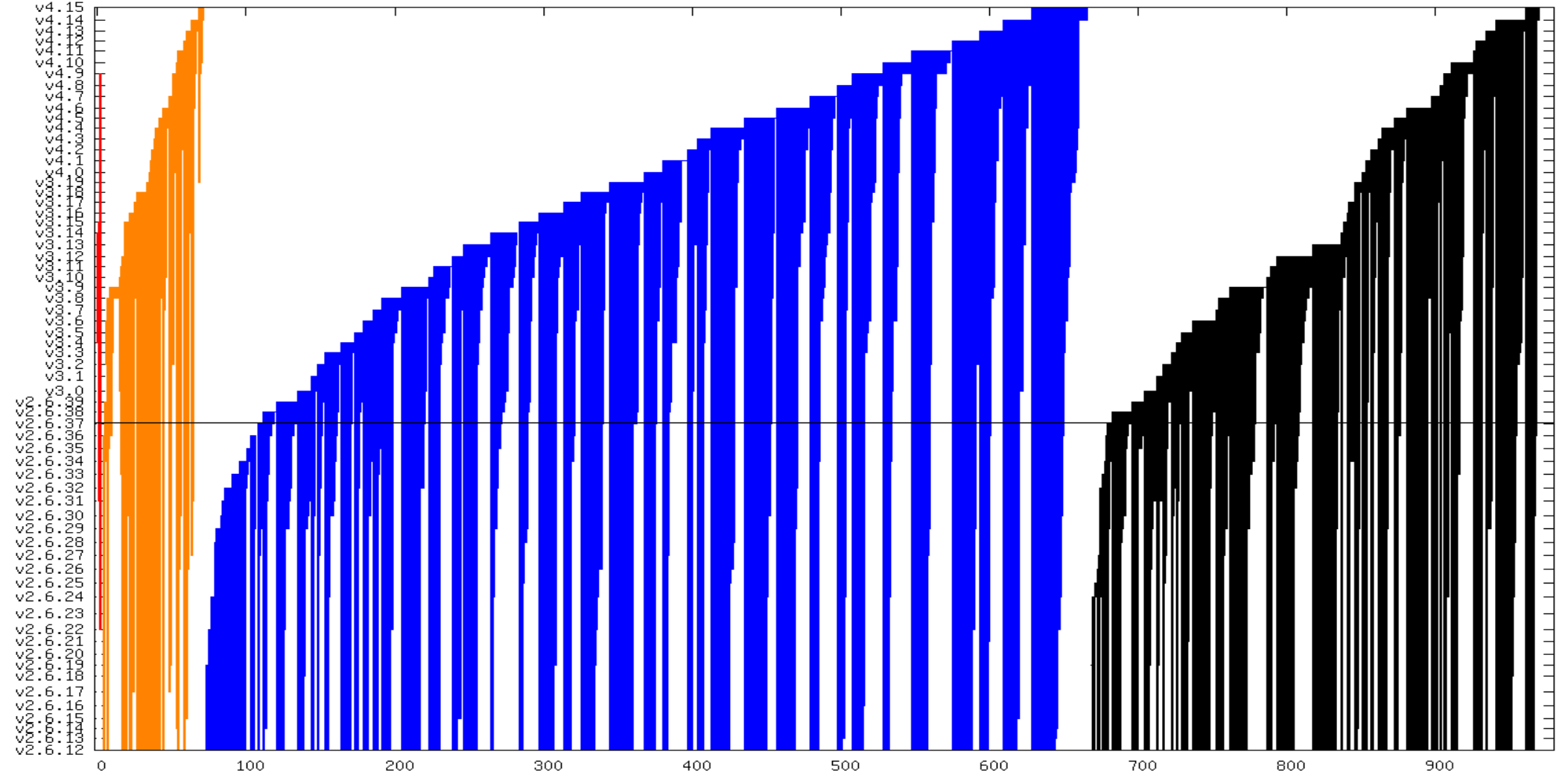
Devices using Linux

- Servers, laptops, cars, phones, TVs, toasters, …
- >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.

