Secure CPU configuration for KVM-based guests

Kashyap Chamarthy <kchamart@redhat.com>

Red Hat Tech Day

Brussels; 24 Jan 2020

Timeline of recent CPU flaws, 2018 (a)

Timeline of recent CPU flaws, 2018 (b)

Jun 29	**NetSpectre**
Jul 10	**Spectre-NG**
Aug 14	**L1TF (Foreshadow)**
Nov 01	**PortSmash**

Timeline of recent CPU flaws, 2019 (a)

May 14 — **ZombieLoad**

May 14 — **RIDL**

May 14 — **Fallout - TAA**

May 14 — **Microarchitectural Data Sampling**

Timeline of recent CPU flaws, 2019 (b)

Oct 30	**Another variant of L1TF**
Nov 12	**New variants of TAA & RDIL**
...	?

What this talk is not about

What this talk is not about

Out of scope:

- Internals of various side-channel attacks
- Exploitation techniques
- Detailed performance analysis

What this talk is not about

Out of scope:

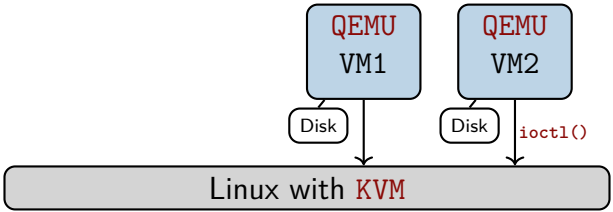
- Internals of various side-channel attacks
- Exploitation techniques
- Detailed performance analysis

