Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor

Reiner Sailer Trent Jaeger Enriquillo Valdez Ramón Cáceres
Ronald Perez Stefan Berger John Linwood Griffin Leendert van Doorn

{sailer,jaegert,rvaldez,caceres,ronpz,stefanb,jlg,leendert}@us.ibm.com

IBM T. J. Watson Research Center, Hawthorne, NY 10532 USA

Abstract

We present the sHype hypervisor security architecture and examine in detail its mandatory access control facilities. While existing hypervisor security approaches aiming at high assurance have been proven useful for high-security environments that prioritize security over performance and code reuse, our approach aims at commercial security where near-zero performance overhead, non-intrusive implementation, and usability are of paramount importance. sHype enforces strong isolation at the granularity of a virtual machine, thus providing a robust foundation on which higher software layers can enact finer-grained controls. We provide the rationale behind the sHype design and describe and evaluate our implementation for the Xen open-source hypervisor.

1 Introduction

As workstation- and server-class computer systems have increased in processing power and decreased in cost, it has become feasible to aggregate the functionality of multiple standalone systems onto a single hardware platform. For example, a business that has been processing customer orders using three computer systems–a web server front-end, a database server back-end, and an application server in the middle—can increase hardware utilization and reduce its hardware costs, configuration complexity, management complexity, physical space, and energy consumption by running all three workloads on a single system.

Virtualization technology is quickly gaining popularity as a way to achieve these benefits. With this technology, a software layer called a virtual machine monitor (VMM), or *hypervisor*, creates multiple virtual machines out of one physical machine, and multiplexes multiple virtual resources onto a single physical resource. Virtualization is facilitated by recent development in terms of broad availability of fully virtualizable CPUs [2, 15]. These advances make possible efficient aggregation of multiple virtual machines on a single physical machine, with each virtual machine (VM) running its own operating system (OS).

Although co-locating multiple operating systems and their workloads on the same hardware platform offers great benefits, it also raises the specter of undesirable interactions between those entities. Mutually distrusted parties require that the data and execution environment of one party's applications are securely *isolated* from those of a second party's applications. As a result, virtualization environments by default do not give VMs direct access to physical resources. Instead, physical resources (e.g., memory, CPU) are virtualized by the hypervisor layer and can be accessed by a VM only through their virtualized counterparts (e.g., virtual memory, virtual CPU). The hypervisor is strongly protected against software running in VMs, and enforces isolation of VMs and resources.

However, total isolation is not desirable because today's increasingly interconnected organizations require communication between application workloads. Consequently, there is a need for secure resource sharing by enforcing access control between related groups of virtual machines.

The main focus of this paper is on the controlled sharing of resources. In current hypervisor systems, such sharing is not controlled by any formal policy. This lack of formality makes it difficult to reason about the effectiveness of isolation between VMs. Furthermore, current approaches do not scale well to large collections of systems because they rely on human oversight of complex configurations to ensure that security policies are being enforced. They also do not support workload balancing through VM migration between machines well because the policy representations are machine-dependent.

This paper explores the design and implementation of sHype, a security architecture for virtualization environments that controls the sharing of resources among VMs according to formal security policies. sHype goals include (i) near-zero overhead on the performance-critical path, (ii)

non-intrusiveness with regard to existing VMM code, (iii) scalability of system management to many machines via simple policies, and (iv) support for VM migration via machine-independent policies.

These goals are derived from the requirements of commercial environments. Hypervisor security approaches aimed at high assurance have proven useful in environments that give security the highest priority. These approaches control both explicit and implicit communication channels between VMs. We believe that controlling explicit data flows and minimizing, but not entirely eliminating, covert channels via careful resource management is sufficient in commercial environments.

We implemented the sHype architecture in the Xen hypervisor [3], where it controls all inter-VM communication according to formal security policies. The architecture is designed to achieve medium assurance (Common Criteria EAL4 [8]) for hypervisor implementations. Our modifications to the Xen hypervisor are small, adding about 2000 lines of code. Our hypervisor security enhancements incur less than 1% overhead on the performance-critical path and the Xen paravirtualization overhead is between 0%-9% [3]. While this paper describes an sHype implementation tailored to the Xen hypervisor, the sHype architecture is not specific to any one hypervisor. It was originally implemented in the rHype research hypervisor [14] and is also being implemented in the PHYP [13] commercial hypervisor.

Section 2 introduces the Xen hypervisor environment in which we have implemented our generic security architecture. Mutually suspicious workload types serve as an example to illustrate requirements and the use of our hypervisor security architecture. We describe the design of the sHype hypervisor security architecture in Section 3, and its Xen implementation in Section 4. Section 5 evaluates our architecture and implementation, and Section 6 discusses related work.

2 Background

2.1 The Xen Hypervisor

We use the Xen [3] open-source hypervisor as an example of a virtual machine monitor throughout this paper. Figure 1 illustrates a basic Xen configuration. The hypervisor consists of a small software layer on top of the physical hardware. It implements virtual resources (e.g., vMemory, vCPU, event channels, and shared memory) and it controls access to I/O devices.

Virtual machines, also known as *domains* in Xen, are built on top of the Xen hypervisor. A special VM, called Dom0 (domain zero) is created first. It serves to manage other VMs (create, destroy, migrate, save, restore) and controls the assignment of I/O devices to VMs.

