RC24590 (W0806-101) June 23, 2008
Computer Science

IBM Research Report

Virtualization and Hardware-Based Security

Ronald Perez, Leendert van Doorn, Reiner Sailer
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Virtualization and Hardware-Based Security

Ronald Perez, Leendert van Doorn, and Reiner Sailer

Abstract

Hypervisor-based virtualization technologies are frequently mentioned in regard to both their security-related strengths and their weaknesses. We examine emerging hardware and software virtualization technologies in the context of modern computing environments and requirements, keeping in mind the history of these technologies in order to fully appreciate them as we also outline future directions in this space.

I. Introduction

Virtualization is the process of presenting something as being genuine when in fact it is not – e.g., virtual reality. Virtualization in the computer architecture domain is the presentation of an environment to one layer in an information technology stack which abstracts or is representative of a layer below. The insertion of this layer of indirection between existing layers in a hardware and/or software stack is typically done to address specific problems such as providing support for legacy functionality, standardizing interfaces on logical models, or transparently load balancing usage of shared resources.

Virtualization may involve language-level runtimes providing high-level abstract architectures for applications, or a thin hardware virtualization layer of software situated between system hardware and the operating system layer providing logical views to physical resources. In either case, the primary justification for virtualization is efficiency, such as the efficient use of programming resources achieved with "write once, run everywhere" language runtimes in the first case, or the efficient use of hardware resources that can be gained with the thin hardware virtualization layer in the second case. While virtualization comes in many forms, including process, storage, and network virtualization, this paper focuses on security and hardware support for that thin hardware virtualization layer, often termed a *virtual machine monitor* or *hypervisor*.

II. Hypervisor Virtualization

Early virtual machines were essentially computing environments that simulated or emulated the existing hardware features of the host system while arbitrating access to shared system resources. This allowed multiple instances of operating systems to run on the same system where each operating system was originally designed to be the sole arbiter of system resources. This was important because systems of the time – largely mainframes that were designed during the mid 1960s and 1970s, such as the predecessors to the IBM VM/370 [12]– were expensive and there were distinct economic advantages in being able to partition one physical system into multiple logical systems. The same advantages exist today for the heirs to the IBM VM/370, such as the IBM PR/SM and z/VM virtual machine monitors and the S/390 and zSeries mainframes [13].

While hypervisors associated with high-end systems have continued to evolve and remain critical to maximizing utilization of these highly reliable and secure systems, increasing the utilization of mid-range and high-volume systems has also become a factor driving hypervisor development and deployment. These mid-range and high-volume systems have developed increased electrical power requirements that rival increases in computing power and overall performance. The expense involved with housing and managing these systems has not kept pace with the decline of raw hardware and software resource costs including the increasing cost of managing the security of ever more complex systems. Therefore, hypervisors have become an attractive option for large datacenters and medium to small enterprises.

In addition to increased utilization, the general flexibility afforded by modern hypervisors and their support of systems that have become relative commodities is also apparent. Continuous or high-availability requirements can potentially be achieved with greater ease and less cost with the ability to checkpoint and replicate virtual machines in a distributed systems environment. Emerging tool chains are enabling the rapid provisioning of workloads to virtual machines and the ability to essentially load-balance an enterprise's entire infrastructure by migrating existing virtual machines and their workloads to more appropriate systems in support of system maintenance cycles or to free-up resources for critical workloads or peak usage patterns.

A. Hypervisors and Security

There are a few basic but very strong security primitives for which hypervisors are particularly well suited, namely separation and controlled sharing. Separation can be achieved through the use of different hardware facilities for different workloads (physical separation), running workloads at different times (temporal separation), cryptographically protecting data specific to each workload (cryptographic separation), and logical separation or isolation, involving the use of a reference monitor [1] (Figure 1) or security kernel to separate workloads and the resources they use. Hypervisors function as reference monitors, providing workload isolation

on an operating system instance granularity (as opposed to operating systems, which strive to provide isolation at the process level).

The role of the reference monitor in secure systems designs is to mediate all security sensitive operations such as access to objects or communications between subjects. In hypervisors, objects and subjects are respectively system resources and virtual machines. The long recognized basic requirements for a reference monitor are that it must be small enough to be fully-tested and analyzed, it is relatively immune from compromise, and it is impossible to bypass the reference monitor's mediation functionality. The hypervisor is also a key element in the trusted computing base of the entire system – the hardware, firmware and software components in layered system architectures the correctness of which are necessary to enforce the explicit or implicit system security policy.