$\rightsquigarrow$ **Related talks in the 'References' section**

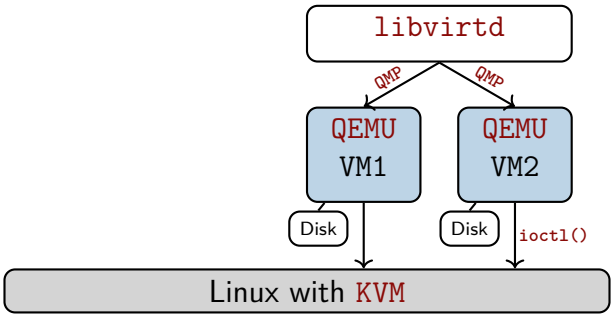
KVM-based virtualization components

Linux with KVM

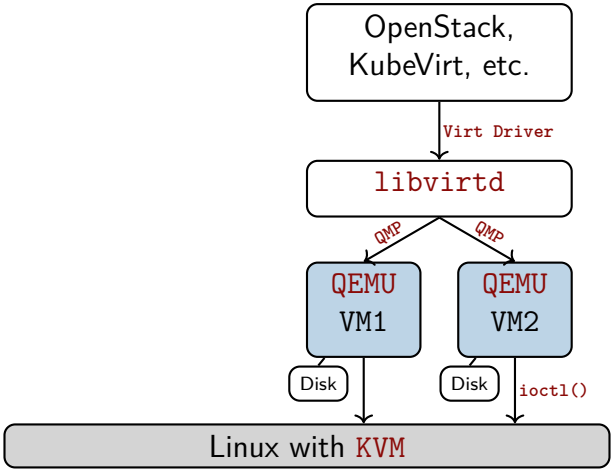
KVM-based virtualization components



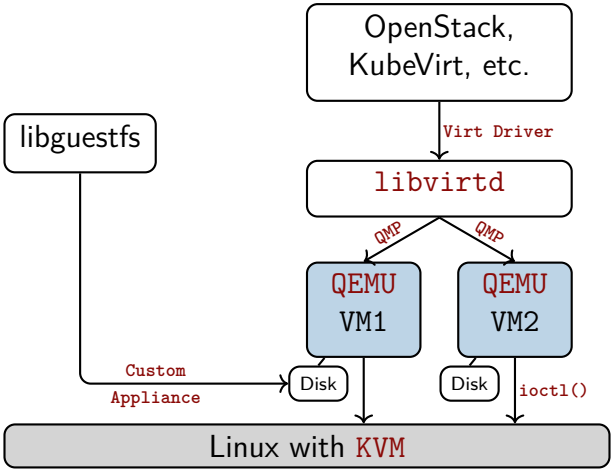
KVM-based virtualization components



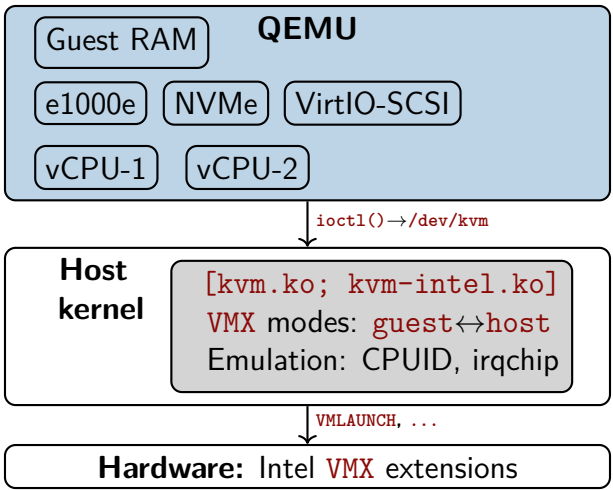
KVM-based virtualization components



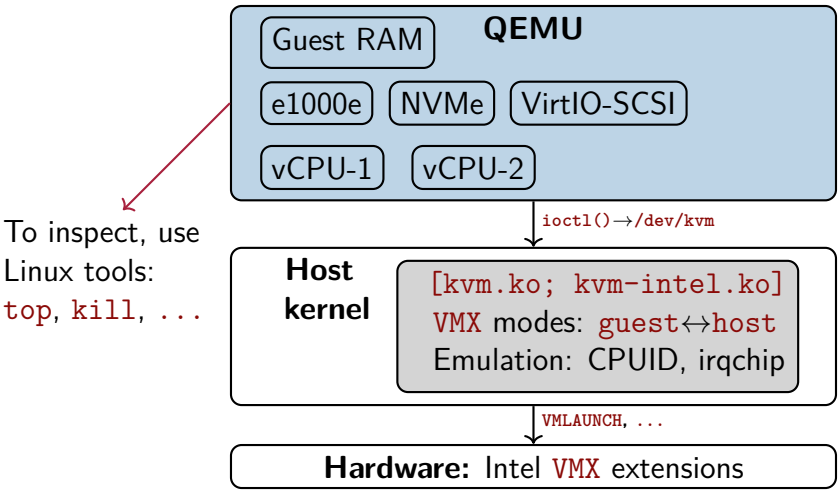
KVM-based virtualization components



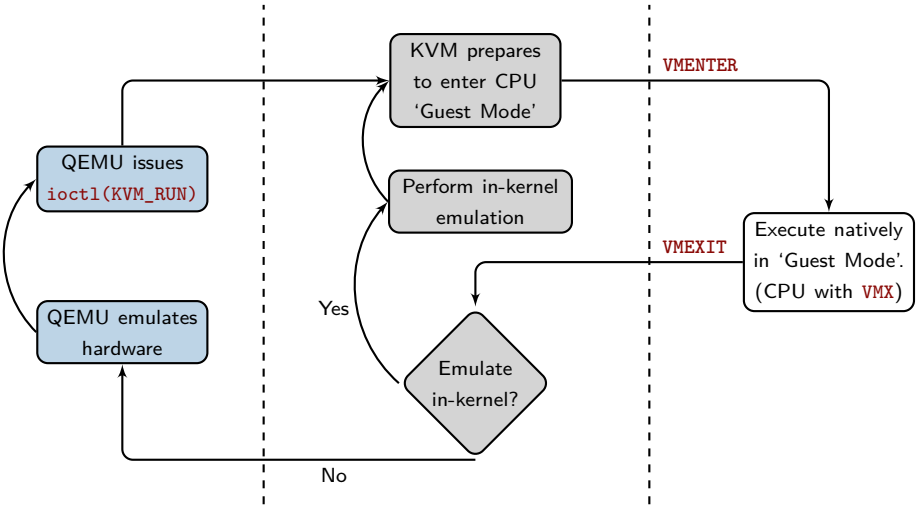
KVM and QEMU: the insides



KVM and QEMU: the insides



Hardware-based virtualization with KVM



Part I

Ways to configure virtual CPUs

x86: QEMU's default CPU models (a)

The default models (`qemu32`, `qemu64`) work on any host CPU

x86: QEMU's default CPU models (a)

The default models (`qemu32`, `qemu64`) work on any host CPU

But they are dreadful choices!

x86: QEMU's default CPU models (a)

The default models (`qemu32`, `qemu64`) work on any host CPU

But they are dreadful choices!

- No `AES` / `AES-NI`: critical for TLS performance
- No `RDRAND`: important for entropy
- No `PCID`: performance- & security-critical (thanks, Meltdown)

x86: QEMU's default CPU models (b)

```
$ cd /sys/devices/system/cpu/vulnerabilities/
$ grep . *
l1tf:Mitigation: PTE Inversion
mds:Vulnerable: ... no microcode; SMT Host state unknown
meltdown:Mitigation: PTI
spec_store_bypass:Vulnerable
spectre_v1:Mitigation: usercopy/swapgs barriers ...
spectre_v2:Mitigation: Full generic retpoline ...
```

x86: QEMU's default CPU models (b)

```
