Effective Virtual CPU Configuration with QEMU and libvirt

Kashyap Chamarthy <kashyap@redhat.com>

Open Source Summit
Edinburgh, 2018

Timeline of recent CPU flaws, 2018 (a)

Jan 03	**Spectre v1**: Bounds Check Bypass
Jan 03	**Spectre v2**: Branch Target Injection
Jan 03	**Meltdown**: Rogue Data Cache Load
May 21	**Spectre-NG**: Speculative Store Bypass
Jun 21	**TLBleed**: Side-channel attack over shared TLBs

Timeline of recent CPU flaws, 2018 (b)

Jun 29 — **NetSpectre**: Side-channel attack over local network

Jul 10 — **Spectre-NG**: Bounds Check Bypass Store

Aug 14 — **L1TF**: "L1 Terminal Fault"

. . . — ?

What this talk is not about

What this talk is not about

Out of scope:

- Internals of various side-channel attacks
- How to exploit Meltdown & Spectre variants
- Details of performance implications

What this talk is not about

Out of scope:

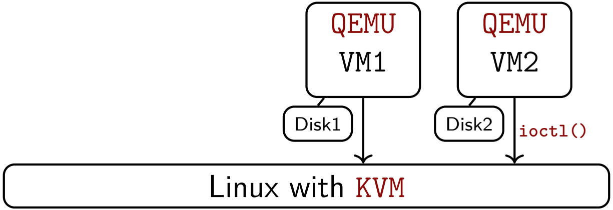
- Internals of various side-channel attacks
- How to exploit Meltdown & Spectre variants
- Details of performance implications

