author	Kazuki Takiguchi <takiguchi.kazuki171@gmail.com>	2022-11-23 14:36:00 -0500
committer	Paolo Bonzini <pbonzini@redhat.com>	2022-11-23 18:50:08 -0500
commit	47b0c2e4c220f2251fd8dcfbb44479819c715e15 (patch)	
tree	04a4bb3b263c55a437accbd373116db0cc24addb	
parent	d79b483193c276f9b6863f69735b97de12ced621 (diff)	
download	linux-47b0c2e4c220f2251fd8dcfbb44479819c715e15.tar.gz	

diff options

context:	3
space:	include
mode:	unified

KVM: x86/mmu: Fix race condition in direct_page_fault

```
make_mmu_pages_available() must be called with mmu_lock held for write.
However, if the TDP MMU is used, it will be called with mmu_lock held for
read.
This function does nothing unless shadow pages are used, so there is no
race unless nested TDP is used.
Since nested TDP uses shadow pages, old shadow pages may be zapped by this
function even when the TDP MMU is enabled.
Since shadow pages are never allocated by kvm_tdp_mmu_map(), a race
condition can be avoided by not calling make_mmu_pages_available() if the
TDP MMU is currently in use.

I encountered this when repeatedly starting and stopping nested VM.
It can be artificially caused by allocating a large number of nested TDP
SPTEs.

For example, the following BUG and general protection fault are caused in
the host kernel.

pte_list_remove: 00000000cd54fc10 many->many
------------[ cut here ]------------
kernel BUG at arch/x86/kvm/mmu/mmu.c:963!
invalid opcode: 0000 [#1] PREEMPT SMP NOPTI
RIP: 0010:pte_list_remove.cold+0x16/0x48 [kvm]
Call Trace:
 <TASK>
 drop_spte+0xe0/0x180 [kvm]
 mmu_page_zap_pte+0x4f/0x140 [kvm]
 __kvm_mmu_prepare_zap_page+0x62/0x3e0 [kvm]
 kvm_mmu_zap_oldest_mmu_pages+0x7d/0xf0 [kvm]
 direct_page_fault+0x3cb/0x9b0 [kvm]
 kvm_tdp_page_fault+0x2c/0xa0 [kvm]
 kvm_mmu_page_fault+0x207/0x930 [kvm]
 npf_interception+0x47/0xb0 [kvm_amd]
 svm_invoke_exit_handler+0x13c/0x1a0 [kvm_amd]
 svm_handle_exit+0xfc/0x2c0 [kvm_amd]
 kvm_arch_vcpu_ioctl_run+0xa79/0x1780 [kvm]
 kvm_vcpu_ioctl+0x29b/0x6f0 [kvm]
 __x64_sys_ioctl+0x95/0xd0
 do_syscall_64+0x5c/0x90

general protection fault, probably for non-canonical address
0xdead000000000122: 0000 [#1] PREEMPT SMP NOPTI
RIP: 0010:kvm_mmu_commit_zap_page.part.0+0x4b/0xe0 [kvm]
Call Trace:
 <TASK>
 kvm_mmu_zap_oldest_mmu_pages+0xae/0xf0 [kvm]
 direct_page_fault+0x3cb/0x9b0 [kvm]
 kvm_tdp_page_fault+0x2c/0xa0 [kvm]
 kvm_mmu_page_fault+0x207/0x930 [kvm]
 npf_interception+0x47/0xb0 [kvm_amd]

CVE: CVE-2022-45869
Fixes: a2855afc7ee8 ("KVM: x86/mmu: Allow parallel page faults for the TDP MMU")
Signed-off-by: Kazuki Takiguchi <takiguchi.kazuki171@gmail.com>
Cc: stable@vger.kernel.org
Signed-off-by: Paolo Bonzini <pbonzini@redhat.com>
```

Diffstat

-rw-r--r--	arch/x86/kvm/mmu/mmu.c	13

1 files changed, 7 insertions, 6 deletions

```
diff --git a/arch/x86/kvm/mmu/mmu.c b/arch/x86/kvm/mmu/mmu.c
index 1ccb769f62af3..b6f96d47e596d 100644
--- a/arch/x86/kvm/mmu/mmu.c
+++ b/arch/x86/kvm/mmu/mmu.c
@@ -2443,6 +2443,7 @@ static bool __kvm_mmu_prepare_zap_page(struct kvm *kvm,
 {
 	bool list_unstable, zapped_root = false;
 
+	lockdep_assert_held_write(&kvm->mmu_lock);
 	trace_kvm_mmu_prepare_zap_page(sp);
 	++kvm->stat.mmu_shadow_zapped;
 	*nr_zapped = mmu_zap_unsync_children(kvm, sp, invalid_list);
@@ -4262,14 +4263,14 @@ static int direct_page_fault(struct kvm_vcpu *vcpu, struct kvm_page_fault *fault
 	if (is_page_fault_stale(vcpu, fault, mmu_seq))
 		goto out_unlock;
 
-	r = make_mmu_pages_available(vcpu);
-	if (r)
-		goto out_unlock;
-
-	if (is_tdp_mmu_fault)
+	if (is_tdp_mmu_fault) {
 		r = kvm_tdp_mmu_map(vcpu, fault);
-	else
+	} else {
+		r = make_mmu_pages_available(vcpu);
+		if (r)
+			goto out_unlock;
 		r = __direct_map(vcpu, fault);
+	}
 
 out_unlock:
 	if (is_tdp_mmu_fault)
```