tales from the trenches

Presented by
Kyle McMartin
kyle@{redhat.com,fedoraproject.org}

slides: kyle.fedorapeople.org/devconf-2015-02-07.pdf

fedora

about me?

- Fedora / Red Hat for 8 years
- Ubuntu before that
- Debian before even that

about me?

- contributor to many linux ports
  - especially hppa & ia64
  - and now arm64
- mostly generalist

- working on kernel/toolchain bugs

what am i speaking about?

- looked back at year in review
- almost all bugs fixed had 1-liner patches
- how did all of these come about?
- debugging strategies
  - and coping mechanisms

what am i speaking about?

- generally peter robinson or paul whalen ask me to take a look at something
  - usually either a FTBFS, or software crashing.

6 main bugs today

- All in the kernel or gcc/glibc
- Resulted in a mix:
  - runtime failures
  - kernel panic
  - program crash

6 main bugs today

- will attempt to use these bugs to illustrate technical details

1: python failing

- dmarlin reported anaconda failing when dlopen-ing in python
  - started fine, but eventually fell over
  - message about exhaustion of TLS slots

what is TLS?

- thread local storage
- mechanism for private data access on a per-thread basis
- sort of like per-cpu in the kernel for threads

static versus dynamic

- dynamic TLS is most flexible model
- results in a function call into libc
  - which looks up the address in the per-thread TLS area and returns it
- static TLS is set aside at startup time
  - “best guess” for how much is needed based on static links

static versus dynamic

- obviously can't know how many objects will be dlopen'd
  - leave some space, hope for the best

TLSDESC

- invented by alex oliva
- optimisation for dynamic tls
  - optional on most architectures (x86_64, arm...)
- greedily uses space set aside for static TLS as a further optimisation

TLSDESC

- Except if all static TLS space is consumed by TLSDESC use, can't dlopen anymore.
- Trivial workaround to always fallback to dynamic TLS
- This got python/anaconda going again

Better Fix

- Set a flag when loading via dlopen and fall back to dynamic TLS
- Same code path used by both ld.so and dlopen loading, so need to distinguish, otherwise makes static link case worse.

2: kernel panic

- reproducible kernel crash caused by unprivileged user
  - AWESOME (not!)
  - dd if=/dev/zero ibs=1 count=1
    - KERNEL PANIC, SPLAT!

CVE-2014-7843

- fortunately arm64 hardware not terribly common last year
- debugging this from the kernel dump...

kernel panic (reduced)

```
Unable to handle kernel paging request at virtual address 1cc90000
pgd = fffffe03d8410000
*pgd=00000043d0410003, *pud=00000043d0410003, *pmd=00000043d0410003, *pte=0000000000000000
[...]
CPU: 3 PID: 2866 Comm: dd Tainted: G           E  3.17.0-0.49.sa2.jkkm4.aarch64 #1
task: fffffe03d76f5a00 ti: fffffe03de078000 task.ti: fffffe03de078000
PC is at __clear_user+0x40/0x50
LR is at read_zero+0x60/0xc4
pstate: 60000145
sp : fffffe03de07bdf0
x29: fffffe03de07bdf0 x28: fffffe03de078000
x27: fffffe0000bda000 x26: 000000000000003f
x25: 0000000000000118 x24: 0000000000010000
x23: fffffe03de078000 x22: 0000000000000000
x21: 000000001cc90000 x20: 0000000000000001
x19: 0000000000000001 x18: 000003ffc75195e0
x17: 0000000000420208 x16: fffffe00001dac08
x15: 003b9aca00000000 x14: 002422cd6e000000
x13: ffffffffabac2edb x12: 0000000000000018
x11: 0000000001165666b x10: 00000000ffffffff
x9 : 0000000000bf5a48 x8 : 000000000000003f
x7 : fffffe0000c90000 x6 : cb88537fdc8cb300
x5 : 0000000000000000 x4 : fffffe0000406d98
x3 : 0000000000000001 x2 : 0000000000000001
x1 : 0000000000000000 x0 : 000000001cc90000
[...]
Call trace:
__clear_user+0x40/0x50
vfs_read+0x84/0x198
SyS_read+0x4c/0xb0
Code: 7800241f d1000821 b1000421 54000044 (3900001f)
```

what does this tell us?

- address is 0x1cc90000
  - 64K page aligned
  - user address (how do we know?)
    - kernel is in upper-half of 64-bit address space
    - hence all the 0xFFFFFE... addresses

what does this tell us?

- PC is at __clear_user
  - zero-ing a user string
  - Architecture-specific function (suspicious automatically since we've not seen this bug elsewhere...)
- LR is at read_zero
  - makes sense, we're using /dev/zero after all

what does this tell us?

- Code: 7800241f d1000821 b1000421 54000044 (3900001f)
- 0x3900001F is the failing instruction
- Decode using as/objdump
- 0: 3900001f strb wzr, [x0]
  - store byte to [x0]
    - x0 is faulting address

user access and fixups

- optimisation of user access
- assume it will work, and handle it if the user pointer is invalid
- kernel will take a fault, fix it up, and continue executing

user access and fixups

- however, we must annotate these, so the kernel knows how to handle them for a given instruction
- USER(9f, str xzr, [x0], #8)
  - creates a fixup in the kernel for the specific instruction
  - on failure, branch to label 9 address

user access and fixups

- this USER() annotation was missing on the single byte strb case
  - resulted in the kernel being unable to handle single byte dd
  - 9 byte would be fine, because the first 8-byte access correctly fixed it up

3: glibc crash

- while generating locale data during build
- older builds had been fine, but latest snapshot failed
- first step: what changed?
  - git diff -p –stat $snapn-1 $snapn

3: glibc crash

- immediately see a .S file added in aarch64 support
  - ok, that's definitely suspicious.

