Project summary

Goal:
Support SVM in HVM guests.

Description:
Allow to run HVM guests within HVM guests.

Aimed for AMD product:
any

Person in charge:
Christoph Egger

Project Status:
Done.

Project Results:
HVM Guest within an HVM guest is runnable.

Contents

General Design and Concept

Nested SVM is implemented very generic. Only 10% of the code is really SVM specific. That means, Intel can implement Nested SVM by implementing a few functions hooked into VMX. The nested guest, however, believes to use SVM.

This document describes the xen specific implementation. It is assumed that the reader knows the general design and concepts behind Nested Virtualization.

Getting Started

Setup a guest which will run a hypervisor in itself. In the guest configuration file add the line

```
nestedhvm = 1
```

This enables and makes the SVM cpuid feature bits visible to the guest. Start the guest.

Software architecture

Implementation overview

By design, the guest always uses SVM. This allows to implement 90% of the nested virtualization feature in a generic, clean and portable fashion.
The central file for this feature is xen/arch/x86/hvm/nestedhvm.c.

The generic code may only touch the guest vmcb. The SVM/VMX specific implementations touch the host vmcb/vmcs. SVM and VMX specific code must implement the following function pointer hooks:

- nestedhvm_vcpu_initialise
- nestedhvm_vcpu_destroy
- nestedhvm_vcpu_features
- nestedhvm_vcpu_hostsave
- nestedhvm_vcpu_hostrestore
- nestedhvm_vcpu_vmsave
- nestedhvm_vcpu_vmload
- nestedhvm_vcpu_vmrun
- nestedhvm_vcpu_prepare4vmrun
- nestedhvm_vcpu_prepare4vmexit

What the function hooks do is explained in the following sections.

AMD support

All function hooks are implemented in svm specific code under xen/arch/x86/hvm/svm/svm.c.

Intel support

On the Intel side, all function hooks must be implemented in vmx specific code under xen/arch/x86/hvm/vmx/*. The main effort is to implement a vmcb <-> vmcs conversion function in order to implement prepare4vmrun / prepare4vmexit. VMLOAD/VMSAVE instruction emulation does not require a full conversion. Saving/Loading the hoststate requires no conversion at all.

Getting nested Xen booting

Xen and the Dom0 use a number of MSRs such as EFER and NB_CFG for feature detection/enablement, cpu topology detection, etc.

The patches c/s 20058 in Xen-unstable and c/s 19729 in Xen-3.4-testing emulate the NB_CFG MSR. It is needed to make the Dom0 boot in nested Xen.

The HVM BIOS provides some ACPI tables but no ACPI registers via MMIO. Therefore, nested Xen and its nested Dom0 use in/out port instructions to access ACPI registers.

Enable SVM

In order to be able to launch a HVM guest in nested Xen you have to enable the SVM cpuid feature bit. The cpuid feature bits are set by the xen tools in tools/libxc/xc_cpuid_x86.c.

The guest must be able to query cpuid functions 0x80000000, 0x80000001 and 0x8000000a. This is handled in the xen kernel in xen/arch/x86/hvm/hvm.c function hvm_cpuid().

In cpuid function 0x80000001 we mask out the SVM feature bit when nested virtualization is disabled in the guest config file. Currently we also mask out the SVM feature bit when the host does not use nested paging until shadow-on-shadow is implemented.

In cpuid function 0x8000000a we don't report any svm feature bits when nested virtualization is disabled in the guest config file

The nestedhvm_vcpu_features hook exposes the SVM feature bits (see cpuid 0x8000000a) to the guest which are implemented and the guest can use. hvm_cpuid() calls it when guest queries cpuid function 0x8000000a.

Currently SVM specific implementation report the nested paging feature if the host uses nested paging, the lbrv, nextrip and pausefilter.

Xen disallowed to set the SVME bit in the EFER MSR, now we need to allow it if nested SVM is enabled and if SVM cpuid feature bit is set or nested Xen dies at boot with a #GP injected by Xen host, otherwise. So we allow

to set the SVME bit in hvm_set_efer() and in hvm_load_cpu_ctxt() in xen/arch/x86/hvm/hvm.c. Latter one is used on guest restore from saved state.

Nested VMCB

Allocate a VMCB used for the vcpu to enter/exit its guest state. In the following this is called 'nested VMRUN', 'nested VMEXIT' and 'nested VMCB'. The guest sets up its own VMCB and runs the VMRUN, VMLOAD, VMSAVE, STGI and CLGI instructions.

In xen/include/asm-x86/hvm/vcpu.h is a struct nestedhvm holding per-virtual cpu data. The structure contains data for emulating the vcpu's host/guest-mode, the gif, a vmcb, hostsave state area, caches the guest intercepts, io and msr permission bitmaps and some flags.

The data structure is initialized in nestedhvm_vcpu_initialise() called by hvm_vcpu_initialise(). The data structure is freed in nestedhvm_vcpu_destroy() called by hvm_vcpu_destroy().

nestedhvm_vcpu_initialise() and nestedhvm_vcpu_destroy() call the nestedhvm_vcpu_initialise and nestedhvm_vcpu_destroy function hooks. In SVM, they initialize/free the MSR and IO permisson bitmap.

What needs to be emulated

MSRs Xen and Dom0 use, the instructions VMRUN, VMLOAD, VMSAVE, CLGI, STGI and the #VMEXIT

Instruction emulation

On any instruction emulation we validate the guest vmcb at first. The validation is done by nestedhvm_vcpu_state_validate() in xen/arch/x86/hvm/nestedhvm.c.