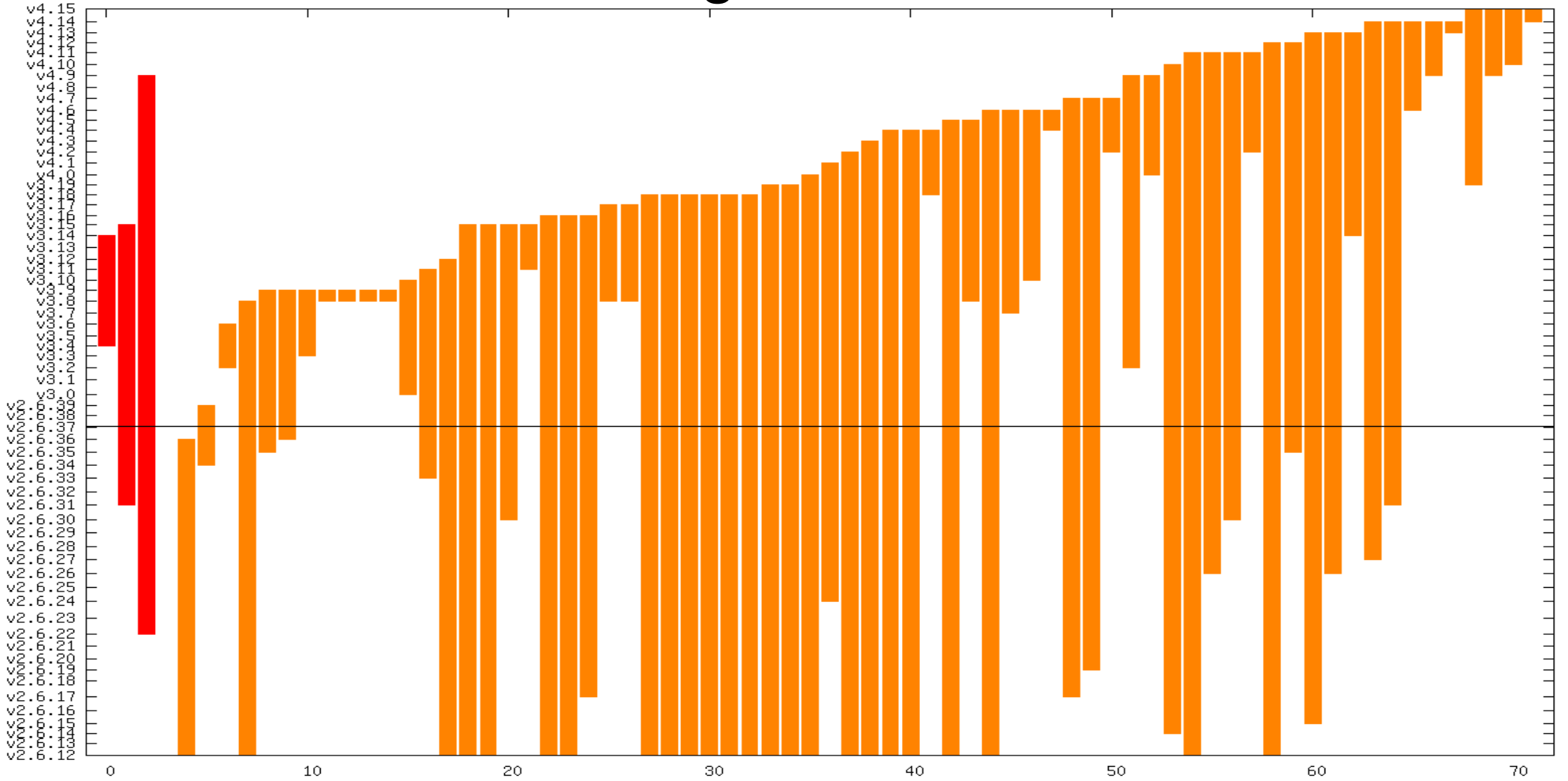
Upstream Bug Lifetime

- In 2010 Jon Corbet researched security 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 has been creeping up to 6 years:
  - Critical: 3 @ 5.3 years
  - High: 68 @ 5.7 years
  - Medium: 593 @ 5.8 years
  - Low: 302 @ 5.9 years

CVE lifetimes



critical & high CVE lifetimes



Upstream Bug Lifetime

- 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...

