VirtIO-Argo Development: Phase 1

This is a proposal for initial development towards a Linux VirtIO Argo transport device driver and the back-end platform support software to connect it, to implement some of the prerequisite critical pieces that are suitable for incorporation into upstream projects, and provide demonstration of the viability of this path.

Project: XSM Firewall and Front-end Interface Points

Destinations

- XSM policy controls over Argo connections: to upstream Xen
- Documentation of XSM policy controls: to upstream Xen
- XSM test cases: to be determined: potentially Xen Project or meta-virtualization
- Prototype demonstration over an instrumented existing VirtIO transport: to an OpenXT branch

Rationale

The motivation for this proposal is that it:

- enables development of the XSM Argo firewall that can be upstreamed to the Xen Community ahead of the further phases of implementation of the VirtIO-Argo driver components
    - The XSM Argo firewall should be able to be used without the remaining VirtIO-Argo components, using the existing Linux Argo device driver and upstream Xen XSM policy extended to support the new firewall
- enables a static XSM/Flask policy configuration to govern connectivity between Argo <domain, port> endpoints according to the XSM labels that have been issued to the VMs at either end
- allows the structure of the front-end VirtIO-Argo transport driver to be practically explored
- enables the hook points of the Argo access control checks within the hypervisor to be identified and validated
- exercises the new XSM Argo firewall replacement: the XSM policy will govern allowed Argo communication connectivity, which will be adhered to by the client developed for this phase, even if the shared-memory implementation of the transport actually used isn't being policed by XSM as Argo communication will be
- enables exploration of the XSM policy tooling for representing a static XSM firewall to govern Argo connections
- shows a running system, without requiring the full VirtIO-Argo implementation
- provides reference code for the backend driver work of later development phases to be analysed and developed against

Plan

Demonstrate the integration points for the VirtIO-Argo frontend driver and exercise the Argo XSM firewall

- Establish an initial development environment: enable a VirtIO virtual device on Xen, using one of the existing VirtIO transport drivers in the guest
    - use that existing transport - eg. virtio-mmio or virtio-pci - over shared memory, to be enabled via temporary software modifications if necessary -- to provide working driver front and backends for a functional VirtIO virtual device on Xen. eg. virtio block or virtio net.
    - note that this does not preclude also using the existing Xen device drivers in the same guest at the same time, for ease of enabling a running guest.
- Instrument the guest VirtIO transport device driver (ie. virtio-mmio or virtio-pci) to insert Argo hypercall operations at the points where Argo operations will be required with the new virtio-argo transport.
    - ie. where the VirtIO split driver rings are established, for the virtio-argo transport, the memory allocated to the VirtIO "used ring" is also registered with the hypervisor as an Argo ring, with permission for a backend domain (eg. dom0, or a hardware domain or a driver domain) to map it.
    - allocate a kernel memory buffer to be used as a buffer for incoming driver data read operations, and register it as an Argo ring for the backend domain to send to.
        - This buffer will actually not be used in the initial prototype where Argo is not used for the actual transport for driver data, but ring registration will exercise the Argo XSM firewall at the correct point for when Argo is enabled as the transport.
        - This will enable an XSM access control check to be performed
            - AVC log messages will indicate if a refusal occurs
            - If the Argo register operations are refused, the VirtIO ring setup can be aborted and driver initialization for the device cancelled, which will match the control flow when MAC policy denies communication for a guest.

        - when data is sent:
            - after the descriptor is written to the VirtIO split-driver ring:
                - phase 1: add a log message to indicate Argo sendv would occur
                - phase 2: add an Argo sendv operation, which will require the backend domain to have registered a ring to send to
- Development recommended to be performed with Xen and a basic meta-virtualization OE development environment, rather than a full OpenXT build, as no OpenXT-specific functionality should be required
- Test cases needed to validate the XSM policy control over Argo communication
    - Option to evaluate: launching Xen on QEMU, with XSM enabled and guests containing test cases
- New documentation to be written to describe:
    - The method for configuration of the XSM Argo firewall, so that system firewall policies can be written
    - Documentation should be suitable for submission to the Xen Community, in conjunction with the XSM Argo firewall implementation

Project: new Argo Linux driver and userspace

Destinations

- Initial: OpenXT
- Subsequent: Xen Project; to be followed onwards to the Linux kernel depending upon Xen Community participation

Rationale

An Argo Linux device driver is necessary for the domain running the platform VirtIO-Argo software that implements the virtual device backends, so that the userspace process that implements the VirtIO device - typically qemu - can invoke the kernel to interact with Argo for interdomain communication with the remote VirtIO-Argo transport driver in the guest domain.

The current OpenXT Argo Linux device driver is not a suitable codebase for further development, whereas the uXen Linux v4v drivers are high quality and simpler, and suitable for porting to Argo which has a similar interface to v4v.

The ultimate destination for the device driver should be the upstream Linux kernel, but developing a driver suite to that standard could be pursued as a separate effort to the initial development.

