VulnerabilityDetails

Corey edited this page on Jul 5, 2019 · 22 revisions

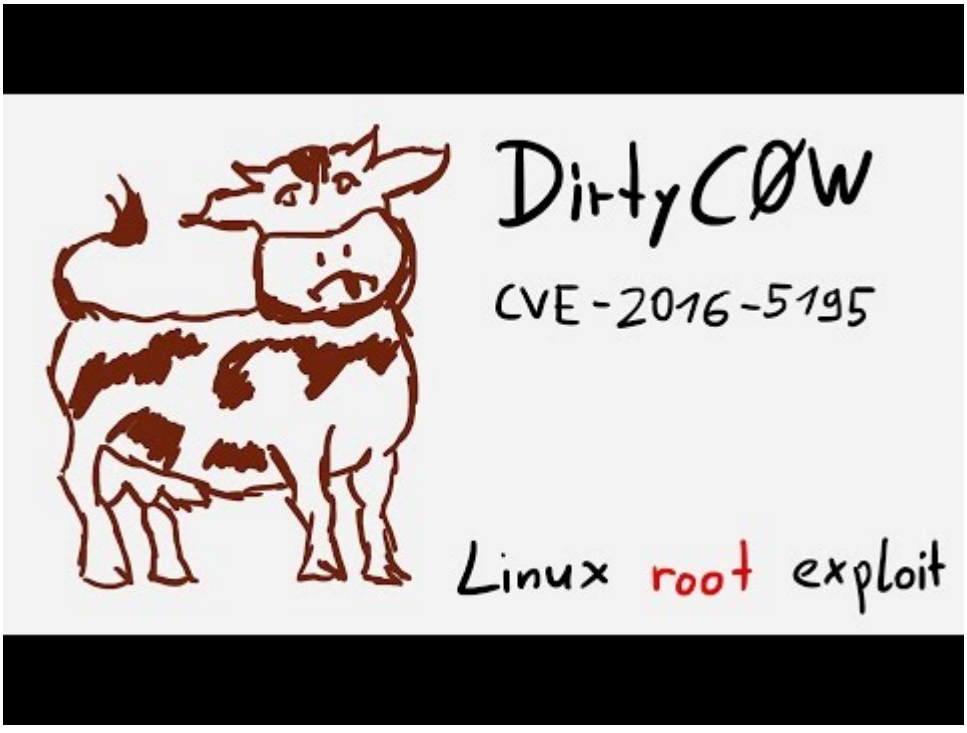
Summary

A race condition was found in the way the Linux kernel's memory subsystem handled the copy-on-write (COW) breakage of private read-only memory mappings. All the information we have so far is included in this page.

The bug has existed since around 2.6.22 (released in 2007) and was fixed on Oct 18, 2016. List of patched versions here

There are proof of concepts available here.

Video Explanation



Impact

- An unprivileged local user could use this flaw to gain write access to otherwise read-only memory mappings and thus increase their privileges on the system.
- This flaw allows an attacker with a local system account to modify on-disk binaries, bypassing the standard permission mechanisms that would prevent modification without an appropriate permission set.

Analysis

```
faultin_page
  handle_mm_fault
    __handle_mm_fault
      handle_pte_fault
        do_fault <- pte is not present
          do_cow_fault <- FAULT_FLAG_WRITE
            alloc_set_pte
              maybe_mkwrite(pte_mkdirty(entry), vma) <- mark the page dirty
                                                        but keep it RO
# Returns with 0 and retry
follow_page_mask
  follow_page_pte
    (flags & FOLL_WRITE) && !pte_write(pte) <- retry fault

faultin_page
  handle_mm_fault
    __handle_mm_fault
      handle_pte_fault
        FAULT_FLAG_WRITE && !pte_write
          do_wp_page
            PageAnon() <- this is CoWed page already
            reuse_swap_page <- page is exclusively ours
            wp_page_reuse
              maybe_mkwrite <- dirty but RO again
              ret = VM_FAULT_WRITE
((ret & VM_FAULT_WRITE) && !(vma->vm_flags & VM_WRITE)) <- we drop FOLL_WRITE

# Returns with 0 and retry as a read fault
cond_resched -> different thread will now unmap via madvise
follow_page_mask
  !pte_present && pte_none
faultin_page
  handle_mm_fault
    __handle_mm_fault
      handle_pte_fault
        do_fault <- pte is not present
          do_read_fault <- this is a read fault and we will get pagecache
                           page!
```

How

- The In The Wild exploit relied on writing to /proc/self/mem on one side of the race.
- ptrace(PTRACE_POKEDATA) can write to readonly mappings.
- The attack relies on racing the madvise(MADV_DONTNEED) system call while having the page of the executable mmapped in memory.

Commit messages

```
commit 4ceb5db9757aaeadcf8fbbf97d76bd42aa4df0d6
Author: Linus Torvalds <torvalds@g5.osdl.org>
Date:   Mon Aug 1 11:14:49 2005 -0700

Fix get_user_pages() race for write access

There's no real guarantee that handle_mm_fault() will always be able to
break a COW situation - if an update from another thread ends up
modifying the page table some way, handle_mm_fault() may end up
requiring us to re-try the operation.

That's normally fine, but get_user_pages() ended up re-trying it as a
read, and thus a write access could in theory end up losing the dirty
bit or be done on a page that had not been properly COW'ed.

This makes get_user_pages() always retry write accesses as write
accesses by making "follow_page()" require that a writable follow has
the dirty bit set.  That simplifies the code and solves the race: if the
COW break fails for some reason, we'll just loop around and try again.
```

```
commit 19be0eaffa3ac7d8eb6784ad9bdbc7d67ed8e619
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Thu Oct 13 20:07:36 2016 GMT

This is an ancient bug that was actually attempted to be fixed once
(badly) by me eleven years ago in commit 4ceb5db9757a ("Fix
get_user_pages() race for write access") but that was then undone due to
problems on s390 by commit f33ea7f404e5 ("fix get_user_pages bug").

In the meantime, the s390 situation has long been fixed, and we can now
fix it by checking the pte_dirty() bit properly (and do it better).  The
s390 dirty bit was implemented in abf09bed3cce ("s390/mm: implement
software dirty bits") which made it into v3.9.  Earlier kernels will
have to look at the page state itself.

Also, the VM has become more scalable, and what used a purely
theoretical race back then has become easier to trigger.
```

```
To fix it, we introduce a new internal FOLL_COW flag to mark the "yes,
we already did a COW" rather than play racy games with FOLL_WRITE that
is very fundamental, and then use the pte dirty flag to validate that
the FOLL_COW flag is still valid.
```

References

- https://bugzilla.redhat.com/show_bug.cgi?id=1384344
- https://access.redhat.com/security/vulnerabilities/2706661
- https://plus.google.com/+KeesCook/posts/UUaXm3PcQ4n
- https://twitter.com/nelhage/status/789196293629370368
- https://bugzilla.suse.com/show_bug.cgi?id=1004418#c14