Thwarting unknown bugs: hardening features in the mainline Linux kernel

ARM

Mark Rutland <mark.rutland@arm.com>
ARM Ltd

Embedded Linux Conference Europe 2016

October 11, 2016

What's the problem?

Linux has bugs today

```
git log --oneline \
--grep='Fixes:'   \
v4.7..v4.8-rc1 |  \
wc -l

503
```

The presence of bugs is practically unavoidable

- Code written by experienced engineers has bugs
- Code reviewed by subject-matter experts has bugs
- Static analysis only finds some bugs
- Testing and fuzzing only finds some bugs
- Formal methods do not scale to size and scope of project
  (30+ architectures with varied ISAs, memory models, system-level details)

All are valuable, but insufficient to rule out bugs.

Some bugs have security implications

There are **1496** CVE entries that match your search.

Name	Description
CVE-2016-6516	Race condition in the ioctl_file_dedupe_range function in fs/ioctl.c in the Linux kernel through 4.7 allows local users to cause a denial of service (heap-based buffer overflow) or possibly gain privileges by changing a certain count value, aka a "double fetch" vulnerability.
CVE-2016-6480	Race condition in the ioctl_send_fib function in drivers/scsi/aacraid/commctrl.c in the Linux kernel through 4.7 allows local users to cause a denial of service (out-of-bounds access or system crash) by changing a certain size value, aka a "double fetch" vulnerability.
CVE-2016-6198	The filesystem layer in the Linux kernel before 4.5.5 proceeds with post-rename operations after an OverlayFS file is renamed to a self-hardlink, which allows local users to cause a denial of service (system crash) via a rename system call, related to fs/namei.c and fs/open.c.
CVE-2016-6197	fs/overlayfs/dir.c in the OverlayFS filesystem implementation in the Linux kernel before 4.6 does not properly verify the upper dentry before proceeding with unlink and rename system-call processing, which allows local users to cause a denial of service (system crash) via a rename system call that specifies a self-hardlink.
CVE-2016-6187	The apparmor_setprocattr function in security/apparmor/lsm.c in the Linux kernel before 4.6.5 does not validate the buffer size, which allows local users to gain privileges by triggering an AppArmor setprocattr hook.
CVE-2016-6162	net/core/skbuff.c in the Linux kernel 4.7-rc6 allows local users to cause a denial of service (panic) or possibly have unspecified other impact via certain IPv6 socket operations.
CVE-2016-6156	Race condition in the ec_device_ioctl_xcmd function in drivers/platform/chrome/cros_ec_dev.c in the Linux kernel before 4.7 allows local users to cause a denial of service (out-of-bounds array access) by changing a certain size value, aka a "double fetch" vulnerability.
CVE-2016-6136	Race condition in the audit_log_single_execve_arg function in kernel/auditsc.c in the Linux kernel through 4.7 allows local users to bypass intended character-set restrictions or disrupt system-call auditing by changing a certain string, aka a "double fetch" vulnerability.
CVE-2016-6130	Race condition in the sclp_ctl_ioctl_sccb function in drivers/s390/char/sclp_ctl.c in the Linux kernel before 4.6 allows local users to obtain sensitive information from kernel memory by changing a certain length value, aka a "double fetch" vulnerability.
CVE-2016-5829	Multiple heap-based buffer overflows in the hiddev_ioctl_usage function in drivers/hid/usbhid/hiddev.c in the Linux kernel through 4.6.3 allow local users to cause a denial of service or possibly have unspecified other impact via a crafted (1) HIDIOCGUSAGES or (2) HIDIOCSUSAGES ioctl call.
CVE-2016-5828	The start_thread function in arch/powerpc/kernel/process.c in the Linux kernel through 4.6.3 on powerpc platforms mishandles transactional state, which allows local users to cause a denial of service (invalid process state or TM Bad Thing exception, and system crash) or possibly have unspecified other impact by starting and suspending a transaction before an exec system call.
CVE-2016-5728	Race condition in the vop_ioctl function in drivers/misc/mic/vop/vop_vringh.c in the MIC VOP driver in the Linux kernel before 4.6.1 allows local users to obtain sensitive information from kernel memory or cause a denial of service (memory corruption and system crash) by changing a certain header, aka a "double fetch" vulnerability.

