☆ Starred by 4 users

Owner:	fwilhelm@google.com **Last visit 15 days ago**
CC:	proje...@google.com
Status:	Fixed *(Closed)*
Components:	----
Modified:	Jun 29, 2021

Deadline-90
Vendor-Linux
CCProjectZeroMembers
Severity-High
Product-Linux
Methodology-source-review
Finder-fwilhelm
Reported-2021-Mar-30
CVE-2021-29657
Fixed-2021-Apr-01

Issue 2177: KVM: double fetch in nested_svm_vmrun can lead to unrestricted MSR access

Reported by fwilhelm@google.com on Tue, Mar 30, 2021, 2:43 PM EDT Project Member

Code 1 of 13 Back to list

KVM: double fetch in nested_svm_vmrun can lead to unrestricted MSR access

Summary: A KVM guest on AMD can launch a L2 guest without the Intercept VMRUN control bit by exploiting a TOCTOU vulnerability in nested_svm_vmrun. Executing vmrun from the L2 guest, will then trigger a second call to nested_svm_vmrun and corrupt svm->nested.hsave with data copied out of the L2 vmcb. For kernel versions that include the commit "2fcf4876: KVM: nSVM: implement on demand allocation of the nested state" (>=5.10), the guest can free the MSR permission bit in svm->nested.msrpm, while it's still in use and gain unrestricted access to host MSRs.

Details:

When an KVM L1 guest on AMD tries to start a L2 guest using the VMRUN instruction, a VM exit is triggered and nested_svm_vmrun (arch/x86/kvm/svm/nested.c) calls nested_vmcb_check_controls to verify that the Intercept VMRUN bit in the L2 VMCB control field is set:

```
static bool nested_vmcb_check_controls(struct vmcb_control_area *control)
{
        if ((vmcb_is_intercept(control, INTERCEPT_VMRUN)) == 0)
                return false;
        ...
        return true;
}
```

However, this check is performed on the guest-controlled VMCB before the structure is copied to the host. Therefore, a malicious guest can bypass the check by repeatedly flipping the intercept VMRUN bit on a different vCPU.

KVM will still set the Intercept VMRUN bit in the real vmcb(02), so when the L2 guest executes another VMRUN instruction it triggers another VM exit.
As KVM is running a nested guest, handle_exit (arch/x86/kvm/svm/svm.c) calls nested_svm_exit_handled, which calls nested_svm_intercept to see if the L1 hypervisor should handle the VM exit.

```
static int nested_svm_intercept(struct vcpu_svm *svm)
{
        u32 exit_code = svm->vmcb->control.exit_code;
        int vmexit = NESTED_EXIT_HOST;

        switch (exit_code) {
        ...
        default: {
                if (vmcb_is_intercept(&svm->nested.ctl, exit_code))
                vmexit = NESTED_EXIT_DONE;
        }
        }

        return vmexit;
}
```

Under normal circumstances, nested_svm_intercept would return NESTED_EXIT for a VMRUN exit and the L1 hypervisor would need to handle it.

However, if the L1 guest exploited the race condition from above svm->nested.ctl won't have the INTERCEPT_VMRUN bit set and the VM exit will be handled by KVM itself.

This results in a second call to nested_svm_vmrun while still running inside the L2 guest context.
nested_svm_vmrun isn't written to handle this situation and will incorrectly overwrite the (L1)
VMCB stored in svm->nested.hsave with data from the L2 VMCB before initializing another nested VM:

```
int nested_svm_vmrun(struct vcpu_svm *svm)
{
        struct vmcb *hsave = svm->nested.hsave;

        ...
         * Save the old vmcb, so we don't need to pick what we save, but can
         * restore everything when a VMEXIT occurs
         */
        hsave->save.es     = vmcb->save.es;
        hsave->save.cs     = vmcb->save.cs;
        hsave->save.ss     = vmcb->save.ss;
        hsave->save.ds     = vmcb->save.ds;
        hsave->save.gdtr   = vmcb->save.gdtr;
        hsave->save.idtr   = vmcb->save.idtr;
        hsave->save.efer   = svm->vcpu.arch.efer;
        hsave->save.cr0    = kvm_read_cr0(&svm->vcpu);
        hsave->save.cr4    = svm->vcpu.arch.cr4;
        hsave->save.rflags = kvm_get_rflags(&svm->vcpu);
        hsave->save.rip    = kvm_rip_read(&svm->vcpu);
        hsave->save.rsp    = vmcb->save.rsp;
        hsave->save.rax    = vmcb->save.rax;
        if (npt_enabled)
                hsave->save.cr3    = vmcb->save.cr3;
        else
                hsave->save.cr3    = kvm_read_cr3(&svm->vcpu);

        copy_vmcb_control_area(&hsave->control, &vmcb->control);
        …
}
```