Plan

The current OpenXT Argo Linux device driver, and the userspace library software that uses it, is derived from the original XenClient v4v Linux software. There is consensus in the OpenXT community that it needs to be replaced with a new implementation, and discussion notes on this have been recorded on the OpenXT wiki.

The uXen hypervisor developed by HP / Bromium implements a more recent v4v hypercall and data transport than the original in OpenXT, and the Open Source Linux guest support software, available at the public uXen github repository, and in a zip archive at: bromium.com/opensource. There is a recipe for building them in the meta-virtualization layer: https://git.yoctoproject.org/cgit/cgit.cgi/meta-virtualization/tree/recipes-extended/uxen?h=dunfell

The uXen Linux devices drivers for v4v should be ported to Argo on Xen as a new foundation for development and use of Argo on Linux on Xen. The abstractions presented to userspace by the new v4v drivers are simpler than those of the current OpenXT Argo driver – eg. they do not implement the “stream” connection type – so changes will be required in the userspace software too.

The initial scope for the new ported driver is narrower than the existing OpenXT Argo device driver, which supports a general interdomain transport (eg. enabling DBUS between domains): the focus for the initial port will be to implement what is necessary to support the communication required for the Virtio-Argo device driver use case.

Note that the uXen v4v driver re-uses the (upstream Linux) VSock AF: it registers a new socket-class using AF_VSOCK as the address-family identifier, but it is not a VSock transport driver. The uXen v4v driver has a different interface than VSock and sits under the AF_VSOCK address family id. This would be a blocker for any upstreaming effort of this, but may be acceptable for enabling a PoC of the VirtIO-Argo transport project. If this driver were not to add a new AF_ARGO, implementing a socket class could avoid some restrictions on the VSock transport protocols.

To be investigated: the more recent HyperV implementation of VSock (VMCI), which follows the pattern of the hypervisor-specific VSock transport driver implementation registering with vsock_core, which may allow for a new Argo VSock implementation where ring management, etc. is handled in a separate driver that reuses the VSock core. Note that `CONFIG_VSOCKETS` does not have a dependency on `CONFIG_NET`, which makes VSock potentially usable on systems that are not networking-enabled.

An alternative to this interaction with VSock or having to make a case for a new socket type would be to implement a new network device driver. A network driver will support use of familiar networking abstractions and existing networking tools over Argo between domains. Since some systems that need to support Argo use kernels that are configured without networking - ie. `CONFIG_NET=n` - an additional interface to access

Argo functionality can then be provided via a char device. The implementation for these multiple interfaces to userspace should structure the code so that common functionality, such as ring management logic, is shared between the separate drivers that provide the userspace-accessible interfaces. The uXen v4v drivers, provide suitable references for this structure.

To be investigated: whether the IOREQ structure is to be adopted for the VirtIO-Argo back end, and whether that means that userspace processes for device support do not need to know about kernel interaction with Argo. Need to verify that MAC denials that are indicated by appropriate error code, both for new and existing connections, are handled correctly in userspace.

There are three core uXen v4v drivers of most relevance:

- uxenv4vlib
  - the common library of v4v driver functions
  - eg. ring registration, interrupt handling, suspend and resume processing
- v4vchar : the character device interface to v4v
  - registers a character device, that implements file operations:
    - write
    - poll
    - flush
    - fsync
  - has a task queue for processing incoming data that is scheduled whenever a v4v IRQ occurs
  - processes received messages from an Argo ring into a message queue within the driver to support sequential processing by userspace
- v4vvsock : the vsock interface to v4v
  - note that the method of registering the vsock interface in this v4v driver that will affect the acceptability of this driver into upstream Linux as it currently stands, and will need to be resolved before that path should be pursued

A library of tests to exercise the new drivers will be essential, to gain confidence in their correctness and suitability for use in OpenXT and elsewhere.

Discussion notes from Topic Call, January 2021

- When using VSock as a kernel interface for interdomain transport, it is not necessarily simple to map from an address identifier to a remote domain: other hypervisor implementations on VSock use pre-known identifiers
- Discussed whether a Xen domain is expected to be able to know its own domain id: guidance was given that it is reasonable to assume and require it.
- VSock will likely not be the interface to use for communicating from userspace (eg. QEMU or other emulator) to a domain kernel in support of the VirtIO-Argo transport backend; however:
- Forward direction: the Argo Linux driver shall be built modularly, similar to the uXen v4v driver, with a library core (a misc driver with ring and interrupt handling logic, etc) plus separate drivers that export different interfaces to userspace for access - ie. VSock can be one and a separate interface to be used for supporting the needs of userspace components for VirtIO-Argo.

