virtual secure boot

Secure boot support in qemu, kvm and ovmf.

Gerd Hoffmann <kraxel@redhat.com>

33C3, Hamburg

Outline

- Terms
- Create a plan
- Implementation in ...
  - qemu
  - kvm
  - ovmf
- Hands on
- Demo
- Q&A

Terms

secure boot

- Specified in UEFI 2.3.1
- Goal: make sure no unsigned (kernel) code runs on the machine.
- Firmware-verified chain of trust.
- Each component verifies the next before running it.
  - Firmware checks bootloader.
  - Bootloader checks kernel.
  - Kernel checks modules.
- Keys are managed by the firmware.

➜ Firmware and key storage must be protected.

qemu - quick emulator

- Open source machine emulator and virtualizer.
- Emulates everything you have in a physical machine.
  - Chipset (x86: i440fx, q35)
  - Timer, interrupt controller
  - Display, keyboard, mouse, tablet
  - ide, sata, scsi, disks, cdrom
  - network, usb, ...
  - http://qemu-project.org/

kvm - kernel virtual machine

- Linux kernel module, handles cpu virtualization.
  - x86: vmx, svm extensions.
  - arm: hyp mode.
- Qemu uses kvm for cpu virtualization.

tcg - tiny code generator

- Part of qemu
- Transform target (guest) instructions into host instructions.
- Qemu uses tcg for cpu emulation.
- Use cases:
  - Foreign architecture, i.e. arm guest on x86.
  - CPUs without virtualization support (no kvm).
  - When kvm ist to fast for old guests.

edk2 - EFI Development Kit II

- Intel's UEFI implementation.
- http://www.tianocore.org/
- https://github.com/tianocore/edk2

ovmf - Open Virtual Machine Firmware

- x86 UEFI Implementation for qemu.
- Lives in `OvmfPkg/` subdir of edk2 repo.
- Comes with drivers for virtio and qemu vga devices.
- FYI: qemu arm/aarch64 firmware is in the edk2 repo too (`ArmVirtPkg/`).

seabios

- Default firmware in qemu.
- Classic BIOS implementation, for qemu and coreboot.
- https://www.seabios.org/.
- FYI: coreboot has qemu support too.

Two years ago

2014: secure boot support in ovmf

- Uses openssl for crypto.
  - Untar openssl tarball into subdir.
  - Apply patch to adapt to efi environment (no stdio).
  - Build with `-D SECURE_BOOT_ENABLE` option.

✔ All secure boot firmware interfaces are there and working.

✗ No firmware RAM protection (code + data).

✗ No flash protection (persistent efi vars and keys).

➜ Malicious guests can tamper as they like.

- Still useful for development and testing.

Designing qemu support

- Stay close to bare metal (emulate hardware)?
- Create something new?

Emulate existing hardware

+ JustWorks™, guest OS comes with drivers.
+ Keeps differences between physical and virtual hardware small.
  - Tends to simplify management.
− Can be quite expensive to emulate (uhci).
− Limited to physical hardware features (cirrus).

Create paravirtual hardware

+ More freedom in designing things.
+ Better performance.
+ Code sharing.
  - All virtio devices and drivers use the same ring format.
+ Can simplify things.
− Must write and maintain guest drivers.
− Simplifying too much can hurt long-term though.
  - Initial virtio spec going "native endian" turned out to be a bad idea (ppc64le).

Firmware protection on physical hardware

- Uses SMM (system management mode).
  - Lets chipset interrupt OS by asserting SMI.
    - OUT to port 0xb2 usually triggers an SMI.
  - Processor state is stored in RAM, execution starts at SMBASE+0x8000.
  - The RSM instruction resumes normal execution.
  - Designed for power management and hardware control.
  - Some RAM can only be accessed in SMM mode.
    - SMRAM (128K, at 0xa0000).
    - TSEG (up to 8M, top of RAM below 4G).
- SMRAM and TSEG configuration can be locked.

Firmware protection on qemu

- Stay close to bare metal and use SMM too?
  - + Code sharing, can reuse the edk2 SmmLockBox implementation.
  - + Not in hot path, so performance shouldn't be a big issue.
  - − SMM was not designed with security in mind and has a bad track record.
    - Check out congress talks from previous years.
    - Must be careful when implementing this.
  - − kvm had no SMM support.
  - tcg had basic SMM support.

