DRAFT RFC: Argo and HMX

This is a pre-release DRAFT version of this document. **Please do not distribute.**
Note: This document will be reformatted into ascii / markdown mailing-list appropriate presentation prior to mailing list distribution.

Primary author and contact: Christopher Clark
Date: 19th June 2018

Argo: Inter-VM communication primitives for Hypervisor-Mediated data eXchange.

This proposal describes a proposed copy-based inter-VM communication system.

Objective for this RFC:

- Communicate clearly:
    - The motivation for this software and its design.
    - The intended scope for this software.
    - The proposed software architecture.

- Obtain approval for the architecture, with the understanding that this will then proceed towards an implementation of it within Xen.

Motivation for Argo

Required: A Strong Mechanism for inter-VM communication, supporting robust isolation.

Security, Safety and Mixed-Criticality Systems have isolation requirements that are relevant to this system.

Context

Concept: **Strength of Mechanism**:
Measure of inherent resistance provided by a mechanism to adversarial conditions.

"An assessment of how well it reduces the skepticism that the mechanism will fail through either direct or indirect application of force." -- Daniel Smith, Apertus Solutions, on TrenchBoot at

Platform Security Summit 2018. Describing the concept documented in the Information Assurance Technical Framework, section 4.5.3, US National Security Agency.
Also **presented at Xen Summit, 2007**, by John McDermott, US Naval Research Lab, with acknowledgement to its origin at the NSA.

Concept: **MILS Architecture Foundational Security Principals**
described in CrossTalk, Journal of Defense Software Engineering, 2005.
Authors: Vanfleet, Beckwith, Calloni, Luke, Taylor, Uchenick.

- **Data Isolation**
    - Information in a partition is accessible only by that partition and private data remains private.

- **Control of Information Flow**
    - Information flow between partitions is from an authenticated source to authenticated recipients; the source of information is authenticated to the recipient, and information only goes where intended.

- **Periods Processing / Temporal Separation**
    - Resources may be used by different components by time-slicing, where the system enforces that the resource is cleaned to remove any trace of its previous use before being reassigned.

- **Fault Isolation**
    - Failure within a partition is prevented from cascading to any other partition. Failures are detected, contained and recovered locally.

Concept: **Hypervisor-Mediated data eXchange**

We introduce this term and define it to refer to inter-VM communication protocols that have this key architectural point: *The hypervisor is responsible for performing the write of data into the guest-accessible memory buffer, in the manner according to the agreed transfer protocol.*

This structure ensures that there is Strength to the transport mechanism, because the transmitting side of the communication is the hypervisor, which can be trusted by the receiver, and the buffer is isolated from access by any other potential sources outside the receiver.

The receiver can trust that the hypervisor will:

- Provide a protocol implementation adhering to hardware synchronization requirements for concurrent access to system memory by communicating components.

- Only deliver data from an approved source.
  Policy for Mandatory Access Control will be enforced.

- Indicate the correct sender of the data.

- Only transmit the intended data, adhering to the access protocol of the data structure in the buffer.
  eg. If the memory region is being used as a ring, then:

  - Data writes will only occur within the ring region that is indicated as available for incoming data by the ring indexes.
  - The indicated length of data written will exactly match the length of data actually written.
  - The write for each piece of data will occur only once.
  - Data will be written sequentially in the order that it is sent.

- Issue notifications to inform of delivered data correctly.

The structure allows for augmentation by the hypervisor to identify the sending entity within the source VM, and then provide the receiver with assured context information about the data source. This enables the receiver to make decisions based on fine-grained knowledge of the source of the data (see: MILS: Control of Information Flow).

The structure also supports the optional interposition of additional validation within the transmission path: the hypervisor can be configured to submit the data to another VM for approval according to policy, and this can be done without requiring a change of interface to either of the communicating VMs.

This allows for different implementations of the transport with varying levels of assurance of data integrity or confidentiality properties for the connection transport.

This structure is also of strong interest for nested virtualization: transport via the hypervisor should enable construction of highly efficient communications between VMs at different levels of nesting, with the difference in nesting level being potentially transparent to the endpoints.

General challenges with using shared memory correctly

It is challenging to build and maintain correct and secure concurrent software with data structures built upon shared memory. [todo: refs]

Modern CPU architecture and optimizing compilers result in instruction execution and memory access behaviour that is non-intuitive and yet it is critical for software correctness.

[ref: XSA-155 : compiler optimization, PV drivers, double-fetch]

The Linux Kernel Memory Model [see References section at end of document] is an example of modern, advanced, comprehensive tooling constructed to support reasoning about concurrent computation with shared memory.