Project: unification of the v4v and Argo interfaces

Destination

- Initial: OpenXT
- Subsequent: HP/Bromium uXen and the Xen Project, and the Linux kernel

Rationale

- Enable common guest software between hypervisors, with compatibility of testing driver software on each platform
- Enable communication across nesting levels in systems with multiple hypervisors - see the HAT architecture

Argo in Xen, and v4v in uXen have a common origin, beginning in the original v2v and then the subsequent v4v developed for XenClient, and still retain a similar structure. Argo was developed for inclusion into the Xen Project hypervisor, adhering to the requirements of that community, and uXen's v4v implementation was one of the source code references that informed the development of Argo.

Plan

In the current public Open Source implementations (as of December 2020), the basic primitives in the Xen Argo hypercall interface and the uXen v4v hypercall interface are similar enough that a Linux device driver could abstract the differences and provide a basic common interface to userspace, so that the same guest application software could work on a system that uses either v4v or Argo. The uXen v4v interface has more operations than Argo, and some investigation into the use cases that motivated the new operations, and whether those are applicable to Xen or OpenXT could be warranted.

There is an intentional enforced policy limitation within uXen that contrains v4v communication between peer domains: the use case for the uXen hypervisor in the HP/Bromium product does not require allowing that, so it is disabled. Argo will instead be governed by the XSM Argo firewall. Some communication policy configuration mechanism will be needed for uXen to allow a more liberal v4v policy for communication between domains, to be configured by the uXen guest mangement software.

Going further than providing a common v4v and Argo interface to userspace software, and implementing a single common hypervisor interface that incorporates the functions of both, into modified Xen and uXen hypervisors respectively, would provide a good platform for development of nested interdomain communication on a system running both Xen and uXen. With the VirtIO-Argo device drivers, that should allow for a system structure that supports hosting virtual device implementations within uXen guests on a Xen system.

Project: VirtIO-Argo with uXen and PX hypervisors

Destination

- HP/Bromium uXen
- the new PX hypervisor project

Plan

The VirtIO-Argo transport enables the use of Argo for interdomain communication between the guest domain's virtual devices and the external platform software that provides the device implementation.

With a uXen hypervisor that implements Argo, the VirtIO-Argo transport driver within the guest could be connected to the uXen device model that supports the guest, to provide the virtual device implementation. Alternatively, it could be plumbed to connect to an implementation within another uXen guest VM, with suitable modifications to the uXen tooling.

On a system with a PX hypervisor, it would have the opportunity to apply system-wide Mandatory Access Control to communication paths between guest VMs of uXen and Xen hypervisors using Argo, which would govern all virtual devices using the VirtIO-Argo transport.

The minutes from a meeting in Cambridge in December 2019 were published to xen-devel:
Notes from December 2019 Xen F2F in Cambridge
These include a section "Naming Method Proposal #2: Externally-connected ports and implicit destinations" that describe the addition of two new concepts to Argo to support establishing communication between Argo endpoints:

- 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 (`<source domid>, <source 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.

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.

Project: VirtIO-Argo transport: Virtual Device Discovery

Destinations

- VirtIO-Argo transport driver
- Xen and subsequently other Argo-enabled hypervisors

Rationale

Each VirtIO transport device driver is responsible for discovering and surfacing the virtual devices that the platform has made available to the guest. To do so, the transport driver needs a means of interrogating the platform to discover the devices.

Discussion notes from Topic Call, January 2021

- A range of Argo ports are to be reserved and registered as well-known addresses for access to guest services

- A platform toolstack or a suitably authorized guest domain can program the destination table in the hypervisor to direct service requests to their available service endpoints
- A guest VM can talk to the VirtIO-Argo device discovery service registered for it by communicating to the well-known port for it.

Plan

See the above: “Project: VirtIO-Argo with uXen and PX hypervisors” where the planned extensions to Argo’s addressing are described, which can then be used by the VirtIO-Argo transport driver and the toolstacks that support it.

Project: IOREQ for VirtIO-Argo

Destinations

- Initial: OpenXT
- Subsequent: Xen Project; potentially Qemu

Rationale

The intended destination for the VirtIO-Argo work is to upstream communities and the current VirtIO activity in the Xen Project is oriented around enabling VirtIO on Xen on Arm, including a port of the Xen on x86 support for IOREQ servers to Arm. This is to support implementation of userspace software in a driver domain in support of guest virtual devices.

Plan

The Xen Community development mailing list has a relevant thread for VirtIO on Xen, with interest from the Xen on Arm community, as well as at least one Xen x86 maintainer, which considers using Xen’s IOREQs to support VirtIO data transport:

https://lists.archive.carbon60.com/xen/devel/592351#592351

The work to add IOREQ support to Arm is making progress at the moment, with version 3 of the series posted in late November 2020, specifically focussed on enabling VirtIO-MMIO transport:

https://lists.xenproject.org/archives/html/xen-devel/2020-11/msg02159.html

[update: v6 is now posted here:
https://lists.xenproject.org/archives/html/xen-devel/2021-01/msg02403.html]

The VirtIO-Argo transport development work needs to take this work in the upstream community into account; eg. consider planning for integration with it as an optional alternative mode of transport, to co-exist with other transport options on Xen.

