CPU models after Spectre & Meltdown

Paolo Bonzini
Red Hat, Inc.
KVM Forum 2018

“Can this guest run on that machine?”

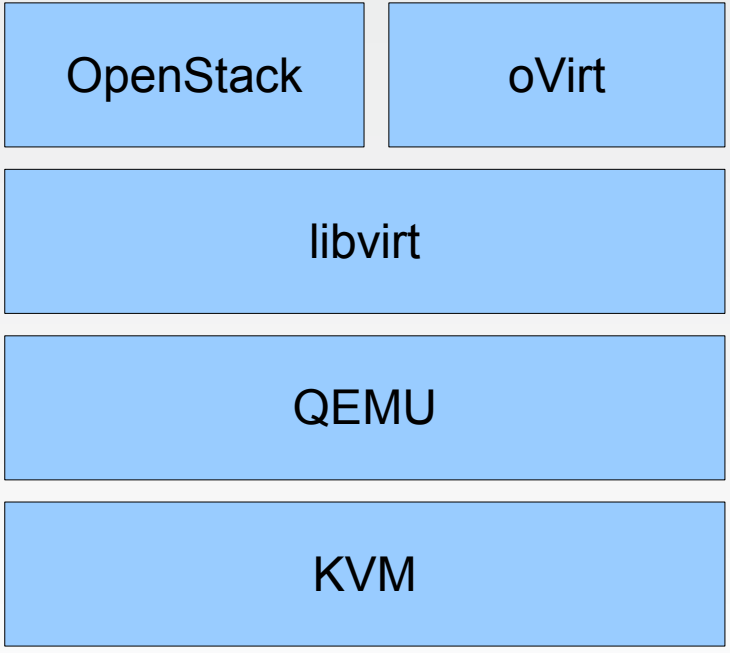
- It depends!
- Host processor
- Microcode version
- Kernel version
- QEMU
- Machine type

How can things go wrong?

- After upgrading microcode a CPUID flag disappears (TSX)
- After downgrading microcode a CPUID flag disappears (IBRS)
- After upgrading QEMU a new kernel is required (e.g. kvm-pv-eoi requires 3.6)
- After changing a kernel module parameter a flag disappears (e.g. VMX)
- When migrating to an older kernel, vhost drops support for some virtio features

“Can this guest run on that machine?”

- Lower levels of the stack affect the availability of features
- Higher levels of the stack decide what features to enable



Why CPU models?

- Most processor features must be present on the host for the guest to use them
- Guest ABI must not change across live migration
- CPU models describe the guest processor ABI in order to:
  - set up identical ABI on source and destination
  - discern which hosts you can start your VM

What's in a CPU model?

- Family/model/stepping ("f/m/s")

```
cpu family      : 6
model           : 142
stepping        : 9
```

Not necessarily the actual processor you're running on

- Model name

```
model name      : Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz
```

- Flags

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush …
```

- In the future, MSR values

CPU models in QEMU

- Named configurations allow scheduling flexibility
  - Processor code names (`-cpu IvyBridge`)
  - Least common denominator (`-cpu kvm64`)
  - Whatever TCG provided when the model was added (`-cpu qemu64`)
- Passthrough configurations are simple and perform well:
  - Only features that QEMU knows about (`-cpu host`)
  - All features implemented by KVM (`-cpu host,migratable=off`)
- Default for non-x86 architectures is `-cpu host`
- Default for x86 is `-cpu qemu{32,64}` (horrible)

-cpu qemu64

