# CPUID Handling (part 3)

Revision 1

June 8, 2017

## Contents



## 1 Current state

At early boot, Xen enumerates the features it can see, takes into account errata checks and command line arguments, and stores this information in the `boot_cpu_data.x86_capability[]` bitmap. This gets adjusted as APs boot up, and is sanitised to disable all dependent leaf features.

At mid/late boot (before dom0 is constructed), Xen performs the necessary calculations for guest cpuid handling. Data are contained within the `struct cpuid_policy` object, which is a representation of the architectural CPUID information as specified by the Intel and AMD manuals.

There are a few global `cpuid_policy` objects. First is the **raw_policy** which is filled in from native `CPUID` instructions. This represents what the hardware is capable of, in its current firmware/microcode configuration.

The next global object is **host_policy**, which is derived from the **raw_policy** and `boot_cpu_data.x86_capability[]`. It represents the features which Xen knows about and is using. Next, the **pv_max_policy** and **hvm_max_policy** are derived from the **host_policy**, and represent the upper bounds available to guests.

The toolstack may query for the **{raw,host,pv,hvm}_featureset** information using *XEN_SYSCTL_get_cpu_featureset*. This is bitmap form of the feature leaves only.

When a new domain is created, the appropriate **{pv,hvm}*max*policy** is duplicated as a starting point, and can be subsequently mutated indirectly by some hypercalls (*XEN_DOMCTL_{set_address_size,disable_migrate,settscinfo}*) or directly by *XEN_DOMCTL_set_cpuid*.

# 2 Issues with the existing hypercalls

*XEN_DOMCTL_set_cpuid* doesn't have a return value which the domain builder pays attention to. This is because, before CPUID part 2, there were no failure conditions, as Xen would accept all toolstack-provided data, and attempt to audit it at the time it was requested by the guest. To simplify the part 2 work, this behaviour was maintained, although Xen was altered to audit the data at hypercall time, typically zeroing out areas which failed the audit.

There is no mechanism for the toolstack to query the CPUID configuration for a specific domain. Originally, the domain builder constructed a guests CPUID policy from first principles, using native `CPUID` instructions in the control domain. This functioned to an extent, but was subject to masking problems, and is fundamentally incompatible with HVM control domains or the use of *CPUID Faulting* in newer Intel processors.

CPUID phase 1 introduced the featureset information, which provided an architecturally sound mechanism for the toolstack to identify which features are usable for guests. However, the rest of the CPUID policy is still generated from native `CPUID` instructions.

The `cpuid_policy` is per-domain information. Most CPUID data is identical across all CPUs. Some data are dynamic, based on other control settings (APIC, OSXSAVE, OSPKE, OSLWP), and Xen substitutes these appropriately when the information is requested.. Other areas however are topology information, including thread/core/socket layout, cache and TLB hierarchy. These data are inherited from whichever physical CPU the domain builder happened to be running on when it was making calculations. As a result, it is inappropriate for the guest under contraction, and usually entirely bogus when considered alongside other data.

# 3 Other problems

There is no easy provision for features at different code maturity levels, both in the hypervisor, and in the toolstack.

Some CPUID features have top-level command line options on the Xen command line, but most do not. On some hardware, some features can be hidden indirectly by altering the `cpuid_mask_*` parameters. This is a problem for developing new features (which want to be off-by-default but able to be opted in to), debugging, where it can sometimes be very useful to hide features and see if a problem reoccurs, and occasionally in security circumstances, where disabling a feature outright is an easy stop-gap solution.

From the toolstack side, given no other constraints, a guest gets the hypervisor-max set of features. This set of features is a trade off between what is supported in the hypervisor, and which features can reasonably be offered without impeding the migrateability of the guest. There is little provision for features which can be opted in to at the toolstack level, and those that are are done so via ad-hoc means.

# 4 Proposal

First and foremost, split the current **max_policy** notion into separate **max** and **default** policies. This allows for the provision of features which are unused by default, but may be opted in to, both at the hypervisor level and the toolstack level.

At the hypervisor level, **max** constitutes all the features Xen can use on the current hardware, while **default** is the subset thereof which are supported features, the features which the user has explicitly opted in to, and excluding any features the user has explicitly opted out of.

A new `cpuid=` command line option shall be introduced, whose internals are generated automatically from the featureset ABI. This means that all features added to `include/public/arch-x86/cpufeatureset.h` automatically gain command line control. (RFC: The same top level option can probably be used for non-feature CPUID data control, although I can't currently think of any cases where this would be used Also find a sensible way to express 'available but not to be used by Xen', as per the current `smep` and `smap` options.)

At the guest level, **max** constitutes all the features which can be offered to each type of guest on this hardware. Derived from Xen's **default** policy, it includes the supported features and explicitly opted in to features, which are appropriate for the guest.

The guests **default** policy is then derived from its **max**, and includes the supported features which are considered migration safe. (RFC: This distinction is rather fuzzy, but for example it wouldn't include things like ITSC by default, as that is likely to go wrong unless special care is taken.)

All global policies (Xen and guest, max and default) shall be made available to the toolstack, in a manner similar to the existing *XEN_SYSCTL_get_cpu_featureset*

mechanism. This allows decisions to be taken which include all CPUID data, not just the feature bitmaps.

New *XEN_DOMCTL_{get,set}_cpuid_policy* hypercalls will be introduced, which allows the toolstack to query and set the cpuid policy for a specific domain. It shall supersede *XEN_DOMCTL_set_cpuid*, shall fail if Xen is unhappy with any aspect of the policy during auditing.

When a domain is initially created, the appropriate guests **default** policy is duplicated for use. When auditing, Xen shall audit the toolstacks requested policy against the guests **max** policy. This allows experimental features or non-migration-safe features to be opted in to, without those features being imposed upon all guests automatically.

A guests CPUID policy shall be immutable after construction. This better matches real hardware, and simplifies the logic in Xen to translate policy alterations into configuration changes.

(RFC: Decide exactly where to fit this. *XEN_DOMCTL_max_vcpus* perhaps?) The toolstack shall also have a mechanism to explicitly select topology configuration for the guest, which primarily affects the virtual APIC ID layout, and has a knock on effect for the APIC ID of the virtual IO-APIC. Xen's auditing shall ensure that guests observe values consistent with the guarantees made by the vendor manuals.

The `disable_migrate` field shall be dropped. The concept of migrateability is not boolean; it is a large spectrum, all of which needs to be managed by the toolstack. The simple case is picking the common subset of features between the source and destination. This becomes more complicated e.g. if the guest uses LBR/LER, at which point the toolstack needs to consider hardware with the same LBR/LER format in addition to just the plain features.

`disable_migrate` is currently only used to expose ITSC to guests, but there are cases where is perfectly safe to migrate such a guest, if the destination host has the same TSC frequency or hardware TSC scaling support.

Finally, `disable_migrate` doesn't (and cannot reasonably) be used to inhibit state gather operations, as this interferes with debugging and monitoring tasks.