⇝ **Related talks in the ‘References’ section**

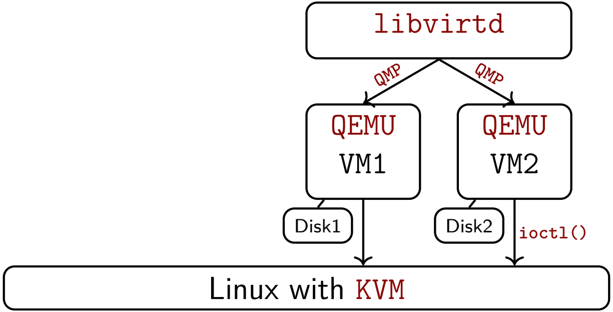
KVM-based virtualization components

Linux with KVM

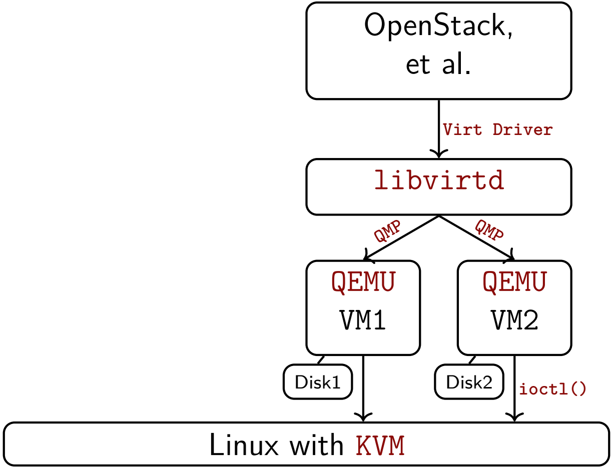
KVM-based virtualization components



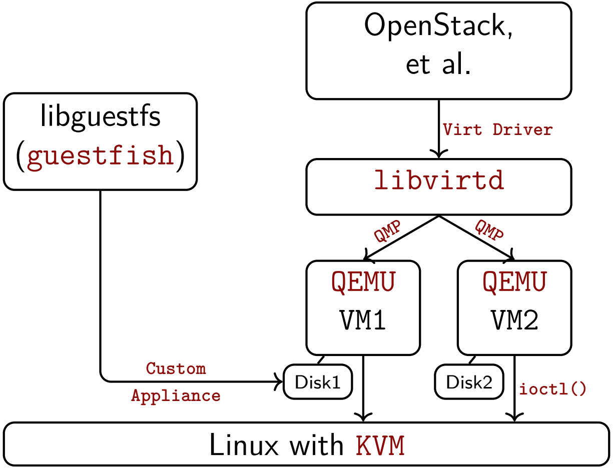
KVM-based virtualization components



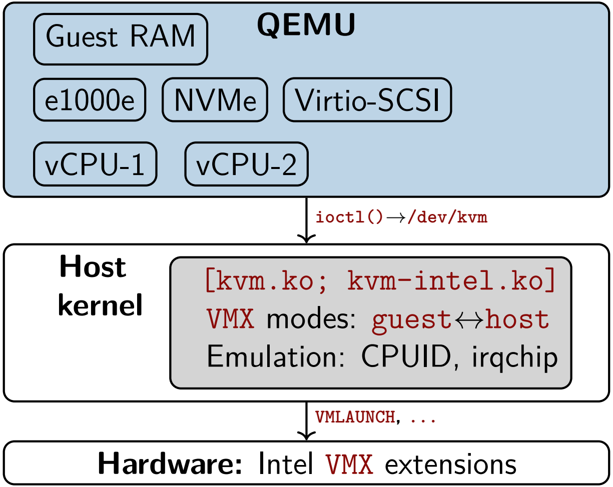
KVM-based virtualization components



KVM-based virtualization components



QEMU and KVM



QEMU

Guest RAM

e1000e

NVMe

Virtio-SCSI

vCPU-1

vCPU-2

`ioctl()`→`/dev/kvm`

Host kernel

`[kvm.ko; kvm-intel.ko]`

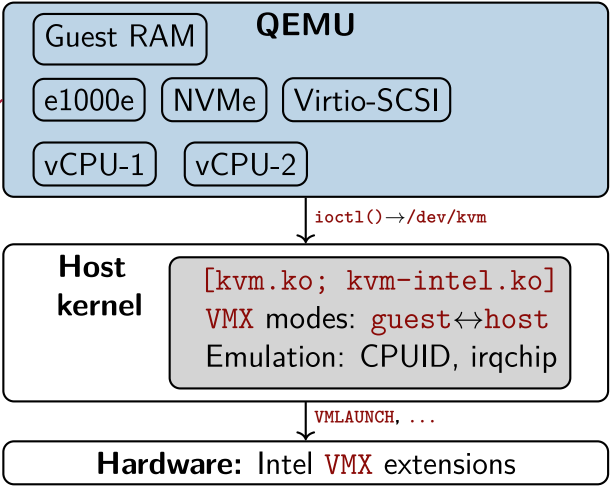
`VMX` modes: `guest`↔`host`

Emulation: CPUID, irqchip

`VMLAUNCH`, ...

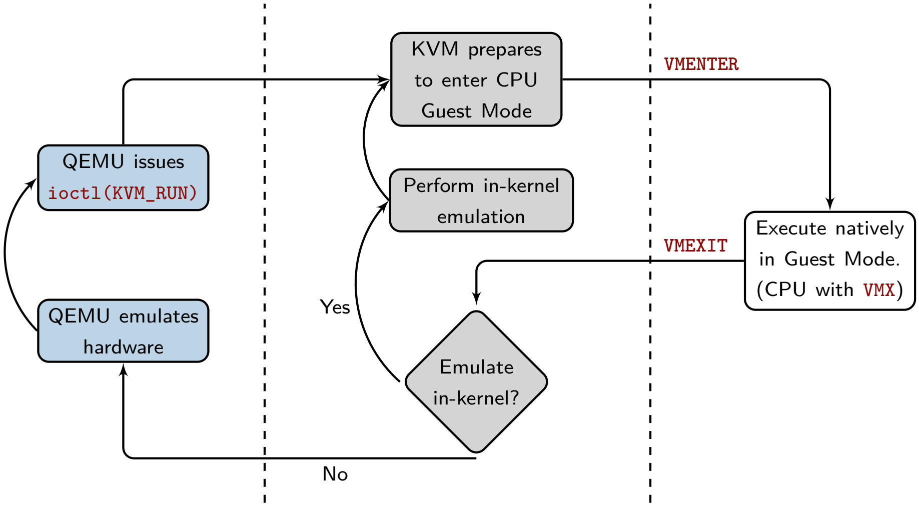
Hardware: Intel `VMX` extensions

QEMU and KVM



To inspect, use Linux tools: `top`, `kill`, . . .