VMs started by Dom0 are called DomUs (user domains). They can run any para-virtualized [3] operating system, e.g., Linux. Guest OSs running on Xen are minimally changed, for example by replacing privileged operations with calls to the hypervisor. Such operations cannot be called directly by the guest OS because they can compromise the hypervisor. In general, calls to the hypervisor have three characteristics: (1) they offer access to virtual resources; (2) they speed up critical path operations such as page table management; and (3) they emulate privileged operations that are restricted to the hypervisor but might be necessary in guest operating systems as well.

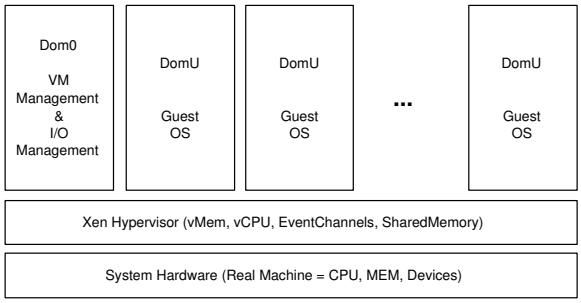

Figure 1. Xen hypervisor architecture

Xen offers just two shared virtual resources on top of which all inter-VM communication and cooperation is implemented:

- Event channels: An event-channel hypervisor call enables a VM to setup a point-to-point synchronization channel to another VM.
- Shared memory: A grant-table hypervisor call enables a VM to allow another VM access to virtual memory pages it owns. Event channels are used to synchronize access to such shared memory.

Shared virtual resources, such as virtual network adapters and virtual block devices, are implemented as device drivers inside the Guest OS. Non-shared virtual resources include virtual memory and virtual CPU.

Physical resources differ from virtualized resources in a couple of key ways: (1) Input/Output Memory Management Units (IO-MMUs) are needed to restrict Direct Memory Access (DMA) to and from a VMM's memory space. (2) Performance is best if the devices are co-located with the code using them in the same VM, and consequently the optimal case is a physical resource per VM, which may not be practically feasible. (3) Driver code is too complex for inclusion in the hypervisor, so a device to be shared by multiple VMs needs to be managed by a device domain, which then makes this device available through inter-VM sharing

to other VMs. In Xen, a SCSI disk or Ethernet device, for example, can be owned by a device domain and accessed by other VMs through virtual disk or Ethernet drivers, which communicate with the device domain using event channels and shared memory provided by the hypervisor.

2.2 Coalitions of VMs

In the near future, we believe that VM systems will evolve from a set of isolated VMs into sets of VM coalitions. Due to hardware improvements enabling reliable isolation, we believe that some control now done in operating systems will be delegated to hypervisors. We aim for hypervisors to provide isolation between coalitions and provide limited sharing within coalitions as defined by a Mandatory Access Control (MAC) policy.

Consider a customer order system. The web services and data base infrastructure that processes orders must have high integrity in order to protect the integrity of the business. However, browsing and collecting possible items to be purchased need not be as high integrity. At the same time, an OEM's software advertising a product that the company distributes may be run as another workload that should be isolated from the order workloads (web service, database, browsing).

In the customer order example, we merge the VMs performing customer orders into the *Order* coalition and protect them from the other VMs on the system. The Order VMs may communicate, share some memory, network, and disk resources. Thus, they are as a coalition confined by the hypervisor. Within the Order coalition, the hypervisor controls sharing using a MAC policy that permits inter-VM communication, sharing of network resources and disk resources, and sharing of memory. All this sharing must be verified to protect security of the order system. However, the MAC policy also enables the hypervisor and device domains to protect the order database from being shared with other VMs outside the *Order* coalition.

2.3 Problem Statement

The problem we address in this paper is the design of a VMM *reference monitor* that enforces comprehensive, mandatory access control policies on inter-VM operations. A reference monitor is designed to ensure mediation of all security-sensitive operations, which enables a policy to authorize all such operations [16]. A MAC policy is defined by system administrators to ensure that system (i.e., VMM) security goals are achieved regardless of system user (i.e., VM) actions. This contrasts with a discretionary access control (DAC) policy which enables users (and their programs) to grant rights to the objects that they own.

We apply the reference monitor to control all references to shared virtual resources by VMs. This allows coalitions of workloads to communicate or share resources within a coalition, while isolating workloads of different coalitions.

Figure 2 shows an example of VM coalitions. Domain 0 has started 5 user domains (VMs), which are distinguished inside the hypervisor by their domain ID (VM-id in Fig. 2). Domains 2 and 3 are running *order* workloads. Domain 6 is running an *advertising* workload, and domain 8 is running an unrelated generic *computing* workload. Finally, domain 1 runs the virtual block device driver that offers two isolated virtual disks, *vDisk Order* and *vDisk Ads*, to the *Order* and *Advertising* coalitions. In this example, we want to enable efficient communication and sharing among VMs of the *Order* coalition but contain communication of VMs inside this coalition. For example, no VM running an *Order* workload is allowed to communicate or share information with any VM running *Computing* or *Advertising* workloads, and vice versa.