This RFC author's view is that LKMM's existence demonstrates that shared memory data structures are considered important enough, and the complexity of the primitives they are built from is known to be high enough, that this intricate reasoning-support infrastructure was determined to be worth building, populating with model data, and including in the mainline Open Source project with a commitment to maintain it.

Specific problems with using shared memory between VMs as Xen's building primitive for communication channels

Distribution of work across all guest implementations

Simple shared memory primitives, such as those provided by the grant tables, place the work of constructing functioning communication channels into both sides of every split device driver in every OS or unikernel. The ring macros are an example of provided assistance for doing this.

This placement of responsibility outside of the hypervisor raises the ongoing cost of securing deployed systems.

This recent xen-devel thread asked the question:

"Should PV frontend drivers trust the backends?"

https://lists.xenproject.org/archives/html/xen-devel/2018-04/msg01942.html

There are different perspectives on the answer to this but when the hypervisor mechanisms render the need for trust between VMs moot, then the difference of opinions goes away, with systems able to make use of resilient communication channels.

Avoidance of shared memory between VMs for data exfiltration

Shared memory pages established for use as rings by PV drivers also allow other data to be transmitted between VMs, even beyond the intended channels that they were established to serve. Interposition by the hypervisor to deter this is invasive and likely to cause inefficiency in the performance of the channel. Structures that avoid this are preferable.

Argo objectives

Simple to use correctly, securely, safely.

HMX-compliant

Adheres to the Hypervisor-Mediated data eXchange architectural requirements.

A simple, clean interface with well-understood semantics

Argo maps well onto both conventional sockets and the Window native I/O interfaces, enabling intuitive use with both existing and newly developed software. This reduces the scope for bugs introduced due to miscomprehension of protocol behaviour.
[todo: expand + explain]

Avoidance of interaction with specific Xen system components

Argo has no dependency or interaction with XenStore or the Grant Tables.

The option to deliver notifications either via VIRQ or Event Channels is configured on a per-guest basis according to MAC and system policy:

- VIRQ: to allow for simple guests to operate without complex Event Channel software required within their OS.
    - Note that the Event Channel implementation in the Linux kernel involves non-trivial shared memory data structures that are required to correctly handle concurrent accesses. It is reasonable for deployments that prioritizes security highly to seek to minimize their kernels and this is a viable candidate for exclusion, as demonstrated by Bromium's hypervisor implementation.
    - Providing Argo the option to use a VIRQ can allow Argo connections to function while using XSM to block a domain's access to Event Channels entirely.
- Events: Enabled for potentially improved performance scalability when Event Channel logic is present within the guest OS and system policy allows it.

VIRQ may be simpler for use in communication on nested hypervisor systems.

Non-goals for this version

- Replace any other interdomain communication mechanisms.
- Provide compatibility support for any other interdomain communication mechanisms.
- Performance quantification or tuning.
- Provision for real-time data connections.
- Provision for latency-sensitive multimedia data connections.
- Resistance to Denial of Service.

- Resistance to Resource Exhaustion.
- Port firewall for interdomain connections.
- Provision of detailed assured connection source context to destination.
- Scalability.
- Clean up of guest connection state on VM migration.
    - Initial use cases are on systems where VMs do not migrate.

Components

The Argo software comprises:

- A hypervisor implementation of a new hypercall, implementing a series of operations.
    - Inclusion of this within the hypervisor will be optional, its presence toggled via KCONFIG option selection.

- A Linux kernel device driver, to provide access to Argo connections to Linux kernel and user space application software.

- A Linux shared library for use with LD_PRELOAD to enable transparent use of Argo connections by existing software designed for TCP/IP networking.

- Windows software to provide access to Argo connections by the Windows kernel and user space applications.

User Experience: Guest Kernel Perspective

Argo provides hypervisor primitives to transmit data between VMs by performing data copies into receive memory rings registered by domains. There is no memory sharing between VMs.

The receiving domain registers a ring with Xen, supplying the memory to be used via hypercall.

The sending domain issues a hypercall to request that Xen inserts data into the receiving domain's ring.

Notification to a receiver that there is new data to read is delivered either by a VIRQ or an Event on an Event Channel.

A domain can request that an interrupt be generated when sufficient space is available in another domain's ring.

The code that inserts transmitted data into the ring is in the hypervisor, which writes a header describing the data and where it came from in a header preceding the data. As each of the data contents, the ring indices and the header are all written by the hypervisor, the receiving domain can trust their content and the maintained integrity of the ring.