Because the hypervisor reference monitor mediates access to and between coarse grained entities – e.g., processors, memory (in addition to memory management that is performed at the operating system level), disks and virtual machines – they are often orders of magnitude smaller in size and less complex than modern operating systems. Thus they are less difficult to test and analyze for correctness and to protect from compromise. Additional protection is often provided for hypervisors through hardware features such as privileged operating modes and protected memory support. Both, along with hardware architected hypervisor calls that are similar to operating system calls, ensure that the hypervisor mediation functionality cannot be bypassed.

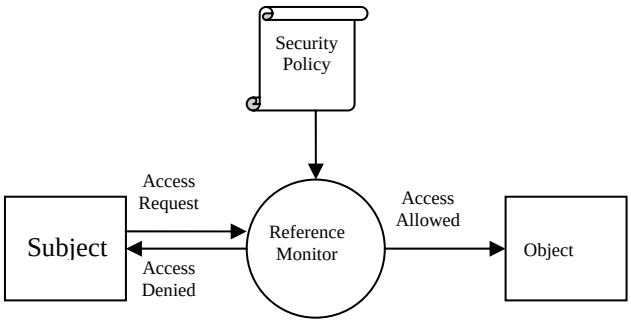

Figure 1. Reference Monitor

Hypervisors such as those associated with early and current mainframe systems have traditionally supported strong isolation or separation of virtual machines and their workloads, including fault isolation – limiting the effects of an application or operating system fault within a virtual machine. While administrators have had the ability to explicitly configure these systems to support sharing of system resources and communications between virtual machines, hypervisors that support sharing based on explicit security policies and labels associated with each virtual machine and its resources, such as the DEC VAX VMM [14] and the KVM/370 [15] security enhanced version of the IBM VM/370, were not available on a large scale perhaps due in part to the commercial requirements of the day and the high assurance government-/military-oriented design targets for these systems. However, policy-driven controlled sharing requirements for commercial hypervisors on mid-range and high-volume systems are increasingly becoming an issue because of the need to maintain isolation while at the same time maximizing resource utilization and support for the more distributed and interconnected nature of contemporary workloads.

The performance impact of security functionality has always been an issue with applications, operating systems and other information technology infrastructure. Whether simple separation or controlled sharing is required, the performance overhead associated with security functionality has and will continue to have a major role in the acceptance of hypervisor security. Keeping the code size of the hypervisor relatively small, largely due to restricting functionality and complexity in the spirit of a trusted computing base, and limiting higher-level hypervisor management and flexibility only where it is necessary to enforce security requirements is key to minimizing the performance impact due to security. Hardware support, such as management of memory and other system and processor resources and for accelerated context switching between protection modes, is even more critical to maximizing overall system performance.

III. Hardware Virtualization Support

Despite the obvious advantages of virtualization such as consolidation and isolation, the major disadvantage of virtualization is its large performance overhead. This is especially true when interpretative or emulation techniques are used. A single emulated machine instruction can easily expand to thousands of real instructions and may cause significant performance degradation. To counter this, CPU manufacturers have been developing hardware support for virtualization where part or all of the emulation takes place in the CPU itself.

All major vendors are currently working on virtualization capabilities for their systems. This includes virtualization capabilities for the CPU, I/O, and some specialized devices. Some of these are examined in greater detail in the following section.

A. Architectural Overview

A traditional computer system consists of memory (RAM), a CPU, an I/O controller (typically a PCI bridge) and one or more I/O devices such as a disk, network and video controllers. Virtualizing the shared resources in this architecture, as shown in Figure 2, involves making changes to the CPU and the bus controller that arbitrates accesses to the I/O bus. The I/O devices themselves may also be virtualization aware but that is beyond the scope of this paper.

A typical CPU consists of many shared resources. Examples of these are interrupt vectors, page tables, interrupt controllers, timers, and special descriptor tables. It is the task of the hypervisor to virtualize these resources, but its job can

be considerably easier when the right processor abstractions are available [21].

The hypervisor has the task to provide the illusion of a virtual machine to the guest operating system that is running inside the VM. This involves managing physical memory, interrupts, faults, and I/O devices.