Implementing SMM in qemu

qemu: our todo list

- i440fx chipset (-M pc) has no TSEG, SMRAM is too small
  ➔ not supported (with SMM) by edk2.
- Fix/complete q35 chipset emulation (-M q35).
  - Complete/improve SMRAM support.
  - Implement TSEG support.
  - Implement lockbit support.
- Flash protection.

qemu: the memory api

- Created by Avi Kivity (2011), merged in qemu 1.0.
- Puts qemu memory management upside down.
- Old: register-that-at-this-address (destructive).
- New: hierarchy of memory regions.
  - Regions can be enabled, disabled, moved, aliased.
  - Regions are used to build address spaces.
- Big step forward in emulation correctness.
  - PAM register emulation (0xa0000 -> 0xfffff).
  - PCI bar remapping.
  - PCI busmaster config bit.
- Implementing SMM would have been impossible without this.
- http://git.qemu.org/?p=qemu.git;a=blob;f=docs/memory.txt

qemu: "info mtree" snippet (-M isapc)

```
memory-region: system
  00000000-ffffffff (prio 0, RW): system
    00000000-07ffffff (prio 0, RW): alias ram-below-4g @pc.ram 00000000-
07ffffff
    000a0000-000bffff (prio 1, RW): cirrus-lowmem-container
      000a0000-000a7fff (prio 1, RW): alias vga.bank0 @vga.vram 00000000-
00007fff [disabled]
      000a0000-000bffff (prio 0, RW): cirrus-low-memory
      000a8000-000affff (prio 1, RW): alias vga.bank1 @vga.vram 00008000-
0000ffff [disabled]
    000c0000-000dffff (prio 1, RW): pc.rom
    000e0000-000fffff (prio 1, RW): alias isa-bios @pc.bios 00000000-0001ffff
    fffe0000-ffffffff (prio 0, RW): pc.bios
```

qemu: smram/tseg system memory regions (-M q35)

```
memory-region: system
  00000000-ffffffff (prio 0, RW): system
    00000000-7fffffff (prio 0, RW): alias ram-below-4g @pc.ram 00000000-
7fffffff
    00000000-ffffffff (prio -1, RW): pci
      000a0000-000fffff [ ... real mode mappings ... ]
      90000000-fff00000 [ ... pci bars ... ]
    000a0000-000bffff (prio 1, RW): alias smram-region @pci 000a0000-000bffff
    000c0000-000fffff [ ... lots of pam regions ... ]
    7f800000-7fffffff (prio 1, RW): tseg-blackhole
    80000000-8fffffff (prio 0, RW): pcie-mmcfg-mmio
    fec00000-fec00fff (prio 0, RW): ioapic
    feda0000-fedbffff (prio 1, RW): alias smram-open-high @pc.ram 000a0000-
000bffff [disabled]
    fee00000-feefffff (prio 4096, RW): apic-msi
    ffe00000-ffe1ffff (prio 0, R-): system.flash1
    ffe20000-ffffffff (prio 0, R-): system.flash0
```

- old qemu versions just enable/disable smram-region (tcg only).

qemu: smram/tseg memory regions (tcg)

```
memory-region: smram
  00000000-ffffffff (prio 0, RW): smram
    000a0000-000bffff (prio 0, RW): alias smram-low @pc.ram 000a0000-000bffff
    7f800000-7fffffff (prio 0, RW): alias tseg-window @pc.ram 7f800000-7fffffff
    feda0000-fedbffff (prio 0, RW): alias smram-high @pc.ram 000a0000-000bffff
[disabled]
address-space: CPU
  00000000-ffffffff (prio 0, RW): memory
    00000000-ffffffff (prio 1, RW): alias smram @smram 00000000-ffffffff
[disabled]
    00000000-ffffffff (prio 0, RW): alias memory @system 00000000-ffffffff
address-space: nec-usb-xhci
  00000000-ffffffff (prio 0, RW): alias bus master @system 00000000-ffffffff
```

