KVM: X86: MMU: Use the correct inherited permissions to get shadow page

Browse files

```
When computing the access permissions of a shadow page, use the effective
permissions of the walk up to that point, i.e. the logic AND of its parents'
permissions.  Two guest PxE entries that point at the same table gfn need to
be shadowed with different shadow pages if their parents' permissions are
different.  KVM currently uses the effective permissions of the last
non-leaf entry for all non-leaf entries.  Because all non-leaf SPTEs have
full ("uwx") permissions, and the effective permissions are recorded only
in role.access and merged into the leaves, this can lead to incorrect
reuse of a shadow page and eventually to a missing guest protection page
fault.

For example, here is a shared pagetable:

   pgd[]   pud[]        pmd[]           virtual address pointers
                      /->pmd1(u--)->pte1(uw-)->page1 <- ptr1 (u--)
        /->pud1(uw-)--->pmd2(uw-)->pte2(uw-)->page2 <- ptr2 (uw-)
   pgd-|           (shared pmd[] as above)
        \->pud2(u--)--->pmd1(u--)->pte1(uw-)->page1 <- ptr3 (u--)
                      \->pmd2(uw-)->pte2(uw-)->page2 <- ptr4 (u--)

  pud1 and pud2 point to the same pmd table, so:
  - ptr1 and ptr3 points to the same page.
  - ptr2 and ptr4 points to the same page.

(pud1 and pud2 here are pud entries, while pmd1 and pmd2 here are pmd entries)

- First, the guest reads from ptr1 first and KVM prepares a shadow
  page table with role.access=u--, from ptr1's pud1 and ptr1's pmd1.
  "u--" comes from the effective permissions of pgd, pud1 and
  pmd1, which are stored in pt->access.  "u--" is used also to get
  the pagetable for pud1, instead of "uw-".

- Then the guest writes to ptr2 and KVM reuses pud1 which is present.
  The hypervisor set up a shadow page for ptr2 with pt->access is "uw-"
  even though the pud1 pmd (because of the incorrect argument to
  kvm_mmu_get_page in the previous step) has role.access="u--".

- Then the guest reads from ptr3.  The hypervisor reuses pud1's
  shadow pmd for pud2, because both use "u--" for their permissions.
  Thus, the shadow pmd already includes entries for both pmd1 and pmd2.

- At last, the guest writes to ptr4.  This causes no vmexit or pagefault,
  because pud1's shadow page structures included an "uw-" page even though
  its role.access was "u--".

Any kind of shared pagetable might have the similar problem when in
virtual machine without TDP enabled if the permissions are different
from different ancestors.

In order to fix the problem, we change pt->access to be an array, and
any access in it will not include permissions ANDed from child ptes.

The test code is: https://lore.kernel.org/kvm/20210603050537.19605-1-jiangshanlai@gmail.com/
Remember to test it with TDP disabled.

The problem had existed long before the commit 41074d0 ("KVM: MMU:
Fix inherited permissions for emulated guest pte updates"), and it
is hard to find which is the culprit.  So there is no fixes tag here.

Signed-off-by: Lai Jiangshan <laijs@linux.alibaba.com>
Message-Id: <20210603052455.21023-1-jiangshanlai@gmail.com>
Cc: stable@vger.kernel.org
Fixes: cea0f0e ("[PATCH] KVM: MMU: Shadow page table caching")
Signed-off-by: Paolo Bonzini <pbonzini@redhat.com>
```

master
v6.1 ··· v5.13-rc6

Lai Jiangshan authored and **bonzini** committed on Jun 8, 2021 1 parent e898da7 commit b1bd5cba3306691c771d558e94baa73e8b0b96b7

Showing **2 changed files** with **11 additions** and **7 deletions**.

Split Unified

4 Documentation/virt/kvm/mmu.rst

```
171 171       shadow pages) so role.quadrant takes values in the range 0..3.  Each
172 172       quadrant maps 1GB virtual address space.
173 173     role.access:
174     -     Inherited guest access permissions in the form uwx.  Note execute
175     -     permission is positive, not negative.
    174 +     Inherited guest access permissions from the parent ptes in the form uwx.
    175 +     Note execute permission is positive, not negative.
176 176     role.invalid:
177 177       The page is invalid and should not be used.  It is a root page that is
178 178       currently pinned (by a cpu hardware register pointing to it); once it is
```

14 arch/x86/kvm/mmu/paging_tmpl.h

```
90  90          gpa_t pte_gpa[PT_MAX_FULL_LEVELS];
91  91          pt_element_t __user *ptep_user[PT_MAX_FULL_LEVELS];
92  92          bool pte_writable[PT_MAX_FULL_LEVELS];
93      -       unsigned pt_access;
```

```
94        -         unsigned pte_access;
      93  +         unsigned int pt_access[PT_MAX_FULL_LEVELS];
      94  +         unsigned int pte_access;
95    95            gfn_t gfn;
96    96            struct x86_exception fault;
97    97    };
418   418                   }
419   419
420   420                   walker->ptes[walker->level - 1] = pte;
      421 +
      422 +                 /* Convert to ACC_*_MASK flags for struct guest_walker.  */
      423 +                 walker->pt_access[walker->level - 1] = FNAME(gpte_access)(pt_access ^ walk_nx_mask);
421   424           } while (!is_last_gpte(mmu, walker->level, pte));
422   425
423   426           pte_pkey = FNAME(gpte_pkeys)(vcpu, pte);
424   427           accessed_dirty = have_ad ? pte_access & PT_GUEST_ACCESSED_MASK : 0;
425   428
426   429           /* Convert to ACC_*_MASK flags for struct guest_walker.  */
427       -         walker->pt_access = FNAME(gpte_access)(pt_access ^ walk_nx_mask);
428   430           walker->pte_access = FNAME(gpte_access)(pte_access ^ walk_nx_mask);
429   431           errcode = permission_fault(vcpu, mmu, walker->pte_access, pte_pkey, access);
430   432           if (unlikely(errcode))
463   465           }
464   466
465   467           pgprintk("%s: pte %llx pte_access %x pt_access %x\n",
466       -                    __func__, (u64)pte, walker->pte_access, walker->pt_access);
      468 +                  __func__, (u64)pte, walker->pte_access,
      469 +                  walker->pt_access[walker->level - 1]);
467   470           return 1;
468   471
469   472   error:
643   646           bool huge_page_disallowed = exec && nx_huge_page_workaround_enabled;
644   647           struct kvm_mmu_page *sp = NULL;
645   648           struct kvm_shadow_walk_iterator it;
646       -         unsigned direct_access, access = gw->pt_access;
      649 +         unsigned int direct_access, access;
647   650           int top_level, level, req_level, ret;
648   651           gfn_t base_gfn = gw->gfn;
649   652
675   678                   sp = NULL;
676   679                   if (!is_shadow_present_pte(*it.sptep)) {
677   680                           table_gfn = gw->table_gfn[it.level - 2];
      681 +                         access = gw->pt_access[it.level - 2];
678   682                           sp = kvm_mmu_get_page(vcpu, table_gfn, addr, it.level-1,
679   683                                                 false, access);
680   684                   }
```

0 comments on commit `b1bd5cb`

Please sign in to comment.