Hardware-based virtualization with KVM


KVM prepares to enter CPU Guest Mode
VMENTER
QEMU issues ioctl(KVM_RUN)
Perform in-kernel emulation
VMEXIT
Execute natively in Guest Mode. (CPU with VMX)
Yes
QEMU emulates hardware
Emulate in-kernel?
No

Part I

Interfaces to configure vCPUs

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
meltdown:Mitigation: PTI
spec_store_bypass:Vulnerable
spectre_v1:Mitigation: __user pointer sanitization
spectre_v2:Mitigation: Full generic retpoline
```

x86: QEMU's default CPU models (b)

```
$ cd /sys/devices/system/cpu/vulnerabilities/
$ grep . *
l1tf:Mitiga
meltdown:Mitigation: PTI
spec_store_bypass:Vulnerable
spectre_v1:Mitigation: __user pointer sanitization
spectre_v2:Mitigation: Full generic retpoline
```

On a guest running with qemu64

x86: QEMU's default CPU models (b)

```
$ cd /sys/devices/system/cpu/vulnerabilities/
$ grep . *
l1tf:Mitigation: PTE Inversion
meltdown:Mitigation: PTI
spec_store_bypass:Vulnerable
spectre_v1:             _user pointer sanitization
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
meltdown:Mitigation: PTI
spec_store_bypass:Vulnerable
spectre_v1:Mitigation: __user pointer sanitization
spectre_v2:Mitigation: Full generic retpoline
```

⇝ **Always specify an explicit CPU model;
or use libvirt’s `host-model`**

Defaults of other architectures

AArch64: Doesn’t provide a default guest CPU

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

Defaults of other 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 other 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 CPU on the command-line

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

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

Configure 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 IvyBridge-IBRS [...]
```

Configure 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 IvyBridge-IBRS [...]
```

Named CPU model

Control guest CPU features

Enable or disable specific features for a vCPU model:

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

Control guest CPU features

Enable or disable specific features for a vCPU model:

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

For a list of supported vCPU models, refer to:

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

Or libvirt’s — ‘virsh cpu-models x86_64’

QEMU's CPU-related run-time interfaces

Granular details about vCPU models, their capabilities & more:

- `query-cpu-definitions`
- `query-cpu-model-expansion`
- `query-hotpluggable-cpus`
- `query-cpus-fast`; `device_{add,del}`

⇝ **`libvirtd` caches some of this data under `/var/cache/libvirt/qemu/capabilities/`**

Run-time: Probe QEMU for CPU model specifics

```
[Upstream-QEMU]$ ./qmp-shell -v -p /tmp/qmp-sock
(QEMU) query-cpu-definitions
    ...
    "return": [
        {  "typename": "Westmere-IBRS-x86_64-cpu",
           "unavailable-features": [],
           "migration-safe": true,
           "static": false,
           "name": "Westmere-IBRS" }]
    ... # Snip other CPU variants
```

redhat.

Part II

CPU modes, models and flags

Host passthrough

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

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

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 stable CPU for the guest

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 stable CPU for the guest
- Live migration is a no go with mixed host CPUs

Host passthrough

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

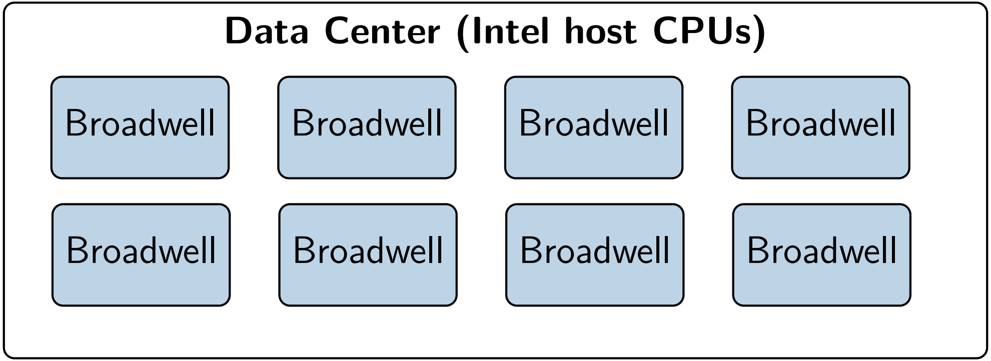
```
$ qemu-system-x86_64 -cpu host [...]
```

Caveats:

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

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

Host passthrough – when else to use it?

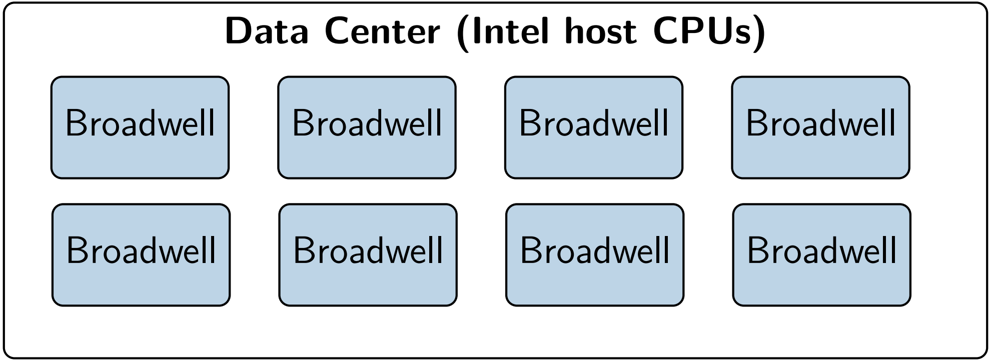


Data Center (Intel host CPUs)

Broadwell Broadwell Broadwell Broadwell

Broadwell Broadwell Broadwell Broadwell

Host passthrough – when else to use it?



Data Center (Intel host CPUs)

Broadwell Broadwell Broadwell Broadwell

Broadwell Broadwell Broadwell Broadwell

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

QEMU's named CPU models (a)

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 [...]
```