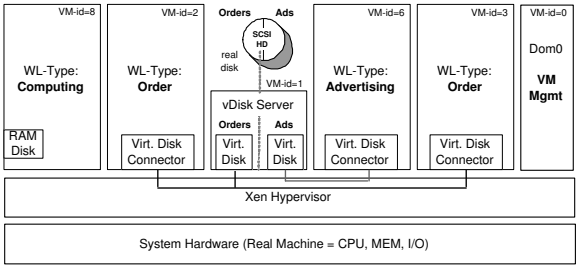

Figure 2. VM coalitions and payloads in Xen

While the hypervisor controls the ability of the VMs to connect to the device domain, the device domain is trusted to keep data of different virtual disks securely isolated inside its VM and on the real disk. This is a reasonable requirement since device domains are not application-specific and can run minimized run-time environments. Device domains thus form part of the Trusted Computing Base (TCB).

3 sHype Design

Figure 3 illustrates the overall sHype security architecture and its integration into the Xen VMM system. sHype is designed to support a set of security functions: secure services, resource monitoring, access control between VMs, isolation of virtual resources, and TPM-based attestation.

sHype supports interaction with secure services in custom-designed, minimized, and carefully engineered VMs. An example is the policy management VM, which we use to establish and manage the security policies for the Xen hypervisor. Resource accounting provides control of resource usage. This enables enforcement of service level agreements and addresses denial of service attacks on hypervisor or VM resources. The mandatory access control

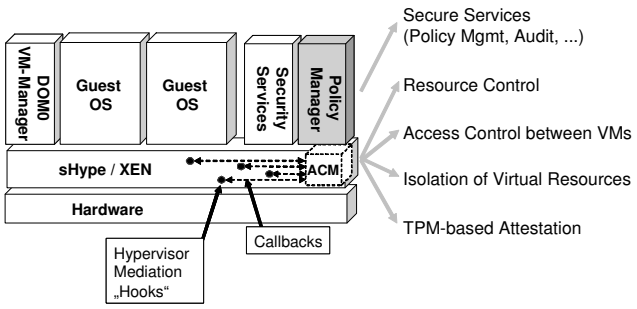

Figure 3. sHype architecture

enforces a formal security policy on information flow between VMs.

sHype leverages existing isolation between virtual resources and extends it with MAC features. TPM-based attestation [28] provides the ability to generate and report runtime integrity measurements on the hypervisor and VMs. This enables remote systems to infer the integrity properties of the running system.

The rest of this paper focuses on the sHype mandatory access control architecture, consisting of: (1) the *policy manager* maintaining the security policy; (2) the *access control module* (ACM) delivering authorization decisions according to the policy; and (3) and *mediation hooks* controlling access of VMs to shared virtual resources based on decisions returned by the ACM.

3.1 Design Decisions

Three major decisions shape the design of sHype:

(1) By *building on existing isolation properties of virtual resources*, sHype inherits the medium assurance of existing hypervisor isolation while requiring minimal code changes in the virtualization layer (hypervisor).

(2) By using *bind-time authorization* and controlling access to spontaneously shared resources only on first-time access and upon policy changes, sHype incurs very low performance overhead on the critical path.

(3) By *enforcing formal security policies*, sHype enables reasoning about the effectiveness of specific policies, provides the basis for effective defense against denial of service attacks (through resource policy enforcement), and enables Service Level Agreement-style security guarantees (through TPM-based attestation of system properties).

3.2 Access Control Architecture

The key component of the access control architecture is the reference monitor, which in sHype isolates virtual machines by default and allows sharing of resources among virtual machines only when allowed by a mandatory access control (MAC) policy. To support various business requirements, sHype supports various kinds of MAC policies: Biba [5], Bell-LaPadula [4], Caernarvon [30], Type Enforcement [6], as well as Chinese Wall [7] policies.

The classical definition of a reference monitor [16] states that it possesses three properties: (1) it mediates all security-critical operations; (2) it can protect itself from modification; and (3) it is as simple as possible to enable validation of its correct implementation. We examine the first requirement in more detail. The second and third requirement are covered by generic hypervisor properties: it is protected against the VMs and consists of a thin software layer.

Mediating security-critical operations. A security-critical operation is one that requires MAC policy authorization. If such an operation is not authorized against the MAC policy, the system security guarantees can be circumvented. For example, if the mapping of memory among VMs is not authorized, then a VM in one coalition can leak its data to other VMs.

We identify security-critical operations in terms of resources whose use must be controlled in order to implement MAC policies. We also identify the location of the mediation points for these resources. The combination of resources to be controlled and their mediation points forms the reference monitor interface. We discuss only virtual resources, because real resources can only be used exclusively by one VM or shared in the form of virtual resources. The following resources must be controlled in a typical Xen VMM environment:

- Sharing of virtual resources between VMs controlled by the Xen hypervisor (e.g., event channels, shared memory, and domain operations).
- Sharing of local virtual resources between local VMs controlled by MAC domains (e.g. local vLANs and virtual disks).
- Sharing of distributed virtual resources between VMs in multiple hypervisor systems controlled by MAC-bridging domains (e.g., vLANs spanning multiple hypervisor systems).