3: glibc crash

- let's revert that commit and see...
  - and the build passes, and testsuite shows no regressions...
- ok, that's weird, this looks pretty sane
- ask around, apparently well tested assembly code

Working backwards

- We have a crash, but everything looks fairly sane there
- Why are we crashing?
- Work backwards, get reproducible

Comparison debugging

- Hack glibc to call both assembler and generic C version based on env. variable
- Run both in gdb, compare state at point of crash
- Eliminate possible skew, compare return from strchrnul

Something funky

- Compare state on entry and exit of both functions
- Weird, v15 varied in assembler version (not entirely unexpected, since it's used)
- However, not save/restored by caller!
- Time to look up the ABI

ABI

- Tells us the calling conventions for functions
- Allows compiler writers & toolchain people to write code that can call each other
- Details where arguments are in registers, how the stack is laid out

ABI

- Also optimisations...
  - If certain registers aren't used, they can be ignored
  - Avoids saving/restoring all GPRs/FPRs unnecessarily on every function call
  - callee-saves versus caller-saves

ABI

- On Aarch64, %v15 is callee-saved
  - But used as if caller-saved by strchrnul.S
- One-line patch to pick a different register which is, and suddenly things work again

4: gcc code gen bug

- libcap-ng failed to set process caps
  - errored in its test suite
  - uses TLS to store thread caps
- tickled by “fix” for python above
- not immediately obvious how it's possible

Back to dynamic TLS

- Results in a function call
- Previously would have used static TLS which was fixed up at load-time

- Try to figure out how we fell over

Narrow the problem

- Re-build qemu without optimisation
  - This works properly
- Now, start optimising individual objects until it fails again

Narrow the problem

- Now, the real fun, grind away the optimisation passes in gcc until we figure out which fails
- Compare assembly output between pass and fail

Narrow the problem

- OK, wow, that's a pretty big diff...
- But, we know where it fails, so only look at those pieces of assembly where things changed
- Movement of a comparison across the TLS assembly sequence

Weirdness in GCC

- Moving a comparison across a function call is risky
- Why? Functions can have side effects the compiler doesn't know about
- Bug in GCC's aarch64 support

Weirdness in GCC

- rth spots the problem, the TLSDESC sequence in gcc wasn't clobbering the condition register
- Anything code which moved a comparison around a TLS variable could be broken
- Mass rebuild needed with fixed gcc

Mass rebuilds suck

- Most of the time... new FTBFS can creep in since we're building in a different order than what was tested
- Can we avoid doing one?
  - YES!

glibc hack

- Workaround by saving/restoring the condition code to the stack in the assembly wrapper
- Four line hack avoids rebuilding world

- glibc-aarch64-workaround-nzcv-clobber-in-tlsdesc.patch

5: keyctl testsuite failure

- jbastian reported PAGE_SIZE-1 keyctl commands were failing
  - obviously that's immediately suspicious
  - reminded me of a bug from last year
    - strnlen_user had an off-by-one

strnlen(3)

- size_t strnlen(const char *s, size_t maxlen);
  - number of bytes in s
    - less the terminating NUL
    - but at most maxlen
  - looks only at maxlen bytes worth
- return strlen(s) < maxlen ? strlen(s) : maxlen;

strnlen_user

- kernel semantics differ from userspace
  - returns size of string INCLUDING NUL
  - if strlen(s) > maxlen, returns maxlen+1
  - returns 1 if s[0] = NUL
  - returns 0 if s is a bad user pointer or maxlen is invalid (ie: negative or whatnot)

strnlen_user

- needed to handle max+1 condition
  - previously truncated to max-1
- new bug had was similar, but had the issue where n == max, not max >= n
- handle that case, suddenly PAGE_SIZE sized strings work as expected

strnlen_user

- illustrates the danger of similarly (or same) named APIs
  - when semantics differ in subtle ways
  - when they're not exercised properly
- importance of writing test-cases for kernel code if it can be lifted to userspace
  - exercise syscalls before merge
  - ditto any optimised functions

6: qemu failure

- richard jones reporting qemu failing in rawhide on a TLS variable (noticing a theme?)
  - failed when built -pie -O2
  - succeeded when build -O0

6: qemu failure

- Narrowing down build flags used
- -pie wasn't the problem
- neither was optimisation
- On a hunch, forced local exec model for TLS
  - that worked too, ok, weird

6: qemu failure

- Local Exec TLS model forces static TLS
- Means no extra GOT entries allocated (since not dynamic)
  - Reloc processing fixes up directly to point into TLS space

All about GOT

- Global Object Table
  - Fundamentally, a list of addresses so that PIC code can find data without needing to modify program code
    - Needed to be able to share program code between processes
    - Otherwise we break CoW

All about GOT

- Dynamic TLS uses extra GOT entries
  - Housekeeping stuff that's irrelevant to discussion
- Binutils was misaccounting for the size on AArch64

Back to qemu

- Debugging qemu in gdb, the data at the addresses it was accessing looked totally bogus
- How's this possible? Test different flags, starting with combreloc which reorders the reloc table.
  - This fixed the bug, suspiciously

binutils the culprit

- combreloc was reordering with an incorrect size as a result of a bug in aarch64 support in binutils
  - backport one line fix from upstream and things are working again with relro and PIE

Questions?

feedback to http://devconf.cz/f/100

Contact:
kyle@redhat.com