- smram alias is enabled when the cpu is in SMM mode.
- each cpu has its own address space.
  - Avoids SMP races.
- pci devices use "system" region for busmaster dma.
  - Same view vcpus have when not in SMM mode.

qemu: smram lockbit (from tests/q35-test.c)

```
static void test_smram_lock(void)
{
    /* check open is settable */
    smram_set_bit(pcidev, MCH_HOST_BRIDGE_SMRAM_D_OPEN, false);
    g_assert(smram_test_bit(pcidev, MCH_HOST_BRIDGE_SMRAM_D_OPEN) == false);
    smram_set_bit(pcidev, MCH_HOST_BRIDGE_SMRAM_D_OPEN, true);
    g_assert(smram_test_bit(pcidev, MCH_HOST_BRIDGE_SMRAM_D_OPEN) == true);

    /* lock, check open is cleared & not settable */
    smram_set_bit(pcidev, MCH_HOST_BRIDGE_SMRAM_D_LCK, true);
    g_assert(smram_test_bit(pcidev, MCH_HOST_BRIDGE_SMRAM_D_OPEN) == false);
    smram_set_bit(pcidev, MCH_HOST_BRIDGE_SMRAM_D_OPEN, true);
    g_assert(smram_test_bit(pcidev, MCH_HOST_BRIDGE_SMRAM_D_OPEN) == false);
}
```

- SMRAM LCK bit locks down TSEG config too.

q35 tseg hardware bug

- SMRAM LCK does not lock down TOLUD.
  - Top of Low Usable DRAM -- the below / above 4G memory split.
  - TSEG is at the end of low memory, i.e. location is relative to TOLUD.
  - Attacker can't turn off TSEG, but move it out of the way.
- qemu simply doesn't implement the TOLUD register.
  - Split is configurable with command line switches instead.
  - q35 has 2G low memory by default.

Flash protection on real q35 hardware

- Memory mapped NOR flash.
- Writes to flash put it into "device mode".
- Writes also triggers an SMI, the SMI handler puts the flash back into ROM mode.
- Complicated and prone to races.
- Newer chipsets use even more complicated protocols to work around the races.

Flash on qemu

- ovmf is splitted into two parts.
- OVMF_CODE.fd holds the code.
  - typically read-only for the guest.
  - updates happen on the host (normal distro package).
- OVMF_VARS.fd holds the variable store.
  - In secure mode writes are simply discarded outside SMM.
  - Implemented using memory transaction attributes.
  - Secure flag (see include/exec/memattrs.h) signals access from SMM mode (x86).
  - Concept is borrowed from arm, where the secure flag is a bus signal.

Implementing SMM in kvm

Memory regions in kvm mode

- Qemu flattens address spaces into a memory map (list of slots).
- The memory map is passed to kvm, shared by all vcpus.
- On each address space change the kernel's memory map must be updated.
  - To make sure qemu and kvm have the same view.
  - Per-cpu address spaces (tcg approach) are expensive.

kvm: adding SMM support

- Add address space IDs for memory maps.
- For SMM support two memory maps (SMM and non-SMM mode) are used.
  - Shared by all vcpus.
- New ioctl: KVM_SMI.
- Add SMM flag to `struct kvm_run`.
  - so qemu can figure vcpu SMM state on vmexit.
  - needed for memory transaction attributes (flash protection).

qemu: smram/tseg memory regions (kvm)

```
address-space: cpu-memory
  00000000-ffffffff (prio 0, RW): system
    00000000-7fffffff (prio 0, RW): alias ram-below-4g @pc.ram 00000000-
7fffffff
    [ ... ]

address-space: KVM-SMRAM
  00000000-ffffffff (prio 0, RW): mem-container-smram
    00000000-ffffffff (prio 10, RW): smram
      000a0000-000bffff (prio 0, RW): alias smram-low @pc.ram 000a0000-000bffff
      7f800000-7fffffff (prio 0, RW): alias tseg-window @pc.ram 7f800000-
7fffffff
      feda0000-fedbffff (prio 0, RW): alias smram-high @pc.ram 000a0000-
000bffff [disabled]
    00000000-ffffffff (prio 0, RW): alias mem-smram @system 00000000-ffffffff
```

- non-SMM memory map is build from cpu-memory address space.
- SMM memory map is build from KVM-SMRAM address space.