B. Intel Vanderpool and AMD Pacifica Technology

Until recently, hardware virtualization support was only available on mainframe computers but now some of these capabilities have been integrated into modern processors. Both Intel and Advanced Micro Devices (AMD) have designed and implemented their own virtualization extensions. Intel's extensions are called Vanderpool Technology (VT) [5] and AMD's are called SVM [6].

Both VT and SVM roughly provide the same functionality. They all create a container which is a virtual CPU. Inside the container you can run an unmodified OS that is unaware of the fact that it is operating in a controlled environment. The only way out of this container is under well defined conditions, known as *exits*. Exits are generated whenever the code in the container executes privileged instructions[1], such as changing the CPU state, changing the page tables, or causing a page fault. These exits cause a trap into the underlying hypervisor that executes in the root container. It is then up to the hypervisor to emulate the correct behavior such that the OS running inside the container is unaware that it is being virtualized.

For example, for an operating system running inside a container to get access to a specific physical memory location, it has to create a mapping from a virtual to physical address in the page table structure. The OS then needs to activate the new page table by assigning it to a special CPU register (cr3, on x86 type processors). The assignment to this special register causes an exit. It is now up to the hypervisor to validate the new page table structure, check that the physical memory addresses are really assigned to this container, instantiate the new page table and continue execution in the container after the assignment instruction.

The approach taken in the Power Architecture™ server processors from IBM (e.g., POWER5™) [9] is different from the direction taken by Intel and AMD. Rather than introducing a heavy weight container and exit concept, the POWER processor introduces a new hypervisor state (akin to user and supervisor state) and duplicates certain key control registers in hypervisor state that operate independently from their supervisor state counter parts. In a way it is the Complex Instruction Set Computer (CISC) versus Reduced Instruction Set Computing (RISC) approach. POWER introduced a few lightweight concepts to support virtualization.

C. IOMMU

There is more to virtualization than just virtualizing the CPU, I/O is another shared resource. The goal of I/O virtualization is to give a virtual machine direct device access such that it cannot overwrite other virtual machines. For most devices, this is not a problem, but some have bus master capabilities that need to be controlled. A bus master capable device can initiate its own memory transfers and write to every memory location in the system, including memory that is not assigned to the virtual machine that controls the device.

This is a common problem. This is why CPUs have an indirection layer and provide a virtual to physical memory map abstraction. This mapping is provided by the Memory Management Unit (MMU) that is part of the CPU. For I/O something analogous is used, the IOMMU[18][9]. It is typically part of the bus controller (see Figure 2). The details will depend on the specific design, but typically, each I/O device has its own address translation map.

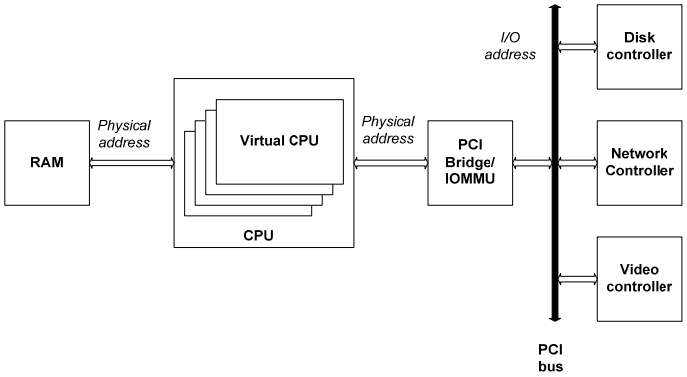

Figure 2. Virtualized System Architecture Overview

The IOMMU maps I/O virtual addresses to physical addresses. Whenever a device initiates a memory transfer, the I/O virtual memory address is first translated into the physical memory address before the memory transfer commences. If the hypervisor ensures that the memory mapping for the virtual machine corresponds with the IOMMU mappings for the devices the virtual machine owns, then the virtual machine can directly interact with its I/O devices without affecting other virtual machines.

D. Intel LaGrande and AMD Presidio Technology

In 2002, Microsoft announced its Palladium initiative, a project aimed at providing a secure client foundation for its next version of Windows. For trademark reasons the initiative was quickly renamed to the Next Generation Secure Computing Base (NGSCB) and it aimed to provide the following guarantees:

- Process isolation
- Sealed storage
- Platform attestation
- Secure I/O paths

With NGSCB Microsoft tried to define an environment that was protected from malicious software and malicious peripheral cards. Unfortunately, the NGSCB project was cancelled in mid 2004 for lack of customer traction. Still, both Intel and AMD had rallied around this initiative. Both defined platforms that embedded these guarantees into their systems

[1] Privileged instructions are used to control critical system resources and can only be executed by the operating system kernel.

and both based it on their virtualization technology.