$ cd /sys/devices/system/cpu/vulnerabilities/
$ grep . *
l1tf:Mitiga
mds:Vulnerable: ... no microcode; SMT Host state unknown
meltdown:Mitigation: PTI
spec_store_bypass:Vulnerable
spectre_v1:Mitigation: usercopy/swapgs barriers ...
spectre_v2:Mitigation: Full generic retpoline ...
```

In a VM, running with qemu64

x86: QEMU's default CPU models (b)

```
$ cd /sys/devices/system/cpu/vulnerabilities/
$ grep . *
l1tf:Mitigation: PTE Inversion
mds:Vulnerable: ... no microcode; SMT Host state unknown
meltdown:Mi
spec_store_
spectre_v1:Mitigation: usercopy/swapgs barriers ...
spectre_v2:Mitigation: Full generic retpoline ...
```

Microarchitectural Data Sampling

x86: QEMU's default CPU models (b)

```
$ cd /sys/devices/system/cpu/vulnerabilities/
$ grep . *
l1tf:Mitigation: PTE Inversion
mds:Vulnerable: ... no microcode; SMT Host state unknown
meltdown:Mitigation: PTI
spec_store_bypass:Vulnerable
spectre_v1: ... sercopy/swapgs barriers ...
spectre_v2: ... ull generic retpoline ...
```

Spectre-NG

x86: QEMU's default CPU models (b)

```
$ cd /sys/devices/system/cpu/vulnerabilities/
$ grep . *
l1tf:Mitigation: PTE Inversion
mds:Vulnerable: ... no microcode; SMT Host state unknown
meltdown:Mitigation: PTI
spec_store_bypass:Vulnerable
spectre_v1:Mitigation: usercopy/swapgs barriers ...
spectre_v2:Mitigation: Full generic retpoline ...
```

⇝ **Always specify a CPU model; or use libvirt's `host-model`**

Defaults of non-x86 architectures?

AArch64: Doesn't provide a default guest CPU

```
$ qemu-system-aarch64 -machine virt -cpu help
```

Defaults of non-x86 architectures?

