

Effective Virtual CPU Configuration with QEMU and libvirt

Kashyap Chamarthi <kchamart@redhat.com>

NLUUG

Utrecht, 2019


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”
- Nov 01 ■ **PortSmash**: Impacts SMT processors

Timeline of recent CPU flaws, 2019

- 
- May 14 ■ **ZombieLoad**: Leaks data across protection boundaries
 - May 14 ■ **RIDL**: Rogue In-Flight Data Load
 - May 14 ■ **Fallout**: Leaks data from Store Buffers
 - ... ■ ?

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
- Detailed performance analysis

What this talk is not about

Out of scope:

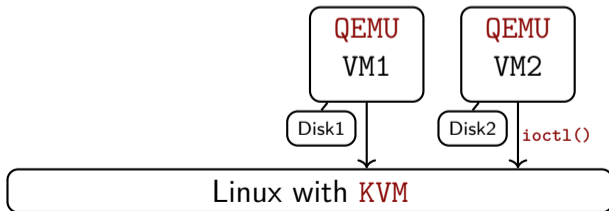
- Internals of various side-channel attacks
- How to exploit Meltdown & Spectre variants
- Detailed performance analysis

↪ **Related talks in the 'References' section**

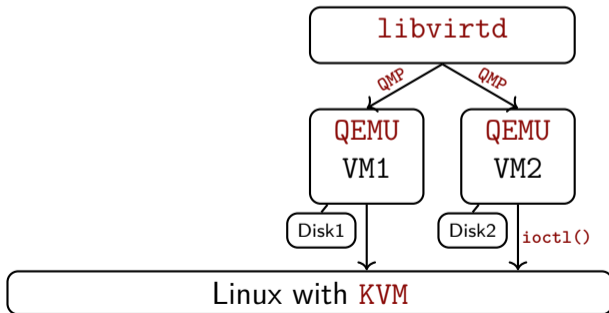
KVM-based virtualization components

Linux with KVM

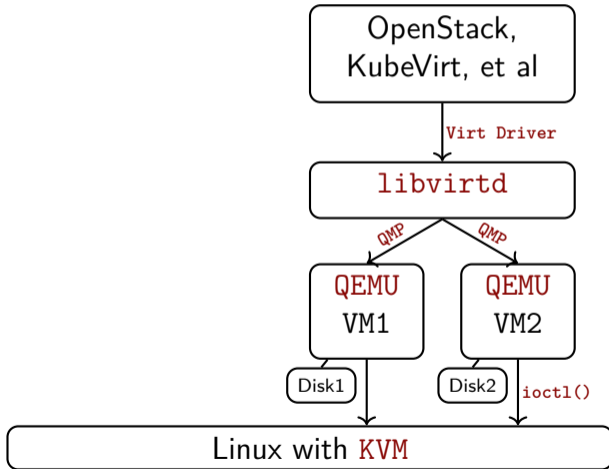
KVM-based virtualization components



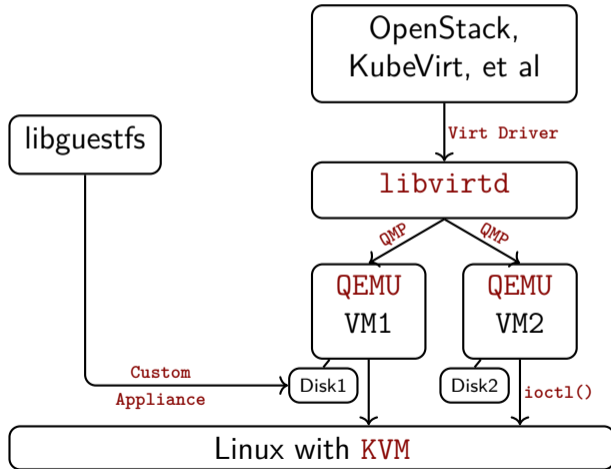
KVM-based virtualization components



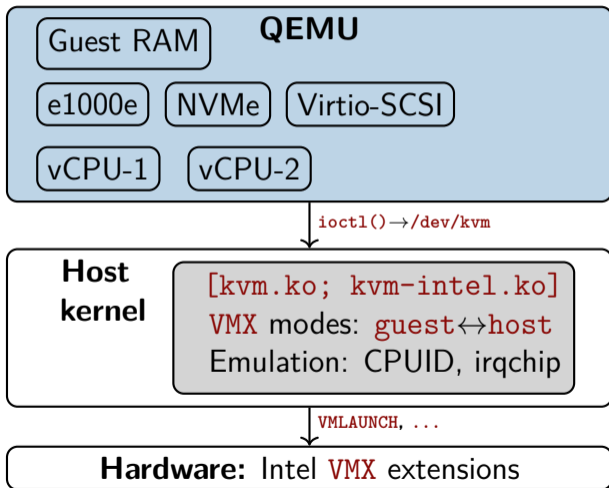
KVM-based virtualization components



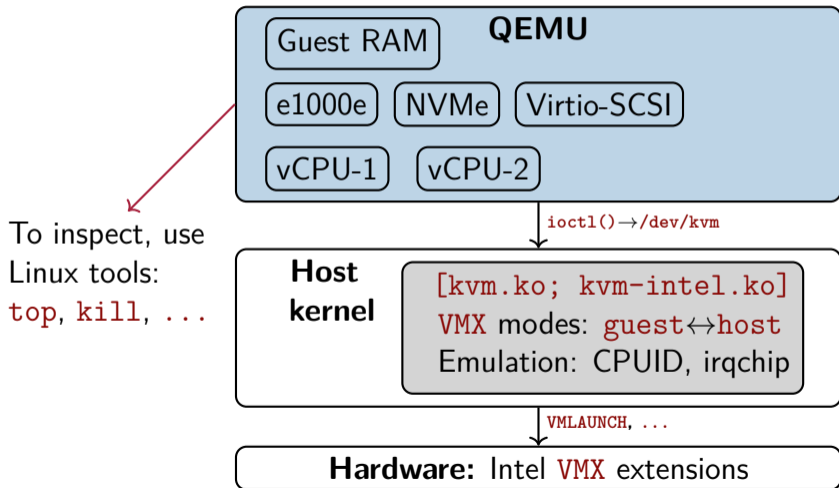
KVM-based virtualization components



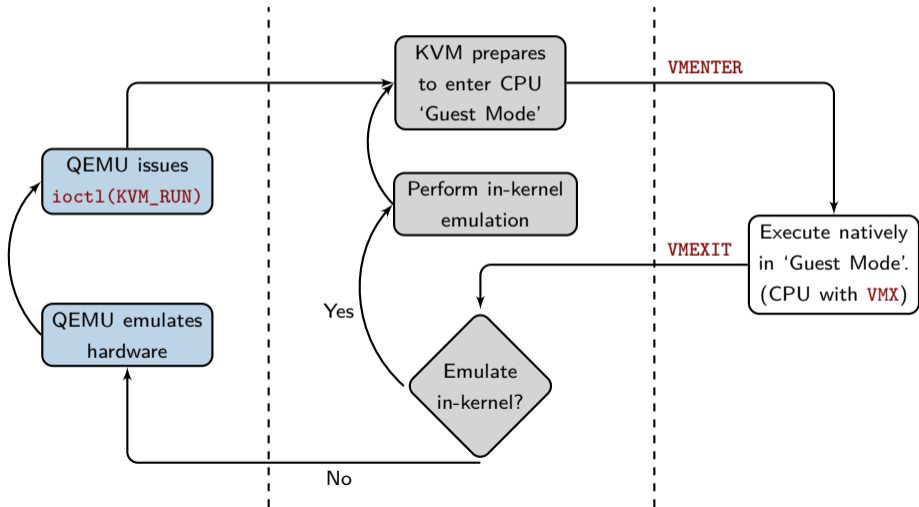
QEMU and KVM



QEMU and KVM



Hardware-based virtualization with KVM



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 . *
```

On a guest running with qemu64