Intel coined its secure computing platform *Lagrande Technology* or LT for short [7]. LT consists of a VT core to provide the process isolation, special keyboard and video capabilities for the secure I/O paths, a DMA exclusion vector to isolate I/O devices from the security kernel, and tight integration with the Trusted Platform Module version 1.2 specification to provide sealed storage and platform attestation.

AMD called their secure computing platform *Presidio* [8] and although it provides the same high-level functionality, its technology roadmap is quite different from Intel's. Unlike Intel, AMD decided to integrate the trusted platform module (TPM) capabilities and the DMA exclusion capabilities directly into SVM. Consequently, Presidio consists of SVM and a set of secure I/O capabilities.

Both Intel and AMD added two new features to their CPUs/chipsets. One of them, the DMA exclusion vector, is an elementary version of an IOMMU. It provides protection from rogue DMA, but no address translation. This function will eventually disappear and be subsumed by the IOMMU itself.

The second enhancement is the introduction of the *dynamic root of trust* as defined by the Trusted Computing Group [3]. Traditional secure or authenticated boot designs start with the assumption that the system is unmodified and preserve this guarantee during the bootstrap into the operating system. The dynamic root of trust design, on the other hand, enables software to securely initialize the system at any point in time. That is, even when the system is already running an operating system and its applications. For this, the vendors added new instructions to the CPU. Intel introduced the SENTER instruction [7] and AMD the SKINIT instruction [6]. Both are conceptually similar. For this discussion we describe the SKINIT behavior.

Upon the execution of an SKINIT instruction, the processor is reinitialized into a well-known state in which it can execute a secure loader such that the loader cannot be tampered with during its execution. This well-know state ensures that the interrupts are disabled, DMA to the memory area where the secure loader is located is inhibited, and that the special purpose registers that control memory accesses are initialized to safe values. This environment guarantees that other programs running on the CPU or external devices cannot modify the loader while it is running.

Once the processor has been reinitialized, a secure hash of the 64KB loader is send to the TPM and stored in PCR[17]. This specific PCR can only be written by the CPU using special "locality" bus cycles that cannot be generated from software. This ensures that only the CPU SKINIT instruction could have generated this hash value. The hash value of the secure loader constitutes the dynamic root of trust. As soon as the hash of the loader is stored inside the TPM, control is transferred to the loader. The loader is arbitrary code, but it could for example, measure the rest of the system, store the result inside the TPM and resume execution where the operating system that invoked SKINIT left off. These measurements can then be used to unseal storage or attest to remote parties the software stack that is running and the hardware platform that it is running on.

The more challenging aspects of LT and Presidio are the secure I/O capabilities. The first aspect of this, secure input, is straightforward. The perceived threat is one where an adversary taps or modifies the communication from the keyboard and mouse into the secure environment. To prevent this, the communication from the keyboard to the recipient of the data, a keyboard device driver in a secure kernel, is encrypted.

Secure output such as video suffers from the same communication threats, but it also suffers from Trojan horse attacks[2]. How can a user distinguish between the output on the screen from a secure and insecure environment? Despite good research in this area, no good practical solutions exist for this yet. Intel has proposed a solution where by the secure environment always displays on top of the current screen and can be activated through a secure attention key mechanism. Others have suggested that a pass-phrase that is only available to secure kernel and user is rendered into the window's background to convey to the user that she is interacting with a secure environment. This too may prove too cumbersome for the end user to handle. While everyone agrees secure user I/O is a critical part to a secure client system, there is not much consensus in the industry what form that should take.

IV. sHype – Hypervisor Security Architecture

We illustrate the sHype [4] security architecture and its integration into a Virtual Machine Monitor environment in Figure 3.