Note that the existing IOREQ implementation uses event channels, as does Argo currently, though see the separate project on this page about interrupt delivery for a potential change to Argo away from that.

To be investigated: XSM policy controls over IOREQ communication and how to correlate these with XSM controls over Argo to ensure that the system policy is able to express the firewall constraints intended.

Update 29 Dec 2020:

Feedback from the Xen and OpenXT Communities has been that IOREQs will not be required for building an Argo transport for VirtIO. The IOREQ infrastructure is not a current fit for the Argo transport, with the IOREQ architecture prescribing the use of emulation infrastructure, foreign mapping privilege, event channels, etc.

Discussion notes from Topic Call, January 2021

Instead of the use of foreign mappings currently used by privileged virtual machines that perform emulation on behalf of other guests in the IOREQ architecture, a new Device Model Operation (DMOP) hypercall could perform fetches of ranges of guest memory on behalf of the emulating guest, to provide an alternative data path for supporting virtual DMA data transfers. This same DMOP could also have an immediate practical application elsewhere for improving the efficiency of VM introspection, lowering the number of hypercalls and decreasing complexity for retrieval of small numbers of bytes of guest memory.

A DMOP that provides this structure for access, with the remote data being retrieved and transferred by the hypervisor rather than accessed directly via a foreign memory mapping, is an alternative structure to that provided by Argo for hypervisor-mediated data exchange (HMX). In contrast to Argo where communicating peer relationships with can be established with ring registration on either side, the proposed new DMOP mechanism is suitable for a VM relationship where the receiving side is privileged and providing platform services in support of the guest that it accesses.

Project: Port the v4v Windows device driver to Argo

Destination

- Xen Project

The Windows v4v driver should be ported to Argo, which will enable it use on modern Xen and OpenXT and allow for it to be upstreamed to the Xen community. It can further be used to enable development of support for VirtIO-Argo virtual device backends in non-Linux domains, and in support of a VirtIO-Argo transport driver for Windows, to enable VirtIO virtual devices on Windows via Argo.

Community Engagement

Items from the November OpenXT Community Call:

- Chris Rogers: working on the uprev of Xen to 4.14, including development to enable upstreaming items from the OpenXT libxl patchqueue (originally developed by Chris)
    - to develop new XSM hooks to cover Argo (domain, port) pairs, in conjunction with the domain label, to allow replacement of the firewall for initial upstream use of Argo.
    - will acted upon at domain creation (VM start) and enable removal of OpenXT libxl state machine patches that currently support the Haskell toolstack's firewall configuration at VM start.
- Agreement to pursue this work then prompted discussion about moving OpenXT to use the upstream Xen XSM policy
- Jason Andryuk: published a script for performing comparisons on XSM Flask policies
    - https://gist.github.com/jandryuk/685b15055041e70a789edd22a08e43e8

Development proposal

Example grouping of development of aspects of the project, to be distributed among the community according on the availability of project participants:

- New XSM policy controls over Argo communication (including ports) to replace the OpenXT Argo firewall and the libxl state machine logic to configure the firewall at the correct points in the VM lifecycle
    - implementation of the Xen XSM Argo hooks, suitable for upstream Xen, and governing (domain, port) pairs
    - tooling to produce a XSM policy with a firewall configuration
- development of new Linux Argo device drivers starting from a port of the uXen v4v drivers to Argo
- QEMU backend (qemu, etc) modifications to register Argo rings at suitable points for the frontend VirtIO transport driver to attempt to send to
- bringup of an initial VirtIO device driver on current Xen, using an existing VirtIO transport (eg. virtio-mmio, as referenced by current VirtIO activity in the Xen Project community), to build a development reference platform
    - implementation of the VirtIO frontend transport driver modifications to insert Argo operations
- Documentation for upstreaming to the Xen Project of the new XSM policy controls to firewall Argo communication
    - Input can be solicited from members of the Xen Community with perspectives on Argo beyond the OpenXT Community
        - eg. at Xilinx, Arm, HP, Oracle

Project: Hypervisor-agnostic Hypervisor Interface

Destination

- Xen, and then followed by all other hypervisors

Rationale

VirtIO is widely adopted across many hypervisors and guest operating systems. VirtIO standardization is managed by OASIS.

It could be easier for VirtIO community to accept and standardize a new transport interface if it were not tied to an interface specific to a particular hypervisor, but rather a hypervisor-agnostic interface that any given hypervisor could elect to implement. The VirtIO-Argo transport driver can then be built upon this interface, and make the same transport device driver be compatible with multiple hypervisors.

The current Argo hypervisor interface presented by the Xen hypervisor, on both x86 and Arm hardware, and for HVM and PV guests, is implemented as a hypercall that supports the Argo operations.

While hypervisor calls have the advantage of being generally supportable by hypervisors and so the guest driver implementation can be non-arch specific, it is specific to the Xen hypervisor due to the selection of hypercall number in use and the argument interface for the Argo operations.