AArch64: Doesn't provide a default guest CPU

```
$ qemu-system-aarch64 -machine virt -cpu help
```

Default CPU depends on the "machine type"

Defaults of non-x86 architectures?

AArch64: Doesn't provide a default guest CPU

```
$ qemu-system-aarch64 -machine virt -cpu help
```

ppc64 — `'host'` for KVM; `'power8'` for TCG (pure emulation)

s390x — `'host'` for KVM; `'qemu'` for TCG

Configure guest CPU on the command-line

On **x86**, by default, the `qemu64` model is used:

```
$ qemu-system-x86_64 [...]
```

Configure guest CPU on the command-line

On **x86**, by default, the `qemu64` model is used:

```
$ qemu-system-x86_64 [...]
```

Specify a particular CPU model:

```
$ qemu-system-x86_64 -cpu Broadwell-noTSX-IBRS [...]
```

Configure guest CPU on the command-line

On **x86**, by default, the `qemu64` model is used:

```
$ qemu-system-x86_64 [...]
```

Specify a particular CPU model:

```
$ qemu-system-x86_64 -cpu Broadwell-noTSX-IBRS [...]
```

Named CPU model

Control guest CPU features

Enable or disable specific features for a vCPU model:

```
$ qemu-system-x86_64 \
   -cpu Haswell-noTSX-IBRS,vmx=off,pcid=on [...]
```

Control guest CPU features

Enable or disable specific features for a vCPU model:

```
$ qemu-system-x86_64 \
    -cpu Haswell-noTSX-IBRS,vmx=off,pcid=on [...]
```

Named CPU model

Control guest CPU features

Enable or disable specific features for a vCPU model:

```
$ qemu-system-x86_64 \
   -cpu Haswell-noTSX-IBRS,vmx=off,pcid=on [...]
```

Granular CPU flags

Control guest CPU features

Enable or disable specific features for a vCPU model:

```
$ qemu-system-x86_64 \
   -cpu Haswell-noTSX-IBRS,vmx=off,pcid=on [...]
```

To get the list of supported vCPU models:

```
$ qemu-system-x86_64 -cpu help

# Or via libvirt: virsh cpu-models x86_64
```

Part II

CPU modes, models, and flags

(1) Host passthrough

Exposes the host CPU model, features, etc. as-is to the VM

```
$ qemu-system-x86_64 -cpu host [...]
```

(1) Host passthrough

Exposes the host CPU model, features, etc. as-is to the VM

```
$ qemu-system-x86_64 -cpu host [...]
```

Caveats:

- No guarantee of a predictable CPU for the guest

(1) Host passthrough

Exposes the host CPU model, features, etc. as-is to the VM

```
$ qemu-system-x86_64 -cpu host [...]
```

Caveats:

- No guarantee of a predictable CPU for the guest
- Live migration is a no go with mixed host CPUs

(1) Host passthrough

Exposes the host CPU model, features, etc. as-is to the VM

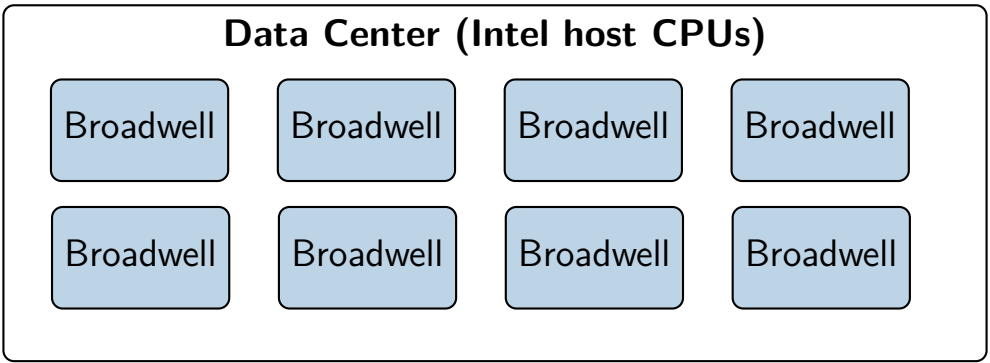
```
$ qemu-system-x86_64 -cpu host [...]
```