Hypercall operations

- register_ring
    - registers a receiving ring with Xen.

- unregister_ring
    - unregisters a receiving ring.

- sendv
    - sends the data supplied in a list of buffers.

- notify
    - query information about a remote ring, optionally registering for notification by interrupt once space becomes available.

- info
    - queries information about the current domain's configuration, eg. which event channel is in use for notifications.

Addressing

Argo connections are formed with a 4-tuple of source port and domain, destination port and domain, with each end of the channel defined by an address structure thus:

```
struct argo_addr {
{
    uint32_t port;
    domid_t domain;
};
```

Domain IDs are unique per host and serve as the endpoint address.
The port value is analogous to a TCP/IP port that specifies some service at a particular address.

Rings

A domain that wishes to communicate must register an argo_ring memory buffer via the hypercall. The ring is identified by an argo_ring_id structure, defined as:

```
struct argo_ring_id
{
    struct argo_addr addr;
    domid_t partner;
};
```

If the partner domain is specified, then communication will only be received from that domain. If the partner domain is not specified, and instead supplied as an ANY wildcard, the ring will be a multicast receiver, able to receive transmissions from any other domain.

The ring memory buffer itself has this structure:

```
struct argo_ring
{
    uint64_t magic;
    struct argo_ring_id id;
    uint32_t len;
    uint32_t rx_ptr;
    uint32_t tx_ptr;
    uint8_t reserved[32];
    uint8_t ring[0];
}
```

The ring size is stored in the len field and the ring data, only ever written by the hypervisor, starts at ring[0]. rx_ptr is the receive pointer into the ring and is only ever written by the domain that owns it, as it consumes ring data. tx_ptr is the transmit pointer into the ring, only ever written by the hypervisor as new data is inserted.

Setup

Rings are registered by issuing a hypercall, providing the arguments with the block of memory owned by the domain to be configured for use as a receive ring. Unregistering is simple, also via hypercall, and can be performed by the owning domain at any time.

Sending

Handles to the data to be transmitted are supplied as arguments to send operation invoked via hypercall. If insufficient receive buffer is available then the return code will indicate that a retry is required and a notification will be issued when sufficient space in the destination becomes available.

Receiving

The receiver domain can read from its receive ring at any time. The rx_ptr and tx_ptr fields indicate the volume of data to be read. Notifications - either VIRQ or events - will be received to indicate the arrival of incoming data.

Notification

The notify operation of the hypercall can be issued by a receiver to trigger notification of prospective interested senders of receive space availability.

The notify operation is also used to register for interest in notifications of space availability in a destination ring.

Ring Reregistration

An OS can reregister a ring that already exists. In this case the memory provided by the guest in the register operation will replace that previously used by the hypervisor as the receive ring for the specified connection.

Notes

The hypervisor is responsible for:

- Ring index manipulation
- Data copying
- Delivery notification to endpoints

Many-to-one "multicast" receiving enables Services to:

- Operate without requiring establishment of a ring-per-client.
- Listen for new connection negociation requests.

Design Rationale / Architecture Questions

Q: Can this be implemented with a multicall containing some grant table copy operations and event channel manipulations?

A: No, not precisely and important aspects - those relating to trust in the integrity of the data and protocol adherence, plus the isolation provided by the HMX architecture - would be lost.

Q: Using one receiver ring for all clients raises the question of access control and admission control: how do you avoid DoS from badly behaved clients?

A: Admission control and resistance to DoS is declared out of scope for this RFC. In future work, the receiver ring registration can be extended to indicate the admission control scheme to be applied, along with code to enforce it. In planning for this, an admission control selection field may be included in the ring registration hypercall op, with a default ‘none’ implementation.

Security Considerations

The trust relationship between two communicating VMs differs from that between each VM and the hypervisor: VMs must rely on the hypervisor to act correctly and respect their interests, but that need not be the case between VMs.

Shrinking the code that a VM must depend upon (TCB) for correct operation supports improvement to security and safety: a smaller amount of code should be easier to audit and reduce the opportunities for bugs.

Simple to validate correctness of code essential to the channel.

The critical code for this mechanism resides within a single hypercall in the hypervisor.

The owner of a VM can be confident in the integrity of the channel if they trust the source of the VM and the hypervisor. Confidence is not required in the VM at the other end of the channel, in order to trust the integrity of the channel itself, as distinct from the data transmitted across the channel, or the timely responsiveness of the remote VM to interactions.

Comparison with existing technologies

TODO: Explanation of why what v4v provides is not attainable via any other existing technologies.