```
l1tf:Mitigation:
```

```
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:Mitigation: PTE Inversion  
meltdown:Mitigation: PTI  
spec_store_bypass:Vulnerable  
spectre_v1: Spectre-NG _user pointer sanitization  
spectre_v2: Spectre-NG full generic retpoline
```

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}`

↪ **libvirt runs some of these at its daemon start-up time, and caches the results**

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?

Data Center (Intel host CPUs)

Broadwell

Broadwell

Broadwell

Broadwell

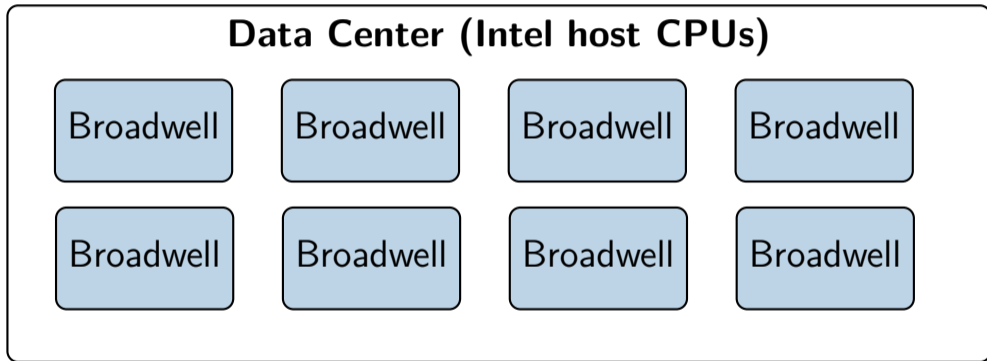
Broadwell

Broadwell

Broadwell

Broadwell

(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 (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 [...]
```

(2) 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'**

(2) 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
...
```

(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

↪ **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 sees the *same* 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 sees the same vCPU model**
- **But:** When the guest 'cold-reboots', it may pick up *extra* CPU features—**prevents migrating back to the source host**

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

OpenStack: Nova and CPU models

Relevant config attributes in the `[libvirt]` section of `/etc/nova/nova.conf` (details in the docs):

- `cpu_mode`
 - Can be: `host-model`, `host-passthrough`, or `custom`
- `cpu_model` & `cpu_model_extra_flags`
 - Possible values from the `<cpu>` element in:
`'virsh domcapabilities'`
 - And from: `'qemu-system-x86_64 -cpu help'`

↪ **Other clients of libvirt & QEMU offer equivalents**

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()`



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>
```

⇒ A “baseline” CPU 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 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-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)  
Traditional  
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)
pc-i440fx-2.9     Standard PC (i440FX + PIIX, 1996)
...
q35             Standard PC (Q35 + ICH9, 2009) (alias of pc-q35-3.0)
Recommended   Standard PC (Q35 + ICH9, 2009)
                  Standard PC (Q35 + ICH9, 2009)
pc-q35-2.8        Standard PC (Q35 + ICH9, 2009)
...
```

⇒ **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 to upgrade machine type**

Machine types and CPU features

Changing machine types is **guest-visible**

After a QEMU upgrade, when using libvirt:

- Need **an explicit request to upgrade machine type**
- The guest needs a 'cold-reboot' (i.e. an explicit stop + start)—**only then** does it pick up the new machine type

Machine types and CPU features

Changing machine types is **guest-visible**

After a QEMU upgrade, when using libvirt:

- Need **an explicit request to upgrade machine type**
- The guest needs a 'cold-reboot' (i.e. an explicit stop + start)—**only then does it pick up the new machine type**

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

Procedure to patch guest CPU models

- First, update **microcode**, host & guest **kernels**;
refer to `/sys/devices/system/cpu/vulnerabilities/`

Procedure to patch guest CPU models

- First, update **microcode**, host & guest **kernels**;
refer to `/sys/devices/system/cpu/vulnerabilities/`
- Next, update **libvirt** & **QEMU**

Procedure to patch guest CPU models

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

↪ **Related guidance:** [qemu/docs/qemu-cpu-models.texi](#)

x86: Important CPU flags

To mitigate guests from multiple Spectre & Meltdown variants:

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

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

~> **Details:**

<https://www.qemu.org/2018/02/14/qemu-2-11-1-and-spectre-update>
& `qemu/docs/qemu-cpu-models.texi`

‘Expectations’ from clients of QEMU & libvirt?

“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”
(A thread on ‘qemu-devel’ mailing list)

Summary

- Identical host CPUs? Go with **host-passthrough**
- Mixed CPUs: **host-model** is **recommended**;
otherwise, work out a 'custom baseline' CPU
- **Evaluate workloads** before changing machine types
- **Take advantage of live migration** to minimize guest downtime

References



CPU model configuration for QEMU/KVM x86 hosts

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



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: kchamart@redhat.com

IRC: kashyap – Freenode & OFTC