This becomes a security issue due to the way
MSR intercepts are handled for nested guests: The MSR permission bitmap for a nested guest is stored in svm->nested.msrpm and its physical address is stored in the vmcb->control.msrpm_base_pa field.Using the described double invocation of nested_svm_vmrun, a malicious guest can copy this value into the svm->nested.hsave VMCB.

Since commit "2fcf4876: KVM: nSVM: implement on demand allocation of the nested state", svm->nested.msrpm is dynamically allocated and freed when a guest changes the SVME bit of the MSR_EFER register:

```
int svm_set_efer(struct kvm_vcpu *vcpu, u64 efer)
{
        struct vcpu_svm *svm = to_svm(vcpu);
        u64 old_efer = vcpu->arch.efer;
        vcpu->arch.efer = efer;

        if ((old_efer & EFER_SVME) != (efer & EFER_SVME)) {
                if (!(efer & EFER_SVME)) {
                        svm_leave_nested(svm); (A)
                        svm_set_gif(svm, true);
                        ...                     /*
                         * Free the nested guest state, unless we are in SMM.
                         * In this case we will return to the nested guest
                         * as soon as we leave SMM.
                         */
                        if (!is_smm(&svm->vcpu))
                                svm_free_nested(svm); (B)

                } ...
}
}
```

For the "disable SVME" case, KVM will first call svm_leave_nested to forcibly leave potential nested guests and then free the svm->nested data structures (including the backing pages for the MSR permission bitmap) in svm_free_nested.
As svm_leave_nested believes that svm->nested.hsave points to the saved context of the L1 guest, it simply copies its control area to the real VMCB:

```
void svm_leave_nested(struct vcpu_svm *svm)
{
        if (is_guest_mode(&svm->vcpu)) {
                struct vmcb *hsave = svm->nested.hsave;
                struct vmcb *vmcb = svm->vmcb;

                svm->nested.nested_run_pending = 0;
                leave_guest_mode(&svm->vcpu);
                copy_vmcb_control_area(&vmcb->control, &hsave->control);
                nested_svm_uninit_mmu_context(&svm->vcpu);
                vmcb_mark_all_dirty(svm->vmcb);
        }
}
```

As we mentioned before, svm->nested.hsave->control.msrpm_base_pa can still point to svm->nested->msrpm. This means that once svm_free_nested is finished and KVM passes control back to the guest, the CPU will use a freed page for its MSR permission checks.
This gives a guest unrestricted access to host MSRs if the page is reused and partially overwritten

with zeros.

I believe this can be turned into a full guest-to-host escape, for example by overwriting MSR_VM_HSAVE_PA with an attacker controlled value.

Proof-of-Concept:

The attached proof-of-concept is a patch to the kvm-unit-tests project. You should see output similar to the following when executed on a vulnerable system, demonstrating read access to the MSR_VM_HSAVE_PA host MSR.

```
felix@host:~/kvm-unit-tests$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-5.11.10-051110-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro init_on_free=1
felix@host:~/kvm-unit-tests$ make; sudo ./x86-run ./x86/svm_escape.flat -smp 2
/usr/bin/qemu-system-x86_64 --no-reboot -nodefaults -device pc-testdev -device
isa-debug-exit,iobase=0xf4,iosize=0x4 -vnc none -serial stdio -device pci-testdev -machine
accel=kvm -kernel ./x86/svm_escape.flat -smp 2 # -initrd /tmp/tmp.9Wu6jRj9Z7
enabling apic
enabling apic
paging enabled
cr0 = 80010011
cr3 = 1007000
cr4 = 20
cpu id 0
cpu count 2
l2 is executing
l3 is executing
[l3] triggering svm_free_nested
[l3] real msr access
[l3] hsave host: 10556b000 hsave virtual: 100d000
```

Uncommenting the line wrmsr(MSR_VM_HSAVE_PA, 0x1234000) line will lead to a host crash. Please note that I’m enabling the init_on_free=1 configuration option in the host kernel, to make the issue easier to reproduce. A real exploit would need to wait for a reallocation of the physical page backing the MSR permission map.