On Arm, SMCCC standardization presents an opportunity to reserve a range of hypercalls that use the 'hvc' instruction for Argo, as a Standardized Hypervisor Service Call, to make Argo available via a hypercall with that identifier on hypervisors that elect to implement it.

On x86, Intel's vmcall and AMD's vmmcall instructions are available for hypercalls from HVM guest VMs, but without the vendors supporting a standardized central registry of operations. A cross-hypervisor interface for Argo using these instructions could be one option for investigation. An alternative interface could be built using MSRs instead, given the expectation that every x86 hypervisor must have some logic to trap and

handle some of those, and that every guest OS has the functions to read and write MSRs with non-vendor-specific instructions. Any hypervisor could then elect to implement that same interface.

For a hypervisor-agnostic MSR interface, a range of identifiers will need to be reserved to guarantee that they will not be used by current and future processors. Intel guarantees that a range of MSRs (0x40000000-0x400000FF) are always be invalid on bare metal, and software developers have started using this range to add virtualization-specific MSRs. (See the defines related to HV_X64_MSR_* in Xen for some of the HyperV ones.)

Other hardware architectures with hypervisors that support paravirtualized guest OSes, or guest kernels that are binary translated, may also not have access to the vmcall/vmmcall instructions, as per PV guests on Xen. While x86 may present a different means of communicating with the hypervisor (eg. via an MSR interface) this may not be generally true across all architectures. It is attractive to have a hypervisor interface that is not too architecture-specific, to avoid incurring complications when porting software to new architectures.

On Arm, the closest equivalent to an MSR interface would be use of System Registers, but this would not allow for reservation of a valid range of IDs that could be used. The hypercall interface is consequently attractive for Arm, and there is not a clear alternative (other than MMIO or PCI) that could be used to support PV or binary translated guests.

On x86, the performance cost of using MSRs forcing a direct VMEXIT could be potentially expensive and if so, would be incurred on all hypervisors using this interface; the motivation for potentially accepting this cost is that standardization of the interface across multiple hypervisors may enable Argo to be proposed as an hypervisor-agnostic mediated data exchange transport, to ease its acceptance into VirtIO.

From considering the above, a candidate path forwards towards building a proposal to discuss with the VirtIO community:

- For x86, design an interface suitable for hypervisor-agnostic mediated data exchange using MSRs
    - with reference to Xen as the first hypervisor and a second Open Source hypervisor to demonstrate and support the cross-hypervisor case
    - also examine whether a vmcall/vmmcall interface could be feasible to standardize across hypervisors, with the same objective: allow a transport driver to be used unmodified across hypervisors
- For Arm, examine the SMCCC standard and determine what a hypercall implementation using one or more reserved identifiers will look like
- For the design of the structure of a new Linux VirtIO-Argo transport device driver, consider how best to define and manage differences in hypervisor interface mechanism (ie. hypercall, MSR) across architectures (or hypervisors) within the driver code
- Review the interfaces of each Argo operation with respect to the needs of the hypervisor interface mechanisms and architecture requirements

It may be the case that for the Arm architecture, in contrast to x86, the hypervisor-agnostic path is simpler – due to the ability to use a transport driver using the known hypercall mechanism, in conjunction with new reserved identifiers to be obtained via SMCCC, across multiple hypervisors – and that this may be sufficient to demonstrate cross-hypervisor compatibility for supporting the proposal to admit the new transport driver into VirtIO. A corresponding design to be able to support transport driver compatibility on multiple x86 hypervisors still remains important: a MSR-based implementation enables a phase one implementation, and if resources are available can be followed by implementation of specialized per-hypervisor adaptation, which could allow hypercall-based interfaces to be used where they are preferred.

For evaluation:

- performance
- identity / label / policy interop
- nested hypervisor interop
- HMX per-hypervisor cost: hypervisor changes, hypervisor requirements, guest drivers
- use of OVMF firmware to provide a common protocol for guest use that would avoid the need for a single uniform hypervisor interface

Credits

- Original suggestion and x86 hypercall interface perspective by Roger Pau Monné of the Xen Community
- OVMF firmware suggestion by Daniel Smith of the OpenXT and Xen Communities
- Interop evaluation suggestion by Rich Persaud of the OpenXT and Xen Communities
- Arm platform perspective review and SMCCC hypercall standardization note by Julien Grall of the Xen Community

Discussion notes from Topic Call, January 2021

- hypercalls: difficult for portable across hypervisors (at least on x86)
- concern re: MSRs: some hypervisors do not intercept MSR accesses at all
    - could provoke unexpected behaviour on nested hypervisors
- performance of the selected interface mechanism will be critical, whichever selected
- alternative options to MSRs exist and are used by other hypervisors
    - HP/Bromium AX uses CPUIDs
    - Microsoft Hyper-V uses EPT faults
- Arm context: hypercalls may be acceptable on Arm across hypervisors