The hypervisor reference monitor enforces access control and isolation on virtual resources in the Xen hypervisor. While sHype enforces mandatory access control on MAC domains regarding their participation in multiple coalitions, it relies on MAC domains to isolate the different virtual resources from each other and allow access to virtual resources only to domains that belong to the same coalition as the virtual resource. A good example of a MAC domain is the device domain in Fig. 2, which participates in both the `Order` and the `Advertising` coalition. MAC domains become part of the Trusted Computing Base (TCB)

and should therefore be of minimal size (e.g., secure microkernel design). Since MAC domains are generic, the cost of making them secure will amortize as they are used in many application environments. We sketch the implementation of MAC domains in Section 4.4.

If coalitions are distributed over multiple systems, we need MAC-bridging domains to control their interaction. The virtual resource that enables co-operation among VMs on multiple systems is typically a vLAN. Mac-bridging domains build bridges between their hypervisor systems over untrusted terrain to connect vLANs on multiple systems. To do so, they first establish trust into required security properties of the peer MAC Bridging domains and their underlying virtualization infrastructure (e.g., using TPM-based attestation). Afterwards, they build secure tunnels between each other, and can from now on be considered as forming a single (distributed) MAC domain spanning multiple systems. Requirements on the resulting distributed MAC domain are akin the requirements described above for local MAC domains. MAC Bridging domains become part of the TCB, similarly to MAC domains.

4 Implementation

In this section, we first define simple policies tailored to the Xen hypervisor environment based on the workload types and resources that must be controlled. Then we describe the management of the policies and the labeling of VMs and resources. Finally, we introduce the access control enforcement in the hypervisor, which guards access of VMs to resources based on the policies.

4.1 Security Policies

We implemented two formal security policies for Xen: (i) a Chinese Wall policy, (ii) a simple Type Enforcement (TE) policy. Both policies work on their own set of types (CW- or TE-types), which are assigned to VMs as a function of the workloads they can run. The CW- and TE-types define the granularity upon which VMs and resources can be distinguished. The assignment of types to VMs and resources is an administrative task (i.e., part of policy management).

Chinese Wall policy: The first policy enables administrators to ensure that certain VMs (and their supported workload types) cannot run on the same hypervisor system at the same time. This is useful to mitigate covert channels or to meet other requirements regarding certain workload types (e.g., workload types of competitors) that shall not run on the same physical system at the same time.

The Chinese Wall policy defines a set Chinese wall types (CW-types), and these are assigned to a VM according to the workloads it can run. It also defines conflict sets using these CW-types and ensures that VMs that are assigned CW-types in the same conflict set never run at the same time on the same system.

Type Enforcement policy: The second policy specifies which running VMs can share resources and which cannot. It supports the coalitions introduced in Section 2.2 by mapping coalition membership onto TE types.

The TE policy defines the set of TE-types (coalitions) and assigns TE types to VMs (coalition membership). The TE policy rules enforce that VMs only share virtual resources if they have a TE type in common, i.e., they are member of at least one common coalition.

4.2 Policy Management

The policy management function is responsible for offering means to create and maintain policy instantiations for the Chinese Wall and Type Enforcement policies. To minimize code complexity inside the hypervisor, the policy management translates an XML-based policy representation into a binary policy representation that is both system-independent and efficient to use by the hypervisor layer.

The binary policy created by the Policy Management includes the assignment of VMs to CW-types and TE-types, as well as the conflict sets to be enforced on the CW-types. No other information is needed by the hypervisor to enforce the policies. The access class of a VM as sHype sees it is exactly a set of CW-types and TE-types. Access classes of virtual resources such as virtual disks comprise only TE-types, typically a single TE-type.

Policy management can either run in a dedicated domain on the managed system (the current Xen approach), or it can run on a separate special-purpose system, such as the Hardware Management Console (HMC) used by PHYP and other commercial virtualization solutions. The policy management is needed to change or validate a policy; it is not necessary to run the system and enforce the instantiated policies.

4.3 Policy Enforcement

Mandatory access control is implemented as a reference monitor. The mediation of references of VMs to shared virtual resources is implemented by inserting *security enforcement hooks* into the code path inside the hypervisor where VMs share virtual resources. Hooks call into the access control module (ACM) for decisions and enforce them locally at the hook. Isolation of individual virtual resources is inherited from Xen since it is a general design issue for hypervisors rather than a security-specific requirement.

4.3.1 Reference Monitor

sHype strictly separates access control enforcement from the access control policy, as in the Flask [33] architecture.

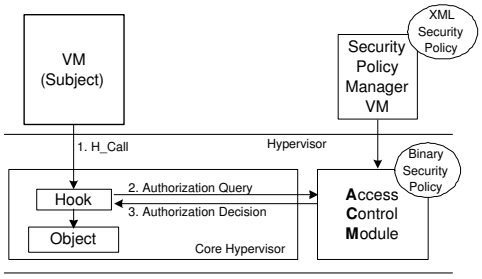

Figure 4. sHype security reference monitor

We describe the control architecture in the context of the hypervisor, but it will also be used in the MAC domains. Figure 4 shows the sHype access control architecture as part of the core hypervisor and depicts the relationships between its three major design components. *Security enforcement hooks* are carefully inserted into the core hypervisor and cover references of VMs to virtual resources. Enforcement hooks retrieve access control decisions from the *access control module* (ACM).