Impact and Patch:

I am not sure about the impact of this issue before commit “2fcf4876: KVM: nSVM: implement on demand allocation of the nested state”. We still end up with a corrupted svm->nested.hsave VMCB and a weird guest-mode state, but I haven’t found a way to turn this into a security issue.
Maybe this could be used to trigger some MMU related issues?
In the best case scenario, this is only a security issue for versions >=v5.10 which include the on demand allocation as this would mean most KVM users are not affected. It would be great if someone can double check this.

I think the correct way to address this issue is to move away from running nested_vmcb_checks on the guest VMCB. While it’s easy to fix this specific issue by adding an additional check that verifies the interrupt in svm->nested.ctl, nested_vmcb_checks just seems to be an error-prone approach and might lead to other issues.
I don’t see a reason why these checks can’t be done later in the function once the fields are copied to host memory, but I might be missing something.

This bug is subject to a 90 day disclosure deadline. After 90 days elapse, the bug report will become visible to the public. The scheduled disclosure date is 2021-06-28. Disclosure at an earlier date is also possible if agreed upon by all parties.

svm_escape.patch
6.8 KB View Download

Comment 1 by fwilhelm@google.com on Wed, Mar 31, 2021, 1:35 PM EDT Project Member
Labels: Type-Enhancement CVE-2021-29657

Comment 2 by fwilhelm@google.com on Wed, Mar 31, 2021, 3:36 PM EDT Project Member
Labels: -Type-Enhancement

Comment 3 by fwilhelm@google.com on Tue, Apr 6, 2021, 8:38 AM EDT Project Member
Status: Fixed (was: New)

Fixed in https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=a58d9166a756a0f4a6618e4f593232593d6df134

Comment 4 by fwilhelm@google.com on Tue, Apr 6, 2021, 8:39 AM EDT Project Member
Labels: Fixed-2021-Apr-01

Comment 5 by fwilhelm@google.com on Mon, May 31, 2021, 4:43 AM EDT Project Member
The attachment contains a proof-of-concept exploit for this bug demonstrating a full guest-to-host privilege escalation against a vulnerable v5.11 KVM host. .

```
[!] exit reason: 0xffffffff. let's try again
[x] l2 is executing
l3 is executing
[x] triggering svm_free_nested
[x] real msr access
[x] hsave host: 854c23000 hsave virtual: bfec5000
[x] leak_host_access: ffffffffc05dc772 851db9d00 ffff9f45d35efd00
[x] leak_host_access: ffffffffc05d32c5 8589982e8 ffff906dd89982e8
[x] leak_host_access: success!
[x] module_base: ffffffffc05cd000 phys_base: ffff906580000000
[x] leak_host_hpa: 10000000000000 ffffffffc05dc766 8559e6489
[x] kvm-amd.ko target hpa: 8559e6000
[x] leak_guest_hpa: 400526 86a8c607b bfec607b
[x] leak_guest_hpa: 40051f 86a8c704f bfec704f
[x] stack hpa: 86a8c7000 fake hsave hpa: 86a8c6000
[x] rip: ffffffffc05dc74b rsp: ffff906dea8c7780
[?] offset: aa
[x] address of final payload: ffffffffc05dc8ec
[.] please stand by and wait for your shell (eta 5 min)
```

svm_escape.tar.gz
132 KB Download

Comment 6 by rschoen@google.com on Tue, Jun 29, 2021, 12:28 PM EDT Project Member

Labels: -Restrict-View-Commit

About Monorail User Guide Release Notes Feedback on Monorail Terms Privacy