("linux kernel" CVEs on mitre.org, 2016-09-27 - `https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=linux+kernel`)

Adversaries find bugs before we do

[redacted] vs x86_64 Linux Kernel

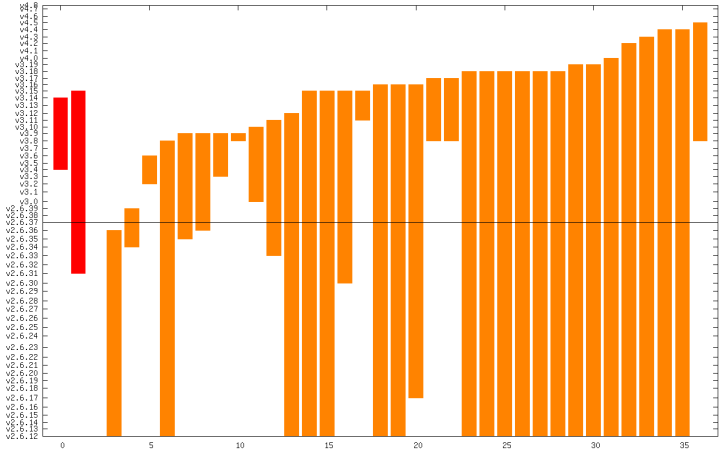
From: [redacted]
Date: Wed, 15 Sep 2010 22:08:23 -0700 (PDT)

```
/*

[redacted] Vs Linux Kernel x86_64 0day

Today is a sad day..

R.I.P.
Tue, 29 Apr 2008  /  Tue, 7 Sep 2010

a bit of history:
MCAST_MSFILTER Compat mode bug found... upon commit! (2 year life on this one)
---------------------------------------------------------------

Thanks you for signing-off on this one guys.

This exploit has been tested very thoroughly
over the course of the past few years on many many targets.

Thanks to redhat for being nice enough to backport it into early
kernel versions (anything from later August 2008+)

Exploit attached. Another 0day bites the dust and goes into our public exploit pack :)
[redacted] brings you ABftw.c - Linux Kernel x86_64 local not0dayanymore exploit.
```

Attachment: ABftw.c
Description:

(Trimmed and redacted announcement - http://seclists.org/fulldisclosure/2010/Sep/268)

Bugs go unnoticed upstream for a long time – 5+ years



(Kees Cook, LSS2016, 'Status of the Kernel Self Protection Project' - `https://outflux.net/slides/2016/lss/kspp.pdf`)

The big picture

- Linux has bugs today, and new bugs will be introduced
- ... these will be attacked in the wild
- ... and we will fix them too late

Vulnerabilities can outlast devices!

Hardening

Making bugs more difficult to exploit

- We don't know about specific bugs, but we see recurring classes, e.g.
  - Stack buffer overflow
  - Dereference of `__user` pointers
- We can attack classes with common protections
  - cover all instances
  - ... including those we don't know about yet
- Complementary to usual bug fixing
  - Not 100% effective
  - Reduces exploitability, does not fix underlying issues

Hardening in mainline

- Kernel hardening / self-protection / etc is a hot topic now
  - Lots of work happening (upstream & elsewhere)
  - ... but there's lots to do
  - ... and most devices aren't running v6.5.x yet
- Mainline has some hardening features already
  - There may be better options not (yet) in mainline
  - ... but these are available today
  - ... they're maintained and improving
  - ... and turning them on is easy
  - ... yet they're not used as often as they could be

Strict kernel memory permissions

Lax kernel memory permissions

- Typically Linux maps all kernel memory with RWX permissions
  - ... so kernel code can be modified
  - ... and `const` data can be modified
  - ... and data can be executed
- These permissions are useful when building attacks
  - ... are (almost always) useless to us
  - ... and use typically indicates a bug

Minimise kernel memory permissions

- Have the MMU enforce minimal memory permissions:
  - Map code (e.g. `.text`) as read-only, executable
  - Map const data (e.g. `.rodata`) as read-only, non-executable
  - Map data (e.g. `.data`) as read-only, non-executable
- Some arch-specific code required
  - Page tables may be rewritten after boot
  - Temporary RW mappings for deliberate code-patching

CONFIG_DEBUG_RODATA

- Badly named: not just a debug feature
- Small amount of additional padding inserted between `.text`/`.data`/`.rodata`
- No changes required to core code, libraries, drivers, etc
- No runtime overhead (ignoring a small amount of TLB pressure)
- Available on arm, arm64, parisc, s390, x86-32, x86-64
- Will be mandatory on arm64 & x86 in v4.9

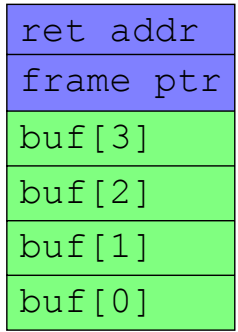
CONFIG_DEBUG_SET_MODULE_RONX

- Just like CONFIG_DEBUG_RODATA, but for modules
  - ... likewise, not just a debug feature
  - No changes required to core code, libraries, drivers, etc
  - No runtime overhead (ignoring a small amount of TLB pressure)
- Available on arm, arm64, s390, x86-32, x86-64