QEMU's named CPU models (a)

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 [...]
```

⇝ **More flexible in live migration than 'host passthrough'**

QEMU’s named CPU models (b)

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
...
```

‘host-model’ – a libvirt abstraction

Tackles a few problems:

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

‘`host-model`’ – a libvirt abstraction

Tackles a few problems:

- 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!

‘`host-model`’ – a libvirt abstraction

Tackles a few problems:

- 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!

⇝ **Targets for the best of ‘host passthrough’ and named CPU models**

‘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:** `/usr/share/libvirt/cpu_map/*.xml`

‘`host-model`’ and live migration

As done by libvirt:

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

‘`host-model`’ and live migration

As done by libvirt:

- Source vCPU definition is transferred as-is to the target
- On target: Migrated guest sees the *same* vCPU model
- But: When the guest ‘cold boots’, it may pick up *extra* CPU features—prevents migrating back to the source

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

OpenStack Nova and CPU models

Provides relevant config attributes:

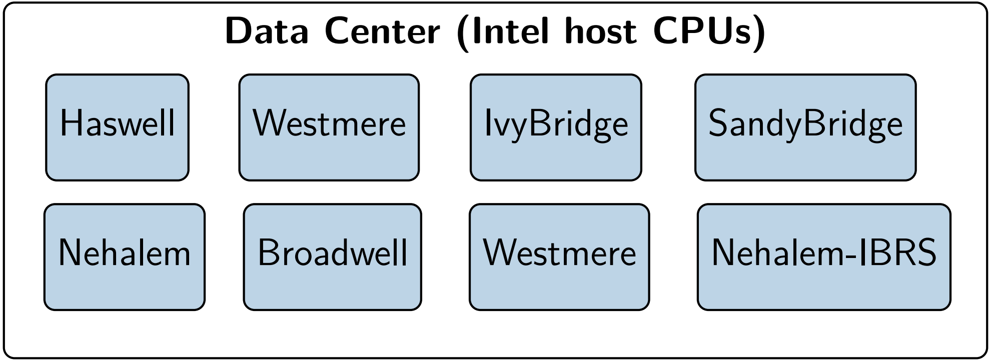
- cpu_mode
  - Can be: custom, host-passthrough; or host-model
- cpu_model & cpu_model_extra_flags
  - Refer to libvirt's /usr/share/libvirt/cpu_map/*.xml
  - Or QEMU's: qemu-system-x86_64 -cpu help

⇝ **Details in documentation of the above config attributes**
https://docs.openstack.org/nova/rocky/configuration/config.html

Part III

Choosing CPU models & features

Finding compatible CPU models



Data Center (Intel host CPUs)

Haswell

Westmere

IvyBridge

SandyBridge

Nehalem

Broadwell

Westmere

Nehalem-IBRS

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()`

↖ (New 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>
```

⇝ **A “baseline” model that permits live migration**

x86: QEMU's "machine types"

x86: QEMU's "machine types"

Two main purposes:

- 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:

- Emulate different chipsets (and related devices)—e.g. Intel's `i440FX` (a.k.a `'pc'`) and `Q35`
- Provide stable guest ABI—virtual hardware remains the same, *regardless* of changes in host software or hardware

x86: QEMU's "machine types" – versioned