- standard way to to do it; able to implement Argo in either firmware or hypervisor; difference in access instruction
- not an option for PV-only hypervisors without hypercalls

Proposal from Topic Call, January 2021

Since it is unlikely that a single mechanism will ever be viable for all hypervisors to support, plan instead to *allow multiple mechanisms to be made available* and then *enable the guest device driver to probe* for what mechanisms the hypervisor it is inteacting with can support

- A hypervisor can implement as many mechanisms as is feasible for it
- A guest can perform selection between the presented available mechanisms reported by the hypervisor
- Preference for mechanisms that are close to platform architecture (ie. well-defined on it)
- Ensure that the discovery mechanism is forward-extensible for new mechanisms later

Plan

Documention, community engagement, implementation, measurement, testing, review, code submission and interation.

Project: Argo interrupt delivery via native mechanism

Destination

- Xen and guest Argo device drivers

Rationale

Argo currently signals notifications to guest VMs via the Xen Event Channel mechanism. The in-guest software that implements the Xen Event channel handling is not always present and should not be made a requirement for supporting the VirtIO-Argo transport driver.

To be investigated: delivering interrupts using a native mechanism eg. MSI delivery by using a destination APIC ID, vector, delivery mode and trigger mode.

Credits

- Suggested by Roger Pau Monné of the Xen Community
- James McKenzie of Bromium/HP has referenced the interrupt mechanism used for signalling in v4v

Discussion notes from Topic Call, January 2021

- MSIs: are ok for guests that support a local APIC
- Hypervisors developed after Xen learned from Xen's experience: register a vector callback
  - MSI is not necessary
  - sometimes hardware sets bits
  - likely architecture-specific; could be hypervisor-agnostic on the same architecture
- Vector approach is right; some OSes may need help since allocation of vectors can be hard
  - eg. an ACPI-type thing or device can assist in communicating a vector to the OS
  - want: OS to register a vector and the driver to communicate the vector to use to the hypervisor
- Want to avoid the extra level of multiplexing when Argo rings are layered on top of Event Channels
- Vector-per-ring or Vector-per-CPU? : Vector-per-CPU is preferable
  - aim: avoid building muxing policy into the vector allocation logic
- Scalability, interface design consideration/requirement: Allow expansion
  - one vector per CPU => multiple vectors per CPU
    - eg. able to assign different priority for different rings: will need different vectors to make notifications work correctly
    - to investigate: specify the vector for every ring when registered and allow same vector for multiple rings (fwds-compatible)

Proposal from Topic Call, January 2021

Vector registration.

Plan

Implement, measure, test, review, submit upstream.

Related Development Project: Hypervisor-mediated access primitive for IOREQ transfers, VM Introspection

See the above Discussion notes on the Project: IOREQ for VirtIO-Argo.

Destination

- Xen Project

Investigate a design for a new Device Model hypercall operation (DMOP) to provide hypervisor transfers of requested ranges of remote guest memory in support of privileged VM services. This project needs to align with new design and development efforts for the hypervisor to provide virtual IOMMUs since this will affect the data paths involved in I/O emulation.

Rationale

This is a separate project from the work to provide an Argo transport for VirtIO, arising out of the discussion of the IOREQ topic in the January 2021 conference call.

- Performance enhancement for VM introspection
- Introducing HMX properties to data transfers in the IOREQ system architecture

Plan

Interest of the Xen hypervisor VM introspection community to be explored.

Credits

This idea was proposed by Andy Cooper of Citrix in discussion with Daniel Smith of Apertus Solutions exploring how to enable HMX primitives in the IOREQ architecture on the Topic Call in January 2021.

Research Proposal: Build and evaluate an asynchronous send primitive

Rationale

The existing Argo send primitive is a simple, synchronous hypercall op for transmission of data into a remote receiver ring. For some use cases, such as supporting guest framebuffer for virtio-wayland or other VirtIO drivers that currently explicitly use shared memory regions, an alternative asynchronous delivery primitive may enable use of the Argo transport in use cases that cannot currently be met with the synchronous send primitive.

An objective for the project is to explore whether higher throughput, lower latency or higher efficiency can be achieved. It is expected that performance characteristics will differ on different hardware architectures.

Research into this should consider that understanding hardware behaviour and primitives that could be made available on new processor architectures as an important part of the work.

- Different capabilities may be available on modern x86 architecture (though not necessarily present on all classes of CPUs)
    - Extended page table attributes and virtual functions for operating on pre-programmed address spaces can be investigated for support of unidirectional or fast transport of bulk data between domains
- New processor architectures (eg. RISC-V) on Open Source soft-CPUs enable exploration of the design of new primitives for efficient support of Hypervisor-Mediated data eXchange

Credits

- Interest in asynchronous send raised by Jean-Philippe Ouellet of the Qubes Community
- Related recent research into high performance interdomain transports has been identified by Rich Persaud of OpenXT
    - See the December 2020 Review feedback section below for links.