The ACM authorizes access of VMs to resources based on the policy rules and the security labels attached to VMs (CW-types, TE-types) and resources (TE-types). The *formal security policy* defines these access rules as well as the structure and interpretation of security labels for VMs and resources. Finally, a hypervisor interface enables trusted policy-management VMs to manage the ACM security policy.

4.3.2 Access Control Hooks

A *security enforcement hook* is a specialized access enforcement function that guards access to a virtual resource by VMs. It enforces information flow constraints between VMs according to the security policy. Each security hook adheres to the following general pattern: (1) gather access control information (determine VM labels, virtual resource labels, and access operation type); (2) determine access decision by calling the ACM; and (3) enforce access control decision. Hooks are functionally transparent if the access is allowed, and they return an error code otherwise.

Using security hooks, sHype minimizes the interference with the core hypervisor while enforcing the security policy on access to virtual resources. We have placed security enforcement hooks at the following places inside the hypervisor in order to enforce the Chinese Wall and Type Enforcement policies.

- *Domain management operations:* This hook calls into the ACM reporting the security reference of the domain originating the operation and of the domain that is being created, destroyed, saved, restored, migrated, etc. Calls from these hooks are used by the ACM (1) to assign security labels to created domains and to free labels of destroyed domains; (2) to check Chinese Wall conflict sets before creating, resuming, or migrating-in domains; and (3) to adjust the set of running CW-types when destroying, suspending, or migrating-out domains.
- *Event channel operations:* Event-channel hooks mediate the creation and destruction of event channels between domains. The ACM uses calls from these hooks to decide whether the two domains setting up an event channel are members of a common coalition. If the ACM returns a permitted decision, the event channel setup continues beyond the hook. The subsequent sending and receiving of eventsq via the connected channel do not need to be mediated because they would yield the same result (unless the policy changes, see below). If the hook receives a deny decision, the event channel setup is aborted and the hypervisor call returns with an error.
- *Shared memory hook:* Grant-table hypervisor calls allow one VM to grant access to some if its memory pages to another VM. This mechanism (synchronized via event channels) enables efficient communication between VMs running on the same hypervisor. Since the shared memory may in some cases be established dynamically during the communication (e.g., sending and receiving network packets or reading and writing from virtual disks), the security hook guarding this operation may be on the performance critical path.

Decision caching. Since neither the event channel nor the shared memory hook calls induce any state change in the ACM, we use *caching* of access control decisions to minimize the overhead introduced by the security hooks calling into the ACM and the ACM authorizing access.

We cache access control decisions locally in the data structures involved in a grant-table or event-channel operation the first time an access control decision is required between two VMs. The decision cache is not used for domain operation hooks because the ACM must be aware of these calls to update its security state. We are experimenting with multiple cache layouts to find the best trade-off between memory requirements and lookup speed.

Decision caching achieves near-zero overhead on the critical path at the cost of additional management and complexity. When a VM is destroyed or migrated, all cache entries regarding this VM must be cleared. The overhead of clearing these caches is very low.

Policy Changes. When the policy changes, we must explicitly revoke a shared resource from a VM that is no longer authorized to use it. Since we use extensive caching, we must propagate access authorization changes into the

caches of VMs. Additionally, we define a re-evaluation function for both event-channel and grant-table hooks because these hooks check permissions only when an event-channel or a shared memory area is set up, and not when it is used. When invoked by the ACM, the re-evaluation function (1) re-evaluates the original access control decision, and (2) revokes shared resources in case the authorization is no longer given.

Revocation of event-channels from inside Xen is straightforward. VMs trying to use revoked event-channels will receive error codes which must be handled regardless of access control. Memory shared between VMs will typically not be directly handed over by the Guest OS to applications but rather used exclusively inside device drivers. Consequently, device drivers might run into a memory access fault when trying to send a request via shared memory to which their access was revoked. We are currently working on a call-back mechanism, initiated by the hypervisor, so that revoked shared memory can be reported to affected VMs and handled there in an more controlled fashion, allowing for more graceful failure.

4.3.3 Access Control Module (ACM)

The ACM maintains policy state, makes policy decisions based on the current policy, interacts with the policy manager VM to establish a security policy, and triggers callback functions to re-evaluate access control decisions in the hypervisor when the policy changes.

The ACM stores all security policy information locally in the hypervisor, and supports policy management through a privileged hypervisor call interface. This interface is access-controlled by a specialized hook and will only be accessible by policy-management-privileged domains.

During domain operations, the ACM is called by security hooks and allocates and de-allocates security labels for created and destroyed domains according to the policy. These labels are used for access control decisions. The virtual machine configuration includes references for the ACM that are used to determine the label for a newly created domain. In our example, such a label consists of a set of TE-types and a set of CW-types, as described in Section 4.1.

The ACM maintains the policy state needed to enforce the Chinese Wall policy. For this purpose, the ACM maintains a *Running CW Types* array indexed by the CW-type and containing a reference count that describes the number of running domains that have this CW-type. Whenever a domain is started, the ACM determines those conflict sets with which this domain shares a CW-type. Then it verifies if any of the other CW-types of these conflict sets is running. If any of these CW-types' reference count is non-zero, then we have a Chinese Wall conflict and the current domain is not permitted to start. Otherwise, the current domain is permitted to start and the *Running CW Types*' reference counts are incremented for those CW types that are assigned to the started domain. If a domain is destroyed, the *Running CW Types*' reference counts of this virtual machine's CW-types are decremented.