Fighting Bugs

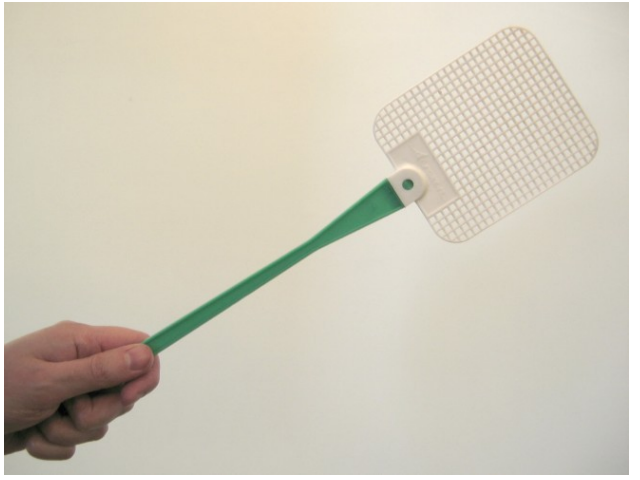
- We’re finding them
  - Static checkers: compilers, coccinelle, sparse, smatch, coverity
  - Dynamic checkers: kernel, trinity, syzkaller, KASan-family
- We’re fixing them
  - Ask Greg KH how many patches land in -stable
- They’ll always be around
  - We keep writing them
  - They exist whether we’re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

Analogy: 1960s Car Industry

- @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

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 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 leaks
- Eliminate anything that assists attackers
- *Even if it makes development more difficult*

Typical Exploit Chains

- Modern attacks tend to use more than one flaw
- Need to know where targets are
- Need to inject (or build) malicious code
- Need to locate malicious code
- Need to redirect execution to malicious code

What can we do?

- Many exploit mitigation technologies already exist (e.g. grsecurity/PaX) or have been researched (e.g. academic whitepapers), but many haven't been in upstream Linux kernel
- There is demand for kernel self-protection, and there is demand for it to exist in the upstream kernel
- http://www.washingtonpost.com/sf/business/2015/11/05/net-of-insecurity-the-kernel-of-the-argument/

Out-of-tree defenses?

- Some downstream kernel forks:
  - RedHat (ExecShield), Ubuntu (AppArmor), Android (Samsung KNOX), grsecurity (so many things)
- If you only *use* the kernel, and don't *develop* it, you're in a better position
  - But you're depending on a downstream fork
  - Fewer eyeballs (and less automated testing infrastructure) looking for vulnerabilities
  - Developing the kernel means using engineering resources for your fork
    - e.g. Android deals with multiple vendor forks already
    - Hard to integrate multiple forks
- Upstreaming means:
  - No more forward-porting
  - More review (never perfect, of course)

Digression 1: defending against email Spam

- Normal email server communication establishment:

Client	Server
[connect]	
	[accept]220 smtp.some.domain ESMTP ok
EHLO my.domain	
	250 ohai
MAIL FROM:<me@my.domain>	
	250 OK
RCPT TO:<you@your.domain>	
	250 OK
DATA	

Spam bot communication

- Success, and therefore timing, isn't important to Spam bots:

Client	Server
[connect]	
	[accept]220 smtp.some.domain ESMTP ok
EHLO my.domain	
MAIL FROM:<me@my.domain>	
RCPT TO:<you@your.domain>	
DATA	
	250 ohai
	250 OK
	250 OK

