CPUID improvements, phase 2

November 8, 2016

Contents

1 Executive Summary

Previous CPUID work in the Xen 4.7 time-frame provided remedial improvements for feature levelling, by having Xen decide what it would tolerate as the maximum featureset for each type of guest and offer this information to the toolstack.

The purpose of that work was to allow a toolstack to be able to safely migrate a guest between two non-identical servers; first by the toolstack having an accurate idea of the hardware capabilities, and second by ensuring that the guest can only see the features permitted.

There were known issues at the start which were deliberately not addressed, and further issues became evident during the work. In particular, Xen's internal handling of guests CPUID information is in need of improvements.

The purpose of the phase 2 work is to improve the internals of CPUID handling in Xen and the lower levels of the toolstack.

2 Identified Issues

2.1 Privileged domain CPUID policies

The control domain and hardware domain have no policy data in Xen. This is a consequence of how the guest CPUID logic was developed. The lack of policy data is worked around in `pv_cpuid()` with logic looking like:

```
if ( !is_control_domain(currd) && !is_hardware_domain(currd) )
    domain_cpuid(currd, leaf, subleaf, &a, &b, &c, &d);
else
    cpuid_count(leaf, subleaf, &a, &b, &c, &d);
```

but this is flawed for several reasons.

There is nothing special about these domains when it comes to CPUID; they should still see CPUID data appropriate for their VM container rather than leaking all hardware state.

The default leaking of hardware state thus far has caused anti-features to enter into the ABI, most notably OXSAVE and MTRR handling for PV guests.

There is also no equivalent logic for HVM control/hardware domains.

2.2 Domain construction

Currently, only PV domains are viable for use as the control domain in production scenarios. As the control domain has previously had full hardware CPUID information leaked into it, the domain construction logic has become reliant on this leakage.

In particular the domain construction logic expects, by default, to seed xsave state information from what it can find in native CPUID. As PV guests are more feature restricted than HVM guests, this fails to construct an HVM guest with all available features if the control domain's CPUID view is properly restricted.

A workaround for this was left over from the feature levelling work to avoid enabling CPUID Faulting on the control domain until this issue is resolved.

2.3 Internal CPUID representation

Xen's internal representation of CPUID policy information for a guest is an unordered list. This list gets searched for all leaf lookups, and contains unaudited toolstack values.

Because the unordered list contains unaudited toolstack data, the main CPUID handling functions attempt to retrofit some sanity to the contents of the list. This is recalculated for every leaf lookup.

Because the toolstack provided data isn't in a flat format, logic in Xen needs to make calls into the CPUID handling functions to find information. This causes searching and recalculation even for information such as maxphysaddr or feature flags which should be accessible in O(1) time.

Some CPUID information depends on other CPUID information. As the data isn't in a flat format, handling of things like xsave state or max_leaf boundary checking requires recursion to obtain, and extra special care to not end up with a hypervisor stack overflow.

2.4 Topology information

Xen has very little concept of topology information in the CPUID policy. Some values (e.g. APIC ID) are hard-coded based on VCPU ID, while some values (e.g. the cache detail in leaf 4) are blindly propagated from the static toolstack-provided values, without understanding that they should vary by vcpu.

The result is very confusing to a VM, as the values it reads across its vcpus do not match the expectations set out by the Intel/AMD software manuals.

3 Proposed Remediation

3.1 Flat internal representation

The following structure is proposed for containing a domains CPUID policy:

```
struct cpuid_leaf { uint32_t a, b, c, d; };

struct cpuid_policy {
    union {
        struct cpuid_leaf raw[CPUID_GUEST_NR_BASIC];
        struct {
            struct {
                uint32_t max_leaf, vendor_ebx, vendor_ecx, vendor_edx;
            };
            struct {
                uint32_t stepping: 4, model: 4, family: 4,
                    type: 2, ext_model: 4, ext_family: 8;
                uint8_t brand, clflush, cpu_count, apic_id;
                union {
                    uint32_t _1c;
                    struct {
                        bool sse3: 1, pclmul: 1, dtes64: 1, monitor: 1,
                        ...
                    };
                };
                union {
                    uint32_t _1d;
                    struct {
                        bool fpu: 1, vme: 1, de: 1, pse: 1, tsc: 1,
                        ...
                    };
                };
           };
           ...
        };
    } basic;

    union {
        struct cpuid_leaf raw[CPUID_GUEST_NR_EXTD];
        struct {
            ...
        };
    } extd;

    ...
};
```

This is a complicated structure, but it has two key advantages.

1. Individual leaves can be accessed as opaque information via their index (e.g. `d->arch.cpuid->basic.raw[0]`). This is is expected to be useful for

marshalling data to/from the toolstack, and responding to cpuid requests from the guest or instruction emulator.