Access control decisions for the Type Enforcement policy are simple. The ACM looks up the Coalition set of those domains that are trying to establish an event channel or shared memory. If both domains share a common TE type (coalition), then the access is permitted. Otherwise it is denied. It can be implemented as an n-bit AND operation over the TE-type vectors of the domains where n is the number of known TE types (coalitions).

4.4 MAC domains

MAC domains enable multiple coalitions to share a real resource by creating isolated virtual resources based on the real resource (recall the `vdisk` device domain in Figure 2). If sufficient hardware resources are available and coalitions don't need to cooperate on higher layers, MAC domains are not necessary because hardware can be exclusively assigned to a single coalition and no VM needs to participate in multiple coalitions. We sketch briefly how we envision MAC domains to work. They must offer the following guarantees in order to conform to reference monitor requirements:

1. Isolate exported virtual resources (e.g., the two virtual disks for the *Order* and the *Advertising* coalition) inside the MAC domain at least as well as the hypervisor isolates its virtual resources.
2. Control access of VMs to those resources according to the Type Enforcement Policy (i.e., only allow VMs that are members of the coalition to which the virtual resource is assigned to access the resource).

The isolation property can be achieved using mandatory access control inside the MAC domain, e.g., using SELinux.

The access control property requires a MAC domain to discover the coalition membership (TE types) of the requesting domain. For this reason, sHype offers to MAC domains a hypervisor call that returns the coalition membership information of a connected domain using the protected policy information of the ACM. The hypervisor will return those coalitions (TE types) of which both the MAC domain and the requesting VM are members. Based on this information, the MAC domain permits access of the requesting VM only to virtual resources that share membership in the same coalition(s).

Multi-coalition VMs, besides implementing the sharing of hardware resources among coalitions, also form the natural environment in which controlled sharing between coalitions on higher layers and with finer granularity can be implemented (e.g., with file and operation granularity based on OS-level MAC policies such as SELinux policies).

While sHype forms isolated coalitions and restricts sharing to multi-coalition VMs, these VMs can overcome this isolation in carefully designed and trustworthy environments to fulfill application requirements.

5 Evaluation

5.1 sHype-Covered Resources

Figure 5 shows the virtualized resources sorted according to where they are implemented. The `TCB coverage` column shows how well their isolation and mandatory access control is covered by the sHype reference monitor. We distinguish whether the implementing entity is serving a `single` coalition or `multiple` coalitions since the latter requires MAC control.

resource / implementation	event channel	shared memory	virtual disk	virtual TTY	virtual LAN	TCB coverage single / multi
Hypervisor	X	X				● / ◐
local VM			X	X	X	● / ◐
VMs on multiple systems					X	◔ / ◔

●..fully covered by sHype ◐..partly covered by sHype

Figure 5. Current resource coverage in Xen

If event channels, shared memory, virtual disks, virtual TTY, or vLANs are shared within a single coalition, sHype fully covers the TCB for sharing between coalitions. While the sHype architecture is comprehensive and its policy enforcement covers the communication between domains, sHype relies on MAC domains to correctly isolate virtual devices from each other (see Section 4.4). Such multi-coalition MAC domains are necessary if real peripherals must be shared between multiple coalitions or if different coalitions shall be able to co-operate using filtering and fine-granular access control implemented inside a MAC domain.

If virtual resources (e.g. vLANs) are distributed over multiple hypervisor systems and communicate over a network, sHype relies on the domains bridging those systems (MAC bridging domains) to securely isolate the vLAN traffic from other traffic on the connecting network and to control access of VMs on the connected systems to the vLAN. In consequence, sHype controls which domains are able to connect to MAC-bridging domains but defers isolation and MAC guarantees for vLAN traffic to these MAC-bridging domains.

5.2 Code Impact

The sHype access control architecture for Xen comprises 2600 lines of code. We inserted three MAC security hooks into Xen hypervisor files to control domain operations, event channel setup, and shared memory setup. Two out of three hooks are off the performance critical path. One hook (shared memory setup) can be on or off the critical path depending on how shared memory is used by a domain. We implemented a generic interface (akin to the Linux Security Modules interface but much simpler) upon which various policies can be implemented. We have implemented the Chinese Wall and the Type Enforcement policies for Xen as well as the caching of event-channel and grant-table access decisions. Maintaining sHype within the evolving Xen hypervisor code base has proven easy.

5.3 Performance

By performing authorization only at bind time and by caching those decisions, sHype aims to introduce minimal overhead on the performance-critical path. Policy changes happen rarely and therefore the related overhead is not on the critical path. Similarly, since Chinese Wall hooks are invoked only during domain operations (e.g, create), they are also not on the critical path. We ran experiments to measure the overhead of Type Enforcement hooks that are invoked when VMs communicate through the Xen event channel and grant table mechanisms.