Trivially blocking Spam bots

- Insert a short starting delay (e.g. “greet_pause”, “postscreen_greet_wait”)

Client	Server
[connect]	
	[accept]
EHLO my.domain	
MAIL FROM:<me@my.domain>	
RCPT TO:<you@your.domain>	
DATA	
	554 smtp.some.domain ESMTP nope

Powerful because it's not the default

- If everyone did this (i.e. it was upstream), bots would adapt
- If a defense is unexamined and/or only run by a subset of Linux users, it may be accidentally effective due to it being different, but may fail under closer examination
- (On the flip side, it is worth noting that heterogeneous environments tend to be more resilient.)

Digression 2: Stack Clash research in 2017

- Underlying issues were identified in 2010
  - Fundamentally, if an attacker can control the memory layout of a setuid process, they may be able to manipulate it into colliding stack with other things, and arranging related overflows to gain execution control.
  - Linux tried to fix it with a 4K gap
  - grsecurity (from 2010 through at least their last public patch) took it further with a configurable gap, defaulting to 64K

A gap was not enough

- In addition to raising the gap size, grsecurity sensibly capped stack size of setuid processes, just in case:

```
do_execveat_common(...) {
    ...
    /* limit suid stack to 8MB
     * we saved the old limits above and will restore them if this exec fails */
    if (((!uid_eq(bprm->cred->euid, current_euid())) ||
         (!gid_eq(bprm->cred->egid, current_egid()))) &&
           (old_rlim[RLIMIT_STACK].rlim_cur > (8 * 1024 * 1024)))
               current->signal->rlim[RLIMIT_STACK].rlim_cur = 8 * 1024 * 1024;
    ...
```

Upstreaming the setuid stack size limit

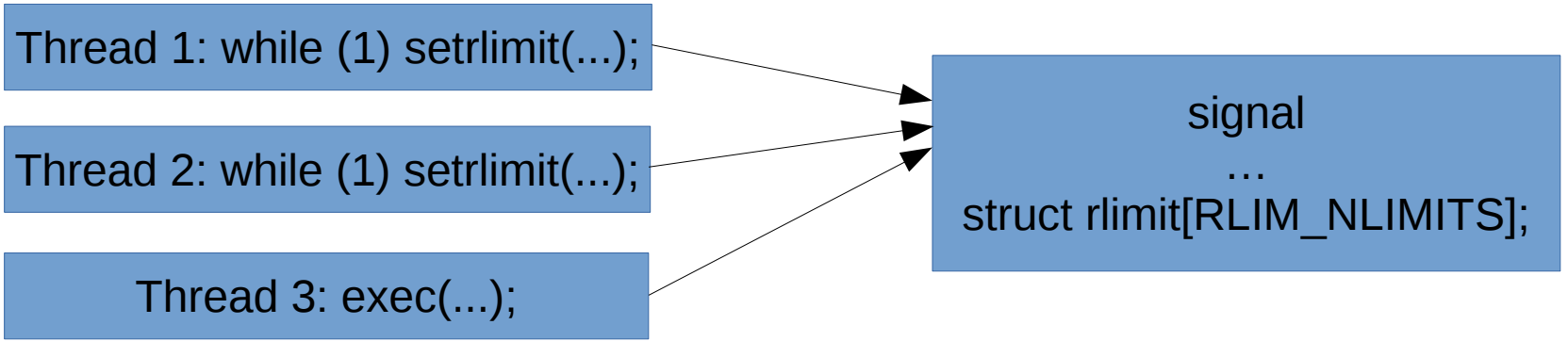
- Landed in v4.14-rc1 (and still wasn't complete, oops)
- 15 patches
- Reviewed by at least 9 other people
- Made the kernel smaller
- Stops one problem with the stack limit for setuid exec

```
16 files changed, 91 insertions(+), 159 deletions(-)
```

- Fixes for the second problem are on the way! (Hopefully in 4.16)