```
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
```


CPU models in libvirt

- Name + flags
- `host-passthrough`
  - Same as `-cpu host` (with default `migratable=on`)
  - User ensures same processor+kernel+microcode across migration source and destination
- `host-model`
  - Automatic conversion to name + flags
  - Restricted to flags known by KVM+QEMU+libvirt
  - Configuration preserved by libvirt across migration
  - Live migration from new to old host will fail gracefully

Spectre & Meltdown vs. CPU flags

- CPUID bits report availability of mitigations (IBRS, IBPB, SSBD, L1FLUSH, …)
- MSR bits report non-vulnerable CPUs (RDCL_NO, RSBA)
- Chicken bit availability is usually keyed by f/m/s
- Unused features become important for performance (PCID)

CPUID bits

- Add new models, or require manual addition of flags?
  - Adding new models was done for IBRS/IBPB (Spectrev2) but it doesn’t scale
  - On the other hand, the list of flags constantly grows: ssbd, pcid, spec-ctrl, virt-ssbd, amd-ssbd, amd-no-ssb, ibpb
  - As of January 2018, OpenStack did not support adding flags
- Not adding any more models after Spectrev2
- Libvirt `host-model` will add critical flags automatically, provided everything is updated

MSR bits

- Reading host MSR features is a privileged operation
  - QEMU cannot probe processor features that are exposed via MSRs
  - In any case, available on the host != supported by KVM
- New KVM ioctls
  - KVM_GET_MSR_FEATURE_INDEX_LIST (return list of MSRs that contain KVM capabilities)
  - KVM_GET_MSR on /dev/kvm (return host capabilities that are exposed via M`SRs)
  - Already used for microcode revision, VMX capabilities, etc.
- QEMU and libvirt users shouldn’t care about CPUID vs. MSR

MSR bits

- IA32_ARCH_FACILITIES
  - Discovered via CPUID
  - Provides extra information on speculative execution mitigations
  - Support added to KVM, not yet in QEMU
- Both “positive” and “negative” features included
  - RDCL_NO - “No rogue data cache load”
  - RSBA - “Always stuff return buffer”

Positive vs. negative features

- Return stack – a simple indirect branch predictor, specific to call/ret
  - On Skylake, empty return stack falls back to BTB
  - OS can do dummy calls at certain points to keep it full
- RSBA (“RSB alternate”?) lets guests avoid checking f/m/s

“When RSBA is set, it indicates that the VM may run on a processor vulnerable to exploits of Empty RSB conditions regardless of the processor’s DisplayFamily/DisplayModel”

Positive vs. negative features

- RDCL_NO - “No rogue data cache load”
  - Must be *clear* if the guest will ever migrate to an unfixed processor
  - Safe choice leads to suboptimal performance
  - Works just like any CPUID bit
- RSBA - “Always stuff return stack”
  - Must be *set* if the guest will ever migrate to Skylake
  - I started this VM one year ago. How was I supposed to know⁉
- Please Intel, DO NOT define negative features!

Chicken bits

- Disable features of the processor for debugging or “emergency” reasons
- Typically set by firmware, sometimes by OS
  - In a guest, we just pass the value that was set by the host
  - Nothing to do in the firmware, just another MSR-based feature
- The MSRs can be provided for any host or guest f/m/s
  - No conflict yet…

So many models!

- Different features between low-end and high-end platforms
  - Consumer/workstation: Pentium, Core, Xeon E3
  - Server: Xeon E5/E7 (now Gold and Platinum)
- Hard to know exactly which features were in which processor!
  - Even harder to know which features *will be* in unreleased processors
  - Result: some models in QEMU are “wrong” (missing or extra CPUID bits due to consumer vs. server)
  - Lack of PCID went unnoticed for some models until Meltdown

What makes CPU models hard?

- Dozens of flags, sometimes interrelated (AVX makes no sense without XSAVE)
- Flags defined in a hurry
- Proliferation of processor SKUs
- Extreme backwards compatibility
- QMP API limitations

Backwards compatibility

- Libvirt wants to keep every past XML runnable and with the same guest ABI
- Not only *virsh define*-d persistent guests; the user XML too
- “pc” machine type default
- “qemu64” CPU model default

QMP and Libvirt limitations

- Libvirt doesn’t have a way to query CPU model changes across QEMU machine types
  - If CPU models change, Libvirt cannot precisely compute runnability anymore
  - Rule: we can't change the CPU model "runnability" between machine-types
- Impossible to add new features provided by new kernels
- How to do better?

Versioned CPU models!

- CPU models exist in multiple versions
- Two ways to specify the version:
  - `-cpu Name-X.Y` (e.g. `-cpu Haswell-3.1`)
  - `-cpu Name,version=X.Y` (e.g. `-cpu Haswell,version=3.1`)
- Libvirt can query all CPU model versions via `query-cpu-model-expansion`
- Machine types specify the version through compat properties
- Users can still override the default version

Open issues

- What is the minimal required kernel version by QEMU?
  - Currently 3.6 for Intel
  - Example: adding nested virtualization by default would bump the minimum kernel version to 4.20
- MAXPHYADDR
  - Different across SKUs, not easily virtualizable
  - Must be solved in KVM
- Duplication of CPU models between QEMU and Libvirt

Questions?