2. Individual pieces of data can be accessed via mnemonic (e.g. `d->arch.cpuid->basic.max_leaf` or `d->arch.cpuid->extd.maxphysaddr`). This access is O(1) and can be used to replace existing logic which needs to query CPUID information in the same manner as a guest. In particular, it removes the need for Xen's CPUID handling to recurse back into itself to find `max_leaf`.

To use the proposed structure in this way, Xen must rely on the contents of the CPUID policy always being accurate.

3.2 Default policies

The existing boot-time featureset calculations will be extended to calculate maximal full CPUID policies, not just maximum featuresets. During domain creation, the appropriate policy shall be duplicated and set as the domains default policy.

- This change along will provide a usable policy for all domains, even privileged ones. The special casing for the control domain can therefore be dropped.

Further modifications to the policy occur as further construction actions are taken.

3.3 Guest CPUID handling

The two primary CPUID functions are `pv_cpuid()` and `hvm_cpuid()`, with the former containing the extra logic to cope with privileged domains. These functions serve the same purpose and have a lot of duplicated code in common. However, they also contain differences; some deliberate but some accidental because of their different logic.

There are some differences between PV and HVM guests in terms of CPUID policy, but they are all about which features are available to the guest. Given a suitable `struct cpuid_policy` to start with, there should be no difference in the handling of CPUID instructions on behalf of the guest.

A further limitation of the existing functions is that they depend on being run in `current` context. This limit can also be lifted.

Therefore, a single new function will be introduced:

```
void guest_cpuid(const struct vcpu *v, unsigned int leaf,
                 unsigned int subleaf, struct cpuid_leaf *res);
```

which will replace existing functions, and handle all aspects of completing a guest CPUID request.

- The use of a single function will remove all accidental inconsistencies between different guest types. It will also remove all differences between privileged and non-privileged domains; there is nothing special (CPUID-wise) between them.

3.4 Changes in hypercall behaviour

During domain construction, some pieces of information critical to the determination of the domains maximum acceptable CPUID policy are available right from the very start (Most notably, the HVM and HAP flags from the `XEN_DOMCTL_createdomain`).

However, some other parameters are not available at convenient points.

1. The disable flag from `XEN_DOMCTL_disable_migrate` is used to set `d->disable_migrate`, whose only purpose is to avoid the unconditional clobbering of the Invariant TSC flag. This flag cannot even be queried by the toolstack once set.

   There are other facilities which should be restricted based on whether a VM might migrate or not. (e.g. The use of LBR, whose record format is hardware specific.)

   As using `XEN_DOMCTL_disable_migrate` implies that more features should be available by default, it would be more convenient to have a migration hint along with the other flags in `XEN_DOMCTL_createdomain`, than to reintroduce the features later, as it introduces an ordering problem with respect to `XEN_DOMCTL_set_cpuid`.

2. The use of `XEN_DOMCTL_set_address_size` switches a PV guest between native (64bit) and compat (32bit) mode. The current behaviour for 32bit PV guests is to hide all 64bit-only features.

   Using this hypercall once to switch from native to compat is fairly easy to cope with, feature wise, but using the hypercall a second time to switch back causes the same ordering problems with respect to `XEN_DOMCTL_set_address_size`.

   The preferred option here is to avoid hiding 32bit features. This is more architecturally correct, as a 32bit kernel running on 64bit-capable hardware will see 64bit-only features. Other options would be to modify the API to make `XEN_DOMCTL_set_address_size` a single-shot hypercall (which, given the existing restrictions, shouldn't impact any usecase), or to require a compat flag for PV guests in `XEN_DOMCTL_createdomain`, (which would definitely impact current usecases).

Once the maximum applicable CPUID policy is determined, the toolstack may specify a preferred CPUID policy, via the `XEN_DOMCTL_set_cpuid` hypercall. Currently the toolstack-provided CPUID data is taken verbatim, and edited towards reality every time a value is requested.

To start with, there is no method for the toolstack to query a domains CPUID policy. One shall be added.

In addition, CPUID-policy auditing logic will be introduced and run as part of `XEN_DOMCTL_set_cpuid`, to ensure that the data in `struct cpuid_policy` is always accurate. This also introduces the expectation that `XEN_DOMCTL_set_cpuid` should fail if it attempts to load a policy which is outside of Xen can provide.

4 Miscellanea

4.1 Feature support statement

A problem which has surfaced several times is the difference between feature development and production use of the the feature. The current Xen/libxl interaction only has the notion of a feature being available or not, resulting in all experimental features becoming available by default to guests.

Instead, it would be better for Xen to caclulate and advertise a maximum policy which includes experimental features, and a default policy, which is what the toolstack chooses by default.

Updates to the CPUID policy will be checked against the maximum set, allowing users to explicilty opt-in to using the experimental features if they wish.