Important detail: threads

- Stack rlimit is a single value shared across entire thread-group
- Exec kills all other threads (part of the “point of no return”) as late in exec as possible
- If you check or set rlimits before the point of no return, you're racing other threads via setrlimit() (and processes via prlimit())



Un-upstreamed and unexamined for seven years

```
$ uname -r
4.9.24-grsec+
$ ulimit -s
unlimited
$ ls -la setuid-stack
-rwsrwxr-x 1 root root 9112 Aug 11 09:17 setuid-stack
$ ./setuid-stack
Stack limit: 8388608
$ ./raise-stack ./setuid-stack
Stack limit: 18446744073709551615
```

Out-of-tree defenses need to be upstreamed

- While the preceding example isn't universally true for all out-of-tree defenses, it's a good example of why upstreaming is important, and why sometimes what looks like a tiny change turns into **much** more work.

- How do we get this done?

Kernel Self Protection Project

- http://www.openwall.com/lists/kernel-hardening/
  - http://www.openwall.com/lists/kernel-hardening/2015/11/05/1
- http://kernsec.org/wiki/index.php/Kernel_Self_Protection_Project
- People interested in coding, testing, documenting, and discussing the upstreaming of kernel self protection technologies and related topics.

Kernel Self Protection Project

- There are other people working on excellent technologies that ultimately revolve around the kernel protecting *userspace* from attack (e.g. brute force detection, SROP mitigations, etc)
- KSPP focuses on the kernel protecting the *kernel* from attack
- Currently ~12 organizations and ~10 individuals working on about ~20 technologies
- Slow and steady

Developers under KSPP umbrella

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

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, but they still have real-world value
- Familiar examples:
  - passwords (most people understand not to expose these...)
  - 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
  - (Requires that the system actually follows the rules it says it does!)
- Familiar examples:
  - Read-only memory (writes will fail)
    - (Yay! Can't write with Meltdown *yet*)
  - Bounds-checking (large accesses fail)
    - (Except, har har, Spectre)

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)**, *best-effort CONFIG selected for compiler*
  - 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 (v4.11)** (e.g. PAX_REFCOUNT)
    - *refcount_t conversions: Elena Reshetova, Peter Zijlstra, Hans Liljestrand, David Windsor*
  - 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 and slab segregation: David Windsor*
  - metadata validation (e.g. glibc's heap protections)
    - **linked-list hardening (based on grsecurity) CONFIG_DEBUG_LIST (v4.10),**
    - **heap freelist obfuscation (based on grsecurity) CONFIG_SLUB_HARDENED (v4.14)**
    - *Heap canaries: Daniel Micay*
    - *Intel MPX: Hans Liljestrand, Elena Reshetova*
  - FORTIFY_SOURCE (inspired by glibc), check str*/mem*() 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)
  - **obfuscate output of %p (v4.15)**: in dmesg, seq_file, user buffers, etc (e.g. GRKERNSEC_HIDESYM + PAX_USERCOPY)

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: **arm64 (v4.6), 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, parisc: forever ago**
    - **x86: v3.18 (more completely)**
    - **ARM: v3.19**
    - **arm64: v4.0**
    - **powerpc: 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)**
    - *write-once memory: Igor Stoppa*
  - **struct timer_list .data field removal (v4.15)**

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)**
    - **PTI (x86: v4.15)**
  - compiler instrumentation to set high bit on function calls

Exploitation: userspace data access

- 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)**
  - hardware protected pointers (e.g. *ARM pointer authentication: Mark Rutland*)
  - 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 (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

Added in v4.15

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

Maybe in v4.16

- 32 refcount_t conversions?
- usercopy whitelisting
- CONFIG_CC_STACKPROTECTOR_AUTO

Various soon and not-so-soon features

- stackleak plugin
- eXclusive Page Frame Owner
- KASLR, arm
- SMAP emulation, x86
- brute force detection
- write-rarely memory
- Link-Time Optimization
- 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/2018/lca/kspp.pdf

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