Caveats:

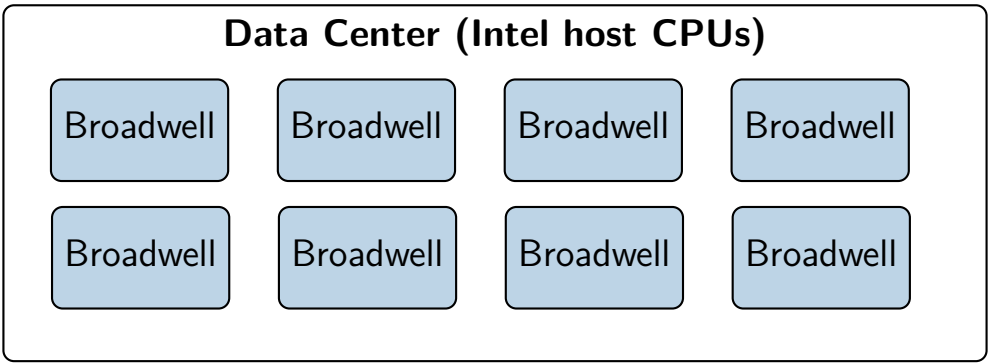
- No guarantee of a predictable CPU for the guest
- Live migration is a no go with mixed host CPUs

⇝ **Most performant; ideal—if live migration is not required**

(1) Host passthrough—when else to use it?



(1) Host passthrough—when else to use it?



⇝ **Along with identical CPUs, identical kernel *and* microcode are a must for VM live migration!**

(2) QEMU's named CPU models

Virtual CPUs typically model physical CPUs

Add or remove CPU features:

```
$ qemu-system-x86_64 -cpu Broadwell-IBRS,\
    vme=on,f16c=on,rdrand=on, \
    tsc_adjust=on,xsaveopt=on,\
    hypervisor=on,arat=off,   \
    pdpe1gb=on,abm=on [...]
```

(2) QEMU's named CPU models

Virtual CPUs typically model physical CPUs

Add or remove CPU features:

```
$ qemu-system-x86_64 -cpu Broadwell-IBRS,\
    vme=on,f16c=on,rdrand=on, \
    tsc_adjust=on,xsaveopt=on,\
    hypervisor=on,arat=off,   \
    pdpe1gb=on,abm=on [...]
```

$\rightsquigarrow$ **More flexible in live migration than 'host passthrough'**

(2) QEMU's named CPU models

QEMU is built with a number of pre-defined models:

```
$ qemu-system-x86_64 -cpu help
Available CPUs:
...
x86 Broadwell-IBRS      Intel Core Processor (Broadwell, IBRS)
...
x86 EPYC                AMD EPYC Processor
x86 EPYC-IBPB           AMD EPYC Processor (with IBPB)
x86 Haswell             Intel Core Processor (Haswell)
...
Recognized CPUID flags:
amd-ssbd apic arat arch-capabilities avx avx2 avx512-4fmaps
...
```

(3) `'host-model'`—a libvirt abstraction

Tackles a few things:

- Maximum possible CPU features from the host
- Live migration compatibility—with caveats
- Auto-adds critical guest CPU flags (e.g. `spec-ctrl`)

(3) `'host-model'`—a libvirt abstraction

Tackles a few things:

- Maximum possible CPU features from the host
- Live migration compatibility—with caveats
- Auto-adds critical guest CPU flags (e.g. `spec-ctrl`); *provided*—microcode, kernel, QEMU & libvirt are updated

(3) `'host-model'`—a libvirt abstraction

Tackles a few things:

- Maximum possible CPU features from the host
- Live migration compatibility—with caveats
- Auto-adds critical guest CPU flags (e.g. `spec-ctrl`); *provided*—microcode, kernel, QEMU & libvirt are updated

$\rightsquigarrow$ **Targets for the best of 'host passthrough' and named CPU models**

(3) ‘host-model’—example libvirt config

From a libvirt guest definition:

```
<cpu mode='host-model'>
  <feature policy='require' name='vmx'/>
  <feature policy='disable' name='pdpe1gb'/>
  ...
</cpu>
```

⇝ **libvirt will translate it into a suitable CPU model, based on ‘virsh domcapabilities’**

(3) `'host-model'` and live migration

As done by libvirt:

- Source vCPU definition is transferred as-is to the target
- On target: Migrated guest *retains* the source vCPU model

(3) `'host-model'` and live migration

As done by libvirt:

- Source vCPU definition is transferred as-is to the target
- On target: Migrated guest *retains* the source vCPU model

↳ But: When the guest cold-reboots, it can pick up extra CPU features

⇝ **Use `host-model` if live migration in both directions is not a requirement**

CPU config with management tools

Most tools offer some form of (e.g. from OpenStack):

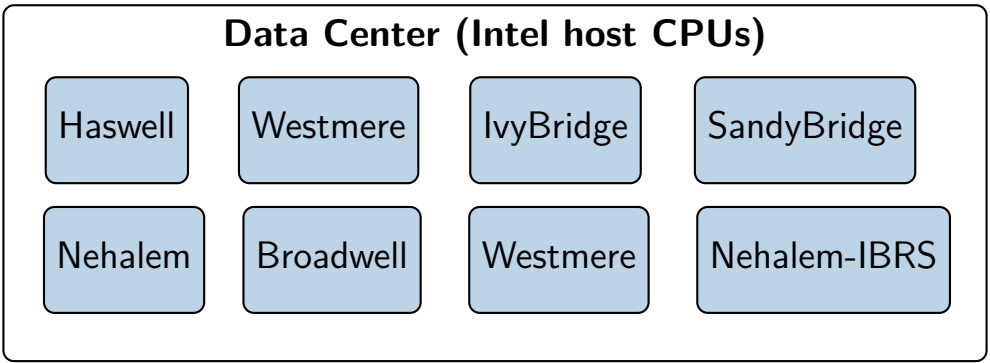
```
$ cat /etc/nova/nova.conf
...
[libvirt]
cpu_mode = custom  # or: host-model/host-passthrough
cpu_model = Broadwell-noTSX-IBRS
cpu_model_extra_flags = ssbd, pdpe1gb
...
```

⇝ **Possible CPU models/flags:** `'qemu-kvm -cpu help'`

Part III

Choosing CPU models & features

Finding compatible CPU models



Finding compatible CPU models

Problem: Determine a compatible model among CPU variants

Finding compatible CPU models

Problem: Determine a compatible model among CPU variants

Enter libvirt's APIs:

- `compareCPU()` and `baselineCPU()`
- `compareHypervisorCPU()` and `baselineHypervisorCPU()`

Available in libvirt 4.4.0+

Intersection between these two host CPUs?

```
$ cat Multiple-Host-CPUs.xml
<cpu mode='custom' match='exact'>
  <model fallback='forbid'>Haswell-noTSX-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy='require' name='vmx'/>
  <feature policy='require' name='rdrand'/>
</cpu>
<!-- Second CPU -->
<cpu mode='custom' match='exact'>
  <model fallback='forbid'>Skylake-Client-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy='disable' name='pdpe1gb'/>
  <feature policy='disable' name='pcid'/>
</cpu>
```