```
$ qemu-system-x86_64 -machine help
...
pc                 Standard PC (i440FX + PIIX, 1996) (alias of pc-i440fx-3.0)
pc-i440fx-3.0      Standard PC (i440FX + PIIX, 1996) (default)
pc-i440fx-2.9      Standard PC (i440FX + PIIX, 1996)
...
q35                Standard PC (Q35 + ICH9, 2009) (alias of pc-q35-3.0)
pc-q35-3.0         Standard PC (Q35 + ICH9, 2009)
pc-q35-2.9         Standard PC (Q35 + ICH9, 2009)
pc-q35-2.8         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-3.0)
pc-i440fx-3.0   Standard PC (i440FX + PIIX, 1996) (default)
                Standard PC (i440FX + PIIX, 1996)

q35             Standard PC (Q35 + ICH9, 2009) (alias of pc-q35-3.0)
pc-q35-3.0      Standard PC (Q35 + ICH9, 2009)
pc-q35-2.9      Standard PC (Q35 + ICH9, 2009)
pc-q35-2.8      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-3.0)
pc-i440fx-3.0   Standard PC (i440FX + PIIX, 1996) (default)
pc-i440fx-2.9   Standard PC (i440FX + PIIX, 1996)
...
q35             Standard PC (Q35 + ICH9, 2009) (alias of pc-q35-3.0)
                Standard PC (Q35 + ICH9, 2009)
                Standard PC (Q35 + ICH9, 2009)
pc-q35-2.8      Standard PC (Q35 + ICH9, 2009)
...
```

Recommended

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

Machine types and CPU features

Changing machine types is guest-visible

Machine types and CPU features

Changing machine types is guest-visible

After a QEMU upgrade, when using libvirt:

- Need an explicit request for machine type upgrade
- The guest needs a 'cold-reboot' (i.e. an explicit stop + start)—to allow QEMU to re-`exec()`

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

x86: Recommended guest CPU models

Before configuring guest CPUs:

- Update microcode, host & guest kernels; refer to—`/sys/devices/system/cpu/vulnerabilities/`

x86: Recommended guest CPU models

Before configuring guest CPUs:

- Update microcode, host & guest kernels; refer to—`/sys/devices/system/cpu/vulnerabilities/`
- Update libvirt & QEMU—and explicitly update guest CPUs to patched variants (e.g. the `*-IBRS` models)
- Cold-reboot the guests—to pick up new `CPUID` bits

x86: Recommended guest CPU models

Before configuring guest CPUs:

- Update microcode, host & guest kernels; refer to—`/sys/devices/system/cpu/vulnerabilities/`
- Update libvirt & QEMU—and explicitly update guest CPUs to patched variants (e.g. the `*-IBRS` models)
- Cold-reboot the guests—to pick up new `CPUID` bits

⇝ **Guidance:** `qemu/docs/qemu-cpu-models.texi`
(Thanks, Daniel Berrangé)

x86: Important CPU flags

To mitigate guests from multiple Spectre & Meltdown variants:

- Intel: `ssbd`, `pcid`, `spec-ctrl`
- AMD: `virt-ssbd`, `amd-ssbd`, `amd-no-ssb`, `ibpb`

Some are built into QEMU's `*-IBRS` & `*-IBPB` CPU models

x86: Important CPU flags

To mitigate guests from multiple Spectre & Meltdown variants:

- Intel: `ssbd`, `pcid`, `spec-ctrl`
- AMD: `virt-ssbd`, `amd-ssbd`, `amd-no-ssb`, `ibpb`

Some are built into QEMU's `*-IBRS` & `*-IBPB` CPU models

⇝ **Details:**

qemu/docs/qemu-cpu-models.texi

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

Future ‘expectations’ from applications?

> “QEMU and libvirt took the joint decision to stop adding new named CPU models when CPU vulnerabilities are discovered from this point forwards. Applications / users would be expected to turn on CPU features explicitly as needed and are considered broken if they don’t provide this functionality.”
>
> — “CPU model versioning separate from machine type versioning”
> From ‘`qemu-devel`’ mailing list

References

- CPU model configuration for QEMU/KVM x86 hosts, by Daniel Berrangé

  `https://www.berrange.com/posts/2018/06/29/cpu-model-configuration-for-qemu-kvm-on-x86-hosts`

- 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`

- KVM and CPU feature enablement, by Eduardo Habkost

  `https://wiki.qemu.org/images/c/c8/Cpu-models-and-libvirt-devconf-2014.pdf`

Questions?

E-mail: `kashyap@redhat.com`
IRC: `kashyap` – Freenode & OFTC

Related talks at the KVM Forum

(1) Security in QEMU: How Virtual Machines Provide Isolation — by Stefan Hajnoczi

- Happening now, but it's being recorded

(2) What Did Spectre and Meltdown Teach about CPU Models? — by Paolo Bonzini

- 26-OCT, Wednesday: 11:30 – 12:00