- If nestedhvm is disabled inject #UD.
- If cpu is in real mode then inject #UD.
- If cpl != 0 then inject #GP.
- If vmcb address is not canonical or not 4kb page aligned then inject #GP.

CLGI

The generic part is implemented in xen/arch/x86/hvm/nestedhvm.c, nestedhvm_vcpu_clgi(). It disables the gif flags and masks events for PV drivers. The svm specific part is implemented in xen/arch/x86/hvm/svm/svm.c, nsvm_vmexit_do_clgi(). It decodes the instruction length, calls nestedhvm_vcpu_clgi() and updates the rip.

STGI

Generic part is implemented in xen/arch/x86/hvm/nestedhvm.c, nestedhvm_vcpu_stgi(). Enables the gif flag and unmasks events for PV drivers. The svm specific part is implemented in xen/arch/x86/hvm/svm/svm.c, nsvm_vmexit_do_stgi(). It decodes the instruction length, calls nestedhvm_vcpu_stgi() and updates the rip.

VMLOAD

Generic code is in xen/arch/x86/hvm/nestedhvm.c, nestedhvm_vcpu_vmload(). It calls the nestedhvm_vcpu_vmload function hook.

The SVM implementation is nsvm_vcpu_vmload(). It maps guest vmcb into host and copies the vmcb fields fs, gs, tr, ldtr, kerngsbase, star, lstar, cstar, sfmask, sysenter_cs, sysenter_esp and sysenter_eip into the host vmcb. Then run the real vmload instruction.

VMSAVE

Generic code is in xen/arch/x86/hvm/nestedhvm.c, nestedhvm_vcpu_vmsave(). It calls the nestedhvm_vcpu_vmsave function hook.

The SVM implementation is nsvm_vcpu_vmsave(). It maps guest vmcb into host, runs the real vmsave instruction and copies the vmcb fields fs, gs, tr, ldtr, kerngsbase, star, lstar, cstar, sfmask, sysenter_cs, sysenter_esp and sysenter_eip from the host vmcb into the guest vmcb.

VMRUN

The svm specific implemention is in xen/arch/x86/hvm/svm/svm.c, svm_vmexit_do_vmrun(). It decodes the instruction length and calls the generic nestedhvm_vcpu_vmrun().

The generic nestedhvm_vcpu_vmrun() is implemented in xen/arch/x86/hvm/nestedhvm.c. It implements the algorithm as described in section VMRUN.

It calls some function pointer hooks:

nestedhvm_vcpu_hostsave saves the hoststate as described in section Saving the hoststate.
nestedhvm_vmcb_prepare4vmrun loads the guest vmcb as described in section Loading VMCB N+1.
nestedhvm_vcpu_vmrun called with a different flag. The purpose is to have an opportunity to do some SVM/VMX specific tweaks, e.g. optimizations or to make VMRUN work in general.

Interrupt handling

Beyond of STGI and CLGI emulation, the HVM interrupt and exception subsystem has to be teached to not inject anything.

A new type of interrupt blocking state has been introduced: hvm_intblk_gif to indicate whether something may be injected or not. The function hvm_interrupt_blocked() has been modified to return hvm_intblk_gif when the gif flag is cleared. The loop in svm_intr_assist() has been modified to abort the loop w/o enabling an interrupt window. "Enabling an interrupt window" means to inject a dummy virtual interrupt to intercept as soon as the guest can accept the real interrupt.

In this respect, Xen has a wishitem to improve in the hardware:

```
/*
 * TODO: Better NMI handling. We need a way to skip a MOV SS interrupt
 * shadow. This is hard to do without hardware support. We should also
 * track 'NMI blocking' from NMI injection until IRET. This can be done
 * quite easily in software by intercepting the unblocking IRET.
 */
```

If the host guest does not intercept interrupts then they will be injected into the nested guest directly. If the interrupts are masked then inject a vintr to get an interrupt window. If the GIF is cleared then do nothing.

The svm_intr_assist runs right before the VMRUN instruction. In this stage it is too late to inject an

1. VMEXIT into the guest. So we enable a forced VMEXIT which basically skips the VMRUN instruction and invoke

the vmexit handler directly with the forced exitcode. The decision if and which interrupt type (e.g. INTR, NMI) should be injected is made by the generic nestedhvm_vcpu_interrupt() in xen/arch/x86/hvm/nestedhvm.c.

nested VMEXIT

The generic implementation is in xen/arch/x86/hvm/nestedhvm.c, nestedhvm_vcpu_vmexit(). It is called by the svm specific vmexit handler when virtual cpu is in guest mode.

nestedhvm_vcpu_vmexit() implements the algorithm as described in section VMEXIT. It is implemented to keep the algorithm very straightforward.

nestedhvm_isintercepted_by_guest() checks if guest intercepts the occured intercept.
nestedhvm_vmexit_intercepts() handles special intercepts by deciding which one needs to be handled by the guest or the host. See section Special intercepts.

nestedhvm_vcpu_vmexit() calls some function pointer hooks:

nestedhvm_vmcb_prepare4vmexit implements the algorithm desccribed in section Prepare VMEXIT.
nestedhvm_vcpu_hostrestore restores the host state as described in section restore the hoststate.
nestedhvm_vcpu_vmrun called with a different flag. The purpose is to have an opportunity to do some SVM/VMX specific tweaks.

Page maintainer: Christoph Egger