Intersection between these two host CPUs?

```
$ cat Multiple-Host-CPUs.xml

<cpu mode='custom' match='exact'>
  <model fallback='forbid'>Haswell-noTSX-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy='require' name='vmx'/>
  <feature policy='require' name='rdrand'/>
</cpu>
<!-- Second CPU -->
<cpu mode='custom' match='exact'>
  <model fallback='forbid'>Skylake-Client-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy='disable' name='pdpe1gb'/>
  <feature policy='disable' name='pcid'/>
</cpu>
```

Two CPU models

Use `baselineHypervisorCPU()` to determine it

```
$ virsh hypervisor-cpu-baseline Multiple-Host-CPUs.xml
<cpu mode='custom' match='exact'>
  <model fallback='forbid'>Haswell-noTSX-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy='require' name='rdrand'/>
  <feature policy='disable' name='pcid'/>
</cpu>
```

Use `baselineHypervisorCPU()` to determine it

```
$ virsh hypervisor-cpu-baseline Multiple-Host-CPUs.xml
<cpu mode='custom' match='exact'>
  <model fallback='forbid'>Haswell-noTSX-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy='require' name='rdrand'/>
  <feature policy='disable' name='pcid'/>
</cpu>
```

Intersection between our Haswell & Skylake variants

Use `baselineHypervisorCPU()` to determine it

```
$ virsh hypervisor-cpu-baseline Multiple-Host-CPUs.xml
<cpu mode='custom' match='exact'>
  <model fallback='forbid'>Haswell-noTSX-IBRS</model>
  <vendor>Intel</vendor>
  <feature policy='require' name='rdrand'/>
  <feature policy='disable' name='pcid'/>
</cpu>
```

$\rightsquigarrow$ **A "baseline" CPU model that permits live migration**

x86: QEMU's "machine types"

x86: QEMU's "machine types"

Two main purposes:

1. Emulate different chipsets (and related devices)—e.g. Intel's `i440FX` (a.k.a `'pc'`) and `Q35`

x86: QEMU's "machine types"

Two main purposes:

1. Emulate different chipsets (and related devices)—e.g. Intel's `i440FX` (a.k.a `'pc'`) and `Q35`
2. Provide a stable guest ABI—virtual hardware remains identical *regardless* of changes in host software / hardware

x86: QEMU's "machine types", versioned

```
$ qemu-system-x86_64 -machine help
...
pc                   Standard PC (i440FX + PIIX, 1996) (alias of pc-i440fx-4.2)
pc-i440fx-4.2        Standard PC (i440FX + PIIX, 1996) (default)
pc-i440fx-4.1        Standard PC (i440FX + PIIX, 1996)
...
q35                  Standard PC (Q35 + ICH9, 2009) (alias of pc-q35-4.2)
pc-q35-4.2           Standard PC (Q35 + ICH9, 2009)
pc-q35-4.1           Standard PC (Q35 + ICH9, 2009)
pc-q35-4.0.1         Standard PC (Q35 + ICH9, 2009)
...
```

x86: QEMU's "machine types", versioned

```
$ qemu-system-x86_64 -machine help
...
pc                   Standard PC (i440FX + PIIX, 1996) (alias of pc-i440fx-4.2)
pc-i440fx-4.2        Standard PC (i440FX + PIIX, 1996) (default)
                     Standard PC (i440FX + PIIX, 1996)
q35                  Standard PC (Q35 + ICH9, 2009) (alias of pc-q35-4.2)
pc-q35-4.2           Standard PC (Q35 + ICH9, 2009)
pc-q35-4.1           Standard PC (Q35 + ICH9, 2009)
pc-q35-4.0.1         Standard PC (Q35 + ICH9, 2009)
...
```

Traditional

x86: QEMU's "machine types", versioned

```
$ qemu-system-x86_64 -machine help
...
pc                   Standard PC (i440FX + PIIX, 1996) (alias of pc-i440fx-4.2)
pc-i440fx-4.2        Standard PC (i440FX + PIIX, 1996) (default)
pc-i440fx-4.1        Standard PC (i440FX + PIIX, 1996)
...
q35                  Standard PC (Q35 + ICH9, 2009) (alias of pc-q35-4.2)
                     tandard PC (Q35 + ICH9, 2009)
                     tandard PC (Q35 + ICH9, 2009)
pc-q35-4.0.1         Standard PC (Q35 + ICH9, 2009)
...
```

Recommended

⇝ **Versioned machine types provide stable guest ABI**

Why bother with machine types?