TODO: Reference to the Hyper-V HMX-compliant implementation.

TODO: Contrast to SEL4 IPC, Linux-KVM Virtio, and other pertinent technologies.

Software history

An earlier version of this software, previously known as v4v, has been in use by OpenXT and its derivatives for several years. An alternative variant has been in use by Bromium and is included in their current product, with source code in uXen at: https://www.bromium.com/opensource

FAQ:

Q: Is it OK to have copying being performed by the hypervisor?
A: Yes. It is already doing that for VM networking. (See GNTTABOP_copy).

Q: Why not use the network front and back ends?
A: Strength of mechanism; TCB reduction.

Q: Naming: Why not v4v? Why Argo?

A: v4v has both been previously communicated on xen-devel, by developers no longer participating in the Xen Project, and in use for a considerable time outside of the Xen Project, and an understanding of it has been formed that may not accurately correspond to the work being performed as part of this development effort and its future evolution.

The mythical Argo was a secure form of transport.

The Argo software is subject to change, and not constrained by previous decisions taken for v4v, so to avoid any misunderstanding, and encourage inspection of the work as it is implemented during this submission process, this effort is undertaken with a different name. As Argo is developed further, the divergence from its origins will increase.

Credits

The architecture, software and documentation described in this proposal would not exist without the contributions of a number of highly capable software engineers and support and advocacy by members of their teams.

- (todo: provide a researched list here -- top of head major contributors: James McKenzie, Jean Guyader, Ross Philipson, but there are others - don't mess this up...)

References

Design and Verification of Secure Systems, John Rushby, SRI International.

Introduces: Separation Kernels.

ACM Symposium on Operating System Principals, 1981.
http://www.csl.sri.com/users/rushby/papers/sosp81.pdf

MILS Architecture Foundational Security Principals

Vanfleet, Beckwith, Calloni, Luke, Taylor, Uchenick.
CrossTalk, Journal of Defense Software Engineering, 2005.
http://www.crosstalkonline.org/storage/issue-archives/2005/200508/200508-Vanfleet.pdf

Strength of Mechanism:

- TRUSTED NETWORK INTERPRETATION ENVIRONMENTS GUIDELINE, NCSCTG-11, National Computer Security Center
- Information Assurance Technical Framework, section 4.5.3, National Security Agency
    - http://www.dtic.mil/dtic/tr/fulltext/u2/a606355.pdf
- Xenon, John McDermott of US NRL, Xen Summit 2007
    - http://www-archive.xenproject.org/files/xensummit_4/XenSummitSpring07_McDermott.pdf
- TrenchBoot, Daniel Smith of Apertus Solutions, Platform Security Summit 2018
    - [todo: link]

Linux Kernel Memory Model

https://lwn.net/Articles/718628/
https://lwn.net/Articles/720550/

https://www.youtube.com/watch?v=F4LrTLFeq4I
https://galois.com/blog/2018/05/linux-kernel-memory-ordering-help-arrives-at-last/

A talk on the Linux Kernel Memory Model, by Paul E. McKenney, of IBM and the maintainer of Read-Copy-Update in the Linux kernel. May 16, 2018.

Describes the difficulties of getting concurrent programming correct with modern performant CPUs and compilers: CPU architecture and compiler optimization produce behaviour that is extremely difficult to reason about. The Memory Model is tooling to assist that reasoning.

Talk mentions defects identified in the RISC-V processor by using the tooling.

Linux commit 63cae12bce9861cec309798d34701cf3da20bc71 includes a "litmus test" in the commit message, that validates that the logic in the commit is safe using the Linux Memory Model.
https://patchwork.kernel.org/patch/9516943/

v4v Documentation

This RFC document includes material from the following v4v document, which is adapted under the terms of the CC BY 4.0 license, and is by Ross Philipson and copyright Citrix Systems, Inc. 2014
https://openxt.atlassian.net/wiki/spaces/DC/pages/14844007/V4V

Significant public contributions were made in discussion on the xen-devel mailing list, with submissions driven by Jean Guyader and Ross Philipson, with notable feedback from Jan Beulich, Tim Deegan, Ian Campbell, Andrew Cooper, David Vrable and Daniel De Graaf.
[todo: list important v4v thread from xen-devel]
[todo: link to OpenXT v4v components on github]
[todo: link to Bromium's v4v implementation in their Open Source material]
[todo: link to Stephen Smalley's OpenXT Summit 2016 presentation]
[todo: link to presentation of v4v in OpenXT presentation at Xen Summit 2016]