SMM support for OVMF

Just wire up SMM for OVMF ...

- We emulate physical hardware, so it should be easy, right?

✔ SMM Lockbox code is in the edk2 repo.

✗ But SMM Initialization is not.

☹ Oops.

Dig up the smm init source code

- Asked the edk2 intel guys.
- Got pointer to Quark_EDKII_v1.1.0 tarball (April 2015)
  - IA32 only.
  - Found security bug in review.
- lots of discussions with intel ...
  - ... edk2 maintainers.
  - ... quark team.
  - ... security team.

Merge the smm init source code

- Intel committed the code to edk2/UefiCpuPkg/ (Oct 2015).
  - Half a year later, finally.
  - Both IA32 and X64.
  - With security bug fixed.
  - We are not aware of any official security advisory for quark though.
- Ongoing maintainance & open source style development.

Wire up SMM for OVMF

- Now it actually is as easy as we expected initially.
- Still not trivial though, lots of details to get right.
- OVMF code reviewed and merged in Nov 2015.

Hands on

Minimum versions

- qemu: 2.5 (Dec 2015)
- linux: 4.4 (Jan 2016)
- libvirt: 2.1.0 (Aug 2016)
- edk2: latest (RHEL-7 uses 20160608 git snapshot)

libvirt domain config (RHEL-7.3 host)

```
<domain type='kvm'>
  <name>secboot-rhel7-kvm</name>
  [ ... ]
  <os>
    <type arch='x86_64' machine='pc-q35-rhel7.3.0'>hvm</type>
    <loader readonly='yes' secure='yes'
type='pflash'>/usr/share/OVMF/OVMF_CODE.secboot.fd</loader>
    <nvram template='/usr/share/OVMF/OVMF_VARS.fd'>/.../secboot-rhel7-
kvm_VARS.fd</nvram>
  </os>
  <features>
    [ ... ]
    <smm state='on'/>
  </features>
  [ ... ]
```

Qemu command line

```
/usr/libexec/qemu-kvm \
    -machine q35,accel=kvm,smm=on \
    -drive
file=.../OVMF_CODE.secboot.fd,if=pflash,format=raw,unit=0,readonly=on \
    -drive file=.../secboot-rhel7-kvm_VARS.fd,if=pflash,format=raw,unit=1 \
    -global driver=cfi.pflash01,property=secure,value=on \
    ${moreargs}
```

libvirt domain config for older versions

- libvirt allows to add args to the qemu command line.

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  [ ... ]
  </devices>
  <qemu:commandline>
    <qemu:arg value='-machine'/>
    <qemu:arg value='smm=on'/>
    <qemu:arg value='-global'/>
    <qemu:arg value='driver=cfi.pflash01,property=secure,value=on'/>
  </qemu:commandline>
</domain>
```

enroll keys

- OVMF doesn't ship with any keys installed.
- Setup procedure:
  - Boot from /usr/share/OVMF/UefiShell.iso
  - Drops you into a efi shell.
  - Run EnrollDefaultKeys efi application.

```
# dmesg | grep "EFI.*cert"
EFI: Loaded cert 'Microsoft Windows Production PCA 2011: [ ... ]
EFI: Loaded cert 'Microsoft Corporation UEFI CA 2011: [ ... ]
EFI: Loaded cert 'Red Hat Secure Boot (CA key 1): [ ... ]
```

cutting edge firmware builds

- repo @ https://www.kraxel.org/repos/jenkins/
- `yum install -y edk2.git-ovmf-x64`
- `ls -l /usr/share/edk2.git/ovmf-x64`
- three different variants:
  - `need-smm` — requires q35 with smm enabled, supports secure boot.
  - `pure-efi` — supports both q35 and i440fx, no secure boot.
  - `with-csm` — built with seabios as CSM for BIOS compatibility.
- FYI: repo has edk2 ArmVirtPkg, seabios, coreboot, ipxe builds too.

Demo

That's it

- Slides
  - https://www.kraxel.org/slides/virtual-secure-boot/
- Credits
  - László Érsek (ovmf).
  - Paolo Bonzini (kvm, tcg).
  - Gerd Hoffmann (q35).
  - Many patch reviewers.