Stack smashing protection

Stack smashing

- Stacks typically contain return address and other data
- Stacks grow downwards, buffers grow upwards[1]
- Often, buffer size checks are missed (e.g. bad `strcpy()` use)
- Attacker-controlled buffer overflow can change return address



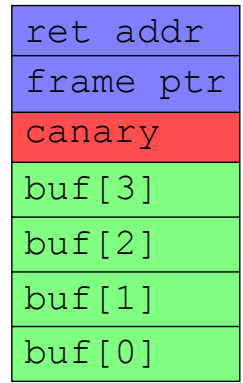
[1] On many, but not all architectures

Stack smashing protection

- Have compiler insert code to detect stack buffer overflow
  - At function entry, place canary between return address and data
  - At function return, verify the canary is unchanged
- Small piece of arch-specific bootstrap code required
- No changes required to core code, libraries, drivers, etc

Beware:

- A known canary value can be spoofed
- Other data (e.g. local variables) can still be corrupted



CONFIG_STACKPROTECTOR_REGULAR

- Protects all functions with a local character array (8 bytes +)
- Affects ~3% of functions, increases kernel size by ~0.3%
- Requires GCC 4.2+
- Available on arm, arm64, mips, sh, x86-32, x86-64

CONFIG_STACKPROTECTOR_STRONG

- Protects all functions which:
  - have a local array (of any type or length), or
  - pass a stack address to a function, or
  - assign a stack address to a variable, or
  - use register local variables
- Affects ~20% of functions, increases kernel size by ~2%
- Requires GCC 4.9+ (introduced in v3.13)
- Available on arm, arm64, mips, sh, x86-32, x86-64

User/Kernel memory segregation

A shared address space

- Typically, user and kernel memory share the same address space
  - the same load/store instructions can access both
  - ... thus accidental `__user` data dereferences aren't caught
  - ... nor are accidental branches to `__user` code
  - ... leaving a very powerful primitive for exploits
- Logically, the two memory spaces are distinct
  - userspace can't access kernel memory
  - kernel only needs to access kernel memory to function
  - kernel access to user memory rare and explicit (e.g. `get_user()`)

Un-sharing the address space

- Some MMUs can't automatically limit kernel access to mapped memory
  - ... but all have the ability to switch page tables
  - ... or other manual access control features (e.g. domains on ARM)
- We can explicitly disallow access to user memory when we don't need it
  - upon entry to kernel, unmap (or disable access to) user memory
  - upon exit from kernel, map (or enable access to) user memory
  - within `get_user()` and friends, enable access to user memory
- Adds some overhead to kernel entry/exit, `get_user()` and friends

Stricter permissions

- Some MMUs can automatically prevent kernel execution of user memory
  - e.g. PXN on arm/arm64, SMEP on x86
  - ... we use these automatically when available
  - prevents arbitrary code execution
  - ... but code-reuse attacks are still possible
- Recent MMUs can automatically prevent kernel access to user memory
  - e.g. PAN on arm64, SMAP on x86
  - some arch-specific work in `get_user()` and friends required
  - prevents implicit `__user` memory accesses
  - ... validation sill necessary after `get_user()` and friends

Segregation options

- `CONFIG_CPU_SW_DOMAIN_PAN` (arm)
  - Low 1MB unprotected if vectors aren't remappable (ancient parts)
  - Requires domains, only available when using short descriptors
- `CONFIG_ARM64_PAN` (arm64)
  - Can be enabled on generic kernel – patched in as necessary
  - Only effective when PAN is present in HW (ARMv8.1)
- `CONFIG_X86_SMAP` (x86)
  - Can be enabled on generic kernel – patched in as necessary
  - Only effective when SMAP is present in HW

In all cases, no changes required to core code, libraries, drivers, etc

More effective testing

CONFIG_KASAN_OUTLINE, CONFIG_KASAN_INLINE

- Not a hardening feature, but very useful when testing
- Byte-granular use-after-free and out-of-bounds detection
- Requires GCC 4.9.2+
  - requires GCC 5.0+ to handle stack-local variables and globals
- Outline keeps kernel small, inline is faster
- Available on arm64, x86-64

CONFIG_UBSAN

- Detects undefined behaviour at runtime
- May have false positives in some edge cases
- Requires GCC 4.9+
- Avaialble on arm, arm64, powerpc, x86-32, x86-64

Questions?

ARM