Discussion notes from Topic Call, January 2021

Topic briefly mentioned but currently a lower priority than other development items discussed.

Comparison of VM/guest Argo interface options

The Argo hypervisor interface to virtual machines is a public Xen interface and narrow, with only a small number of operations - currently four, though likely to increase by a few with v4v integration.

There are also multiple Argo interfaces internal to domains:

- between drivers within the Linux Operating System kernel
- between drivers within the Windows Operating System kernel
- between the Operating System kernel and Argo tools in userspace
    - eg. runtime firewall configuration
- between the Operating System kernel and Argo libraries in userspace
    - eg. device nodes to support data stream abstractions
- between Argo libraries and client applications
    - eg. for interdomain DBUS communication

There are several options for the interface between the Linux kernel and userspace, to be reconciled as part of the work to take an Argo device driver into the upstream mainline Linux kernel.

Interface Name	Upstream Considerations	Pro properties	Con properties	Frontend consumers	Backend consumers
VirtIO-Argo transport driver	• Destination: mainline Linux kernel, via the Xen community • Argo will need to become security-supported in Xen	• Enables use of existing mainline Linux VirtIO virtual device drivers, expanding the available virtual device types without requiring development or maintenance • Mandatory Access Control enforcement	VirtIO interfaces are developed by a standards committee (OASIS), so can move slower	Mainline Linux VirtIO device drivers	None
Existing OpenXT Linux Argo device driver	• Not acceptable	• Tested and functional over an extended period	Implementation is unsuitable for further development	• Existing libargo • Interposer library	• Interdomain DBUS • Argo firewall configuration tool • libxl state machine for VM lifecycle, indirectly
Bromium uXen v4v device drivers, straight port to Argo: VSock interface	• Already present in uXen software distribution • For mainline Linux, or Xen: Unacceptable override of AF_VSOCK for a non-vsock interface, will require resolution	• Strong code foundation for future development • Simpler interface and implementation than the current OpenXT driver	AF_VSOCK use needs resolving	• Potentially OpenXT service VMs; though VirtIO-Argo may be better suited • Ported uXen v4v storage and network drivers • Ported libargo	OpenXT platform VMs
Bromium uXen v4v device drivers, port to Argo and expose char device	char device could be a challenge to justify	Functional even when kernel is configured without networking support	Non-standard char IOCTL interface rather than a networking standard	• Ported uXen v4v storage and network drivers • Ported libargo	• OpenXT platform VMs • Ported libargo • Ported interposer • Potentially a backend to VirtIO-Argo - to be determined
Bromium uXen v4v device drivers, port to Argo and expose network device	network device interface needs to be evaluated for upstream acceptability	Could avoid exposing Argo to userspace at all	• Requires kernel configuration that includes networking • Limited interface if networking-only, as does not expose all wanted /needed functions	• Ported uXen v4v storage and network drivers • Ported libargo	• OpenXT platform VMs • Ported libargo • Ported interposer • Potentially a backend to VirtIO-Argo - to be determined
Bromium uXen v4v device drivers, port to Argo and expose both char and network devices, each optional via KCONFIG	Larger aggregate driver size for upstreaming work	• Flexible • supports standard networking interface expectations		• Potentially OpenXT service VMs; though VirtIO-Argo may be better suited	• OpenXT platform VMs • Ported libargo • Ported interposer

		• char interface can support Argo firewall and misc functions + comms when networking not present		• Ported uXen v4v storage and network drivers • Ported libargo	• Potentially a backend to VirtIO-Argo - to be determined
Xen IOREQ for connection to VirtIO-Argo	Appropriate for Xen community, given current VirtIO IOREQ activity	• Adheres to existing Xen architecture • Can simplify a userspace device model implementation	Requires use of emulation infrastructure in the hypervisor and a privileged domain, foreign memory mapping privilege, grants and event channels	None	Device emulators for providing virtual devices

Considerations on system architecture for the new driver structure

- Changes to the software that will run in the guests:
  - This will affect evaluations of Xen systems that adopt VirtIO-Argo software:
    - Xen device drivers may not be required within guests any more, since the VirtIO network and block alternatives can be operational instead.
    - VirtIO device drivers may be required within guests and will most commonly be obtained from inclusion in the upstream Linux kernel.
    - Xenstore may no longer be required to support guests running with virtual devices enabled.
- Changes to the software that will run in the platform VMs:
  - VirtIO virtual device emulation software will support guest virtual devices
  - XSM/Flask policy will have granular control over guest access to virtual devices
  - Will not require shared memory for device driver operations
- Toolstack changes
  - VirtIO-Argo will not require xenstore, although QEMU may have dependency on it
  - XSM Argo firewall will not require libxl hooks to the VM lifecycle state machine to configure the Argo firewall, instead enforcement will be applied by a static system XSM policy
- Hypervisor software changes
  - A different selection of hypervisor functionality will be enabled and disabled via KConfig options
    - some options will become mandatory: eg. Argo, XSM
    - some options may no longer be mandatory
- Multi-hypervisor systems:
  - There have been design discussions about how to enable communication with Argo between VMs at different levels of nested hypervisors: this can then be applied to the VirtIO-Argo use case for Argo. It will enable driver domains to be hosted at different levels of nesting, and hosted on different hypervisors.

