Xen Community meeting at Citrix, December 2019

Combined Notes from Christopher Clark and Daniel P. Smith

Attendees

- Daniel Smith (Apertus Solutions/TrenchBoot/OpenXT)
- Christopher Clark (OpenXT)
- Rich Persaud (OpenXT)
- James McKenzie (Bromium)
- Lars Kurth (Citrix)
- Andy Cooper (Citrix)
- George Dunlap (Citrix)
- Ian Jackson (Citrix)
- Julien Grall (AWS)
- Paul Durrant (AWS)
- Wei Liu (Microsoft)

Day One @ Xen meeting

State of XSM/Flask

Proposal

- Would like to see XSM/Flask become the default access control mechanism for Xen
- An audit of security hooks may be needed
    - Ensure all subject/object/predicates are covered
- Default policy should not be hard coded/always present
    - IOW, dummy policy should be extrapolated out to as a proper policy file

Discussion

- XSM/Flask needs refactoring
    - AVC user space tool
    - Performance: while there is an expected overhead the OpenXT community uses it without user noticeable performance degradation but to date there is no known benchmark to quantify this overhead

- Provide non-Flask hypervisor developers a means to provide appropriate Flask patches for the hypervisor capabilities that they are implementing

Specifically requested:

- A Developer who adds a XSM hook should not need to understand anything beyond the dummy policy
    - After making XSM/Flask the default access control mechanism then XSM should provide a consistent behavior which can be one of the following: 1. deny access to the new hook and break, 2. fail to apply, or 3. fail to build (in which case XSM would not be testable by default)
- Correctness of the dummy policy should be reviewable by non-Flask-specialist Xen Security Team
- XSM/Flask default policy should not be less secure than the current highly-privileged Dom0 security model

DomB-mode of dom0less

Discussion about the current implementation of dom0 launch and the hardware domain.

- During dom0 creation, and prior to starting it, the hypervisor issues dom0 all possible permissions over i/o ports, i/o mem and IRQs (see: `dom0_setup_permissions`). The hypervisor then reduces these permissions by masking out with a series of defined bitmasks to restrict it.
- A hardware domain (that is not dom0) is identified by matching a domid specified on the hypervisor command line
- When the hardware domain is launched, permissions are removed from dom0 and added to the hardware domain when the hardware domain is started (see: `late_hwdom_init`).

Objectives

- Extend dom0less support to x86 platforms
    - boot domains are to be configured and launched by a single initial domain (domB) that exits once the setup work is completed
- Enable flexible implementation of cryptographic measurement of the set of domains launched at system boot
- Support the use case of implementing a disaggregated vTPM with isolation from the control domain
- Permissions and access of **all** domains should be dictated by domain label from XSM/Flask policy
- Optional: the boot domain image could optionally be bundled within the Xen binary in a similar manner as the initramfs can be bundled or not with Linux kernel
- Want to be able to replace the domB without rebuilding the hypervisor binary.

- Add a new KCONFIG for switching between domB implementation of platform launch and the existing dom0 implementation.

Discussion

Requested: an in-Xen-tree reference domB implementation

Note: the hypervisor interface supports starting a domain with a specified domid. This is not made available by the libxl toolstack.

Xen has logic on permission delegation:

- in order for a VM to be able to delegate a permission to another, it itself must have that permission.
    - **ACTION**: change this to be implemented as a XSM hook, so that policy can choose to override this constraint, while preserving it (ie. the existing behaviour) as the default.

The domain id to assign to domB: should not be zero.

- Recommendation is to use a new fixed domain id allocated from the reserved range.
    - See DOMID_IDLE, DOMID_INVALID, etc in <xen.h>

The is_hardware_domain predicate

- uses within Xen not necessarily all consistent?
- convert this to a XSM hook?
    - ***to be determined***: performance impact since hits the avc?