Changing machine types is guest-visible

Why bother with machine types?

Changing machine types is guest-visible

After a QEMU upgrade, when using libvirt:

- Need a distinct request to upgrade machine type

Why bother with machine types?

Changing machine types is guest-visible

After a QEMU upgrade, when using libvirt:

- Need a distinct request to upgrade machine type
- The guest needs a cold-reboot—an explicit stop + start; only *then* does it pick up the new machine type

Why bother with machine types?

Changing machine types is guest-visible

After a QEMU upgrade, when using libvirt:

- Need a distinct request to upgrade machine type
- The guest needs a cold-reboot—an explicit stop + start; only *then* does it pick up the new machine type

$\rightsquigarrow$ **Change machine types only after guest workload evaluation—CPU features & devices can differ**

Patching guest CPU models

1. Update **microcode**, host & guest **kernels**;
   refer to `/sys/devices/system/cpu/vulnerabilities/`

Patching guest CPU models

1. Update **microcode**, host & guest **kernels**;
   refer to `/sys/devices/system/cpu/vulnerabilities/`
2. Then, update **libvirt** & **QEMU** on the host

Patching guest CPU models

1. Update **microcode**, host & guest **kernels**;
   refer to `/sys/devices/system/cpu/vulnerabilities/`
2. Then, update **libvirt** & **QEMU** on the host
3. Now tell the management tool to **update guest CPUs to their patched variants**—e.g. the `*-IBRS` models
4. **Cold-reboot** the guests—to pick up new `CPUID` bits

⇝ **Guidance:** `qemu/docs/qemu-cpu-models.texi`

x86: Important CPU features

To provide mitigation for MDS, Spectre, Meltdown et al:

- Intel : `ssbd`, `pcid`, `spec-ctrl`, `tsx-ctrl`, `md-clear`, `mds-no`, `taa-no`
- AMD : `virt-ssbd`, `amd-ssbd`, `amd-no-ssb`, `ibpb`

x86: Important CPU features

To provide mitigation for MDS, Spectre, Meltdown et al:

- Intel : `ssbd`, `pcid`, `spec-ctrl`, `tsx-ctrl`, `md-clear`, `mds-no`, `taa-no`
- AMD : `virt-ssbd`, `amd-ssbd`, `amd-no-ssb`, `ibpb`

↳ Some of these are built into QEMU's **versioned CPU models**; refer to '`qemu-kvm -cpu help`'

Recap

→ **Don't use the built-in default**, `qemu64` model

→ Identical host CPUs? Go with `host-passthrough`

→ Mixed CPUs: `host-model`; or a custom baseline

→ **Evaluate workloads** before changing machine types

→ Pay attention to **CPU flags** when updating CPU models

References

- CPU model configuration for QEMU/KVM x86 hosts

  https://git.qemu.org/?p=qemu.git;a=blob;f=docs/qemu-cpu-models.texi

- Microarchitural Data Sampling (MDS) - Virtualization mitigation

  https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/mds.html

- Making use of Spectre/Meltdown mitigation for KVM guests

  https://www.qemu.org/2018/02/14/qemu-2-11-1-and-spectre-update

- Mitigating Spectre and Meltdown (and L1TF), by David Woodhouse

  https://kernel-recipes.org/en/2018/talks/mitigating-spectre-and-meltdown-vulnerabilities/

- Exploiting modern microarchitectures—Meltdown, Spectre, and other hardware attacks, by Jon Masters

  https://archive.fosdem.org/2018/schedule/event/closing_keynote