Initial Use Case: GPU remoting over VirtIO for interdomain graphics

- The Chromium Linux kernel has a device driver virtio_wl, also known as virtio-wayland, that implements a transport for the Wayland protocol over VirtIO, proxying the Wayland protocol socket stream over VirtIO queues.
  - This enables any Wayland compositor to use VirtIO for connections between client applications and the display, so a single desktop VM can render applications running in remote VMs.
  - This technology was developed for Chromium OS to enable crosvm to run Linux applications within sandboxes.
- Note: As a display technology that uses framebuffers, the VirtIO Wayland driver allows the guest to request shared memory file descriptors and implementing handling for these regions will be required in addition to the implementation of Argo as the transport for the VirtIO virtqueues.
- To be explored: opportunities for collaboration with EPAM and Qubes.
- Reference: A Technical Overview of virtio-wayland

December 2020 Review feedback

- Response with multiple points from Jean-Philippe Ouellet:
  - Interest in ensuring interface support for capability-oriented policy enforcement models
  - Method and interfaces for device enumeration and hotplug queried: to be determined
  - Interest in userspace interfaces to Argo, with respect to the existing userspace support provided for the grant tables
  - Granular disaggregation of backend device-model providers desirable
  - Asynchronous non-blocking send primitives could be useful for increased scalability

        - Need to evaluate the Linux device driver interface to userspace for flexibility:
            - prefer “can write N bytes” or “how many bytes can be written?” over “try to write N, only wrote M + EAGAIN”
                - latter can be implemented over former but not vice-versa
                - matters for userspace for backpressure to peer userspace without buffering
                - matters for ensuring truth of durability of writes
                - affects cross-domain EPIPE boundary correctness
                - ref: Qubes porting vchan from Xen to KVM vsock
            - Some significant VirtIO drivers explicitly use shared memory eg. for framebuffers
                - virtio-fs (DAX-like fs backing to share page cache); virtio-gpu, virtio-wayland, virtio-video
    - Projects added above following feedback from Roger Pau Monné and Julien Grall:
        - Hypervisor-agnostic MSR interface to enable use of VirtIO-Argo transport driver with other hypervisors, with reference to the specifics of the x86 and Arm architectures
        - Argo currently delivers notifications to guests via Xen event channels; it may be “better if interrupts are delivered using a native mechanism, something like MSI delivery by using destination APIC ID, vector, delivery mode and trigger mode.”
    - Related material references from Rich Persaud:
        - PX: an OSS L0 “Protection Hypervisor” in the Hardened Access Terminal (HAT) architecture presented by Daniel Smith at the 2020 Xen Design and Developer Summit: https://youtube.com/watch?v=Wt-SBhFnDZY&t=3m48s
            - PX is intended to build on lessons learned from IBM Ultravisor, HP/Bromium AX and AIS Bareflank L0 hypervisors:
                - Notes from Dec 2019 meeting in Cambridge, Day 2 discussion included L0 nesting hypervisor, UUID semantics, Argo, communication between nested hypervisors
                - Xen Summit 2020 Bareflank design session notes
        - In the long-term, efficient hypervisor nesting will require close cooperation with silicon and firmware vendors. Note that Intel is introducing TDX (Trust Domain Extensions):
            - https://software.intel.com/content/www/us/en/develop/articles/intel-trust-domain-extensions.html
            - https://www.brighttalk.com/webcast/18206/453600
        - Recent papers from Shanghai Jiao Tong University, on using hardware instructions to accelerate inter-domain HMX:
            - March 2019: “SkyBridge, a new communication facility designed and optimized for synchronous IPC in microkernels” https://ipads.se.sjtu.edu.cn/_media/publications/skybridge-eurosys19.pdf
            - July 2020: “UnderBridge” … “a redesign of traditional microkernel OSes to harmonize the tension between messaging performance and isolation” https://ipads.se.sjtu.edu.cn/_media/publications/guatc20.pdf

Related material

- VirtIO & Argo
- Analysis of Argo as a transport medium for VirtIO
- New Linux Driver for Argo
- Argo : Hypervisor-Mediated data eXchange : Development
- “Argo: VirtIO” presented to the Automotive Grade Linux Virtualizaton Experts Group, August 2020
    - Slides
    - Mailing list post
- XCP-ng blog: Device Emulation in the Xen Hypervisor for HVM Guests
- A Technical Overview of virtio-wayland

Source for this Document

This document is developed and made available on the OpenXT wiki at the following location:

- VirtIO-Argo Development: Phase 1

License of this Document