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)

Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)
1063-9527/05 $20.00 © 2005 IEEE

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.

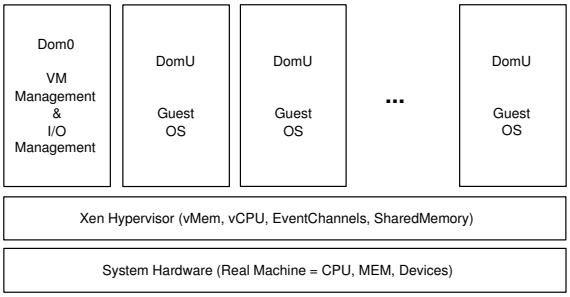

Figure 1. Xen hypervisor architecture

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.

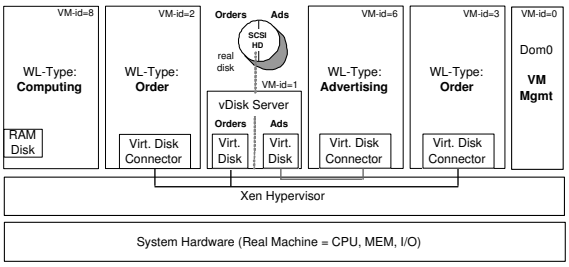

Figure 2. VM coalitions and payloads in Xen

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

Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005)
1063-9527/05 $20.00 © 2005 IEEE

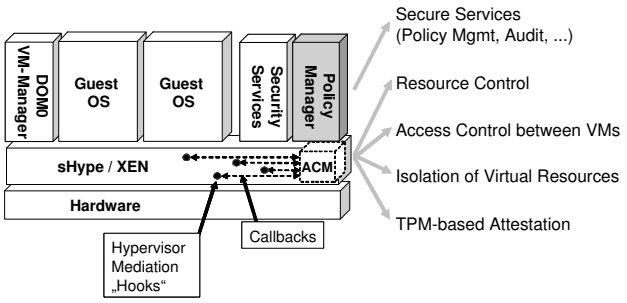

Figure 3. sHype architecture

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.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.