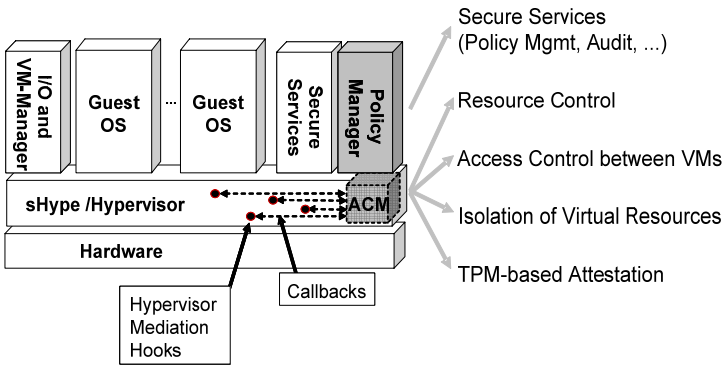

Figure 3. sHype Hypervisor Security Architecture

sHype is implemented in various stages for multiple hypervisors including the Xen [11] Opensource hypervisor. Building upon the previously described hardware support, the major design goal for sHype is to establish a secure foundation for server platforms, providing functions such as:

- *Strong isolation and mediated sharing* between Virtual Machines, strictly controlled by a flexible access control enforcement engine.

[2] A Trojan horse is a malicious program that masquerades itself as a legitimate program by, for example, presenting the same user interface.

- *Attestation and integrity guarantees* for the hypervisor and its virtual machines, supported by a virtual TPM architecture. TPM-based attestation [16] provides the ability to generate and report properties of the running system.
- *Resource control and accurate accounting guarantees*, enabling the enforcement of quality of service agreements between service provider and consumer.
- *Secure Services*, providing the base infrastructure in sHype for refining complex monolithic run-times by moving services such as security policy management or distributed auditing into their own, carefully protected VM.

The sHype access control framework and the virtual TPM architecture enabling attestation of individual virtual machines are described in more detail in the following sections. Those technologies form the basis for the Trusted Virtual Datacenter [19], a new technology developed by IBM Research that simplifies consistent and strong isolation guarantees in distributed virtualized datacenters.

A. sHype Access control Architecture

The sHype access control framework is designed (a) to ensure isolation of virtual machines by default and (b) to control sharing of resources between virtual machines (implicit isolation / explicit sharing). sHype enforces a formal security policy (mandatory access control, MAC) on information flow between virtual machines independently of generic user virtual machines. It leverages existing isolation between virtual resources and extends it with MAC features to control shared resources.

sHype moves the state of information sharing from being defined by a number of ad-hoc system administrator decisions to one that is formally defined by the security policy and enforced by sHype independently of guest virtual machines.

The major components of the access control architecture are the Policy Manager and the access control policy, the Access Control Module (ACM), and the mediation hooks (see

Figure 3). The Policy Manager maintains the hypervisor security policy, which defines the rules the ACM uses to decide which Virtual Machines can access which resources. The Policy Manager is implemented inside a special-purpose virtual machine to keep related complexity out of the hypervisor. It provides the ACM inside the hypervisor with a pre-compiled security policy. Security mediation hooks mediate access to all resources inside the hypervisor that enable information flow from one virtual machine to another. Security hooks are placed both within the hypervisor to mediate direct VM-to-VM sharing, and within the I/O and VM-Manager VM to mediate access to virtualized resources.

Each security hook implements a redirection of access of virtual machines to shared virtual resources implemented inside the hypervisor or within the I/O hosting VM. This redirection code either behaves transparently (permits) or aborts (denies) the access request depending on the result of a callback into the ACM. The callback returns the access control decision of the ACM depending on the active security policy, the type of security hook, and the security labels of VMs or resources participating in the mediated event.

Many virtual machine monitors do not offer sufficient information inside the hypervisor to distinguish the direction of information flow or the semantics of higher level operations inducing such an information flow. Therefore, sHype aims at coarse-grained but very robust and simple access control on VMs and resources within the hypervisor and defers finer-grained access control to higher layers (guest OS, middle-ware, applications). It promotes the layering of multiple security policies over a monolithic one. To this end, sHype explicitly supports an interface that enables higher layer access control functions in authorized virtual machines (e.g., OS security mechanisms) to retrieve sHype security control information on virtual machines and resources. Higher layer policies take advantage of lower layer security policies and focus on refinement rather than re-implementation [23],[24]. This strategy can be compared to communication stacks, where higher layers usually rely on lower layer functions to bridge physical differences (medium access, layer 2) or to limit network exposure (layer 3, IP fire-walling). Yet they sometimes decide to re-implement some of the lower layer functions (e.g., error checks) in higher layers based on additional information that is not available to the lower layer functions.