In our experiments, we ran the management domain (Dom0) and one user domain (DomU), both with Fedora Core 4 Linux installations, on a current uniprocessor desktop system. We assigned common Type Enforcement and Chinese Wall types to Dom0 and DomU. We assigned DomU a physical disk partition (`hda7`) that is managed by Dom0 and mounted by DomU through the Xen virtual block interface. The experiment made 10 transfers of 10^8 disk blocks from Dom0 through the virtual block interface to DomU (`dd if=/dev/hda7 of=/dev/null count=10000000`). Shared-memory grant tables were dynamically set up between Dom0 and DomU when transferring the disk blocks. When we activated the Type Enforcement policy, the 10 transfers invoked the grant-table hook approximately $12 * 10^6$ times, and took between 1196 and 1198 seconds to complete.

Using this time-to-completion metric, we did not observe any overhead. The performance was identical for configurations that did not invoke any hooks (null policy) and for configurations that did invoke hooks (TE policy).

6 Related Work

While there have been instances of highly secure operating systems that have been successfully commercialized

–e.g., GEMSOS [32, 29], KSOS [24], or Multics [4, 18]– their widespread use has not come about. The huge design, development, and evaluation cost proved justified only for specialized application domains with very high security requirements. Access control with process and file granularity in general-purpose OSs, while possible, is very complex as illustrated by SELinux [26] policies. Expressing and enforcing a simple TCB model in general-purpose OSs is very difficult due to interdependencies between processes [17]. VMMs can supplement OS security and provide confinement in case OS security controls fail [23].

Gold et al. [11] demonstrated that virtualization of a single hardware platform enabled the execution of multiple virtual systems, each running at a single security level, so that those virtual systems were strongly isolated from each other. The prevalent approach to creating multiple virtual machines on a single real hardware platform is to use a VMM. [12].

Based on VMs, a single system can implement a multi-level secure system by dividing it into multiple single-level virtual systems, guaranteeing secure separation. Separation Kernels are VMMs that completely isolate virtual machines. Rushby [27] proved that complete isolation and separation of VMs is possible. Based on Rushby's work, Kelem et al. [21] derived a formal model for Separation VMMs. One example of a more recent separation kernel design based on virtualization is NetTop [25]. NetTop implements virtual systems that are isolated from each other on a single hardware platform to allow processing of data belonging to multiple sensitivity levels on a single system.

Recognizing that strictly separated VMs do not map well into cooperating distributed applications, some research has examined kernels that enabled secure sharing between VMs. However, these secure-sharing VMM approaches [19, 11] tend to suffer from high performance overhead as well as large trusted computing bases due to necessary I/O emulation inside the hypervisor layer. Karger et al. [20] report for the KVM approach a 50-90% overhead (limited performance tuning) as compared to VM/370 plus the effort of rewriting 50% of the VMM code; and for the VaxVMM approach a 10-70% overhead (no performance tuning, including virtualization overhead) as compared to the native VMS operating system plus writing the entire VMM code (no retrofit).

Our sHype hypervisor security architecture is motivated by these prior secure VMM systems to adequately address performance overhead issues and to strive for minimal design / code modifications in modern hypervisors that are targeted for the medium-assurance commercial environment. Experience with initial sHype prototypes in multiple hypervisors is very promising in this regard, but will require vetting against enterprise workloads using standardized benchmarks as these initiatives and our architecture mature.

Today, a number of virtualization technologies are deployed successfully in the commercial domain, such as PHYP [13] and VMWare [34]. There are also several promising research VMMs, such as Terra [10], Xen [3], and the IBM Research Hypervisor [14]. All of these offer a basis for broad application of sHype, while none were built for the highest levels of assurance, nor do any use the KVM or VaxVMM approaches.

Micro-kernel system architectures also struggled with the problem of determining how to control access to system resources. Some systems focus on minimality, forgoing all but the most basic security. Others concentrate system-wide security features in the kernel. Notable examples include EROS [31], L4 [22], and Exokernel [9].

In summary, the sHype approach –targeting the commercial hypervisor space– is supplementary to existing secure operating system approaches and orthogonal to existing secure hypervisor approaches.

7 Conclusion

We presented a secure hypervisor architecture, sHype, that we have successfully implemented in the Xen open-source hypervisor. It can be downloaded as part of the Xen distribution [35]. We showed how access control in the hypervisor can be implemented in a way that has very low impact on VM performance and is non-intrusive to existing VMM code.

The hypervisor layer is becoming a standard component in system software. With its coarse-grained resource management, protection against workloads, and relatively small footprint, a hypervisor proved the ideal vehicle for implementing a flexible security framework that supports a range of security policies.

Currently, we are extending our security architecture to cover multiple hardware platforms – involving policy agreements and the protection of information flows that leave the control of the local hypervisor. We need to establish trust into the semantics and enforcement of the security policy governing the remote hypervisor system before allowing information flow to and from such a system. To this end, we are experimenting with establishing this trust based on the Trusted Computing Group's Trusted Platform Module [1] and the related Integrity Measurement Architecture [28].

While Xen separates device drivers and management functions from Dom0 into their own domains, we are experimenting with MAC domains for sharing limited physical resources, e.g., in the mid-range server and desktop space. Future work includes the accurate accounting of resource use, and generating audit trails appropriate for medium-assurance Common Criteria evaluation targets.

References

[1] TCG TPM Specification Version 1.2. http://www.trustedcomputinggroup.org.

[2] Advanced Micro Devices. AMD64 Virtualization Codenamed "Pacifica" Technology, Secure Virtual Machine Architecture Reference Manual, Rev 3.01, May 2005. http://www.amd.com/us-en/assets/ content_type/white_papers_and_tech_docs/33047.pdf.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, October 2003.

[4] D. E. Bell and L. J. LaPadula. Secure computer systems: Unified exposition and multics interpretation. Technical report, MITRE MTR-2997, March 1976.

[5] K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report MTR-3153, Mitre Corporation, Mitre Corp, Bedford MA, June 1975.

[6] W. E. Boebert and R. Y. Kain. A practical alternative to hierarchical integrity policies. *8th National Computer Security Conference*, 1985.

[7] D. F. C. Brewer and M. J. Nash. The Chinese Wall Security Policy. *Proc. IEEE Symposium on Security and Privacy*, pages 206–214, May 1989.

[8] Common Criteria. Common Criteria for Information Technology Security Evaluation. http://www.commoncriteriaportal.org.

[9] D. Engler, M. Kaashoek, and J. J. O'Toole. Exokernel: An operating system architecture for application-level resource management. *Proceedings of the* 15^th *Symposium on Operating System Principles*, 1995.

[10] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *Proc. 9th ACM Symposium on Operating Systems Principles*, pages 193–206, 2003.

[11] B. D. Gold, R. R. Linde, and P. F. Cudney. KVM/370 in Retrospect. In *Proc. IEEE Symposium on Security and Privacy*, 1984.

[12] R. P. Goldberg. Survey of Virtual Machine Research. *IEEE Computer Magazine*, 7(6):34–45, 1974.

[13] IBM. PHYP: Converged POWER Hypervisor Firmware for pSeries and iSeries. http://www-1.ibm.com/servers/enable/site/peducation/ abstracts/ abs_2bb2.html.

[14] IBM Research. The Research Hypervisor – A Multi-Platform, Multi-Purpose Research Hypervisor. http://www.research.ibm.com/ hypervisor.

[15] Intel. Intel Virtualization Technology Specification for the IA-32 Intel Architecture, April 2005. ftp://download.intel.com/ technology/ computing/ vptech/C97063-002.pdf.

[16] J. P. Anderson et. al. Computer security technology planning study. Technical Report ESD-TR-73-51, Vol. I+II, Air Force Systems Command, USAF, 1972.

[17] T. Jaeger, R. Sailer, and X. Zhang. Analyzing Integrity Protection in the SELinux Example Policy. In *12th USENIX Security Symposium*. USENIX, 2003.

[18] P. A. Karger and R. R. Schell. Thirty Years Later: Lessons from the Multics Security Evaluation. In *Annual Computer Security Applications Conference (ACSAC)*, December 2004.

[19] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn. A VMM Security Kernel for the VAX Architecture. In *Proc. IEEE Symposium on Security and Privacy*, May 1990.

[20] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn. A Retrospective on the VAX VMM Security Kernel. In *IEEE Transaction on Software Engineering*, November 1991.

[21] N. L. Kelem and R. J. Feiertag. A Separation Model for Virtual Machine Monitors. In *Proc. IEEE Symposium on Security and Privacy*, 1991.

[22] J. Liedtke. On μ-kernel construction. *Proceedings of the* 15^th *Symposium on Operating System Principles*, 1995.

[23] S. E. Madnick and J. J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. *Proceedings of the ACM workshop on virtual computer systems*, 1973.

[24] E. J. McCauley and P. J. Drongowski. KSOS – The design of a secure operating system. In *Proc. In AFIPS Conference*, pages 345–353, 1979.

[25] R. Meushaw and D. Simard. NetTop - Commercial Technology in High Assurance Applications. *Tech Trend Notes*, Fall 2000.

[26] National Security Agency. Security-Enhanced Linux (SELinux). http://www.nsa.gov/selinux.

[27] J. Rushby. Proof of Separability—A verification technique for a class of security kernels. In *Proc. 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 352–367, Turin, Italy, 1982. Springer-Verlag.

[28] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Thirteenth USENIX Security Symposium*, pages 223–238, August 2004.

[29] R. R. Schell, T. F. Tao, and M. Heckman. Designing the GEMSOS Security Kernel for Security and Performance. *8th National Computer Security Conference*, pages 108–119, 1985.

[30] H. Scherzer, R. Canetti, P. A. Karger, H. Krawczyk, T. Rabin, and D. C. Toll. Authenticating Mandatory Access Controls and Preserving Privacy for a High-Assurance Smart Card. In *(ESORICS)*, 2003.

[31] J. Shapiro, J. Smith, and D. Farber. EROS: A fast capability system. *Proceedings of the* 17^th *Symposium on Operating System Principles*, 1999.

[32] W. R. Shockley, T. F. Tao, and M. F. Thompson. An Overview of the GEMSOS Class A1 Technology and Application Experience. *11th National Computer Security Conference*, pages 238–245, October 1988.

[33] R. Spencer, P. Loscocco, S. Smalley, M. Hibler, D. Anderson, and J. Lepreau. The Flask Security Architecture: System support for diverse security policies. In *Proceedings of The Eighth USENIX Security Symposium*, August 1999.

[34] VMware. http://www.vmware.com/.

[35] XenSource. http://xenbits.xensource.com/xen-unstable.hg.