Need to not shut down the platform when domB exits

- ie. distinguish domB from the hardware domain

Since the hypervisor ABI is unstable, specifically the domain building hypercalls, will need to use the Xen toolstack:

- so libxc/libxl is the right interface for initial domB domain logic to use
    - otherwise problematic when Xen hypervisor version is changed
    - in the near term, this mandates the use of Linux + toolstack

Python bindings as an option was mentioned

Decision:

use full Linux within domB as starting point

- unikraft discussed, not selected: is not deployed in production and want to use mature, QA'd and externally maintained components
- 32MB size for the kernel queried: proposers have no issue with that size

Request made for a script interpreter in an example domB, with scripts to start dom0 with a given set of permissions

- aim is to make domB usable for other people's use cases and widen adoption, help other people with what we build for domB

Brief discussion of "single VM" Xen

Day Two @ Xen meeting

Terminology for this text:

- highest privilege hypervisor (L0) is “at the top”
- guests are lower, decreasing in height (L1, then L2 etc)

Naming Method Proposal #1: Client-visible uuids for names

NB: This first naming method proposal was discussed and then superseded by an alternative, see below. Discussion of this first method is included here for context and the archive.

UUID Semantics:

Three options for what a UUID represents were presented:

1. A running VM instance
2. A running VM instance but the UUID is never recycled
3. A VM associated with a thing on disk

Consensus was for option 2: a running VM instance but the UUID is never recycled.

When a VM is migrated, its UUID must change.

When importing a VM from another machine, allow override of the default-assigned uuid.

- An important case to consider is the localhost migration: uuids must differ.

A VM, a template and snapshot must all have UUIDs within the same address space.

- A useful construct is to have a VM (and a UUID) with no resources assigned, that can be resolved to a running VM's uuid.

Tab-completion of UUIDs for CLI tools (as xapi currently does) is essential for usable human interface.

Migration is either invisible to the guest or needs to be tested continuously to detect and prevent breakage.

Discussion covered a method of making the migration event visible and simple for a guest to act upon, by setting a bit in the well-known page shared between the guest and Xen to indicate the event, and triggering an interrupt to the Xen platform device => triggers rediscovery logic.

Strong concern raised that migration must not require the knowledge or cooperation or correct execution of an algorithm within the guest. Buggy front ends currently decrease the reliability of VM migration.

Want an option for client-specific translation of id state exposed to guests so that they can be shielded from the real uuid in use. This is to avoid a future repeat of the current difficulties experienced with guests being aware of their domid.

Sub-topic: nesting

The L0 hypervisor (closest to the hardware) should not be required to maintain a name translation. Each level of nested hypervisor performs name translation for its guests, which includes translation of a local name for a guest-hypervisor.

When talking up, a particular name gets expanded to the list of names being queried: behaves like a VLAN tag.

- Names are not of fixed length. All hypervisors name their direct guests.

Clients always use relative path names rather than absolute path names.

For data transfer between layers, once you've done a memcpy, doing another of the same data is essentially free (for performance cost) while it remains in the cache.
For measuring performance, compare against CPUID as reference benchmark.

Naming Method Proposal #2: Externally-connected ports and implicit destinations

This is a superseding proposal to the above-described use of guest-exposed uuid names.

Approach

Add the following two new concepts to Argo:

Concept 1: add “implicit destinations” where messages can be sent (via the sendv op) with only a specified <source Argo port>, leaving unset both the <destination domid> and <destination Argo port>. The unset destination values are then filled in by the hypervisor by performing an internal lookup from (src domid, src Argo port) to obtain a fixed (dst domid, dst Argo port) for the message destination.

Concept 2: allow the toolstack to create and manage the entries in its hypervisor’s “implicit destinations” table. This enables the toolstack to perform "patch-cable"-like connection of ports between guests with its hypervisor, and can be done external to the VMs.
ie. the L<n> toolstack connects implicit Argo ports between the guests of the L<n> hypervisor.

With both of the above in place, a VM can then send messages that specify only the client port, and the hypervisor will complete the destination VM and destination port, enabling a VM to communicate with an endpoint determined by the toolstack. This enables the use of well-known client port numbers -- ie. agreed between VM and its local toolstack that manages it -- for services eg. "my storage".

A special "destination domid" in the implicit destination table indicates "up to the next hypervisor", for sending messages upwards when nesting.

To complete the nesting communication path, a hypervisor needs a method of receiving messages from its parent and mapping them to its guests: A per-CPU receive buffer where messages will be delivered into.

A new XSM/Flask control can be added to constrain a guest to only be allowed to send to “implicit destinations”, which should be useful for enabling controlled guest access to Argo communication without requiring a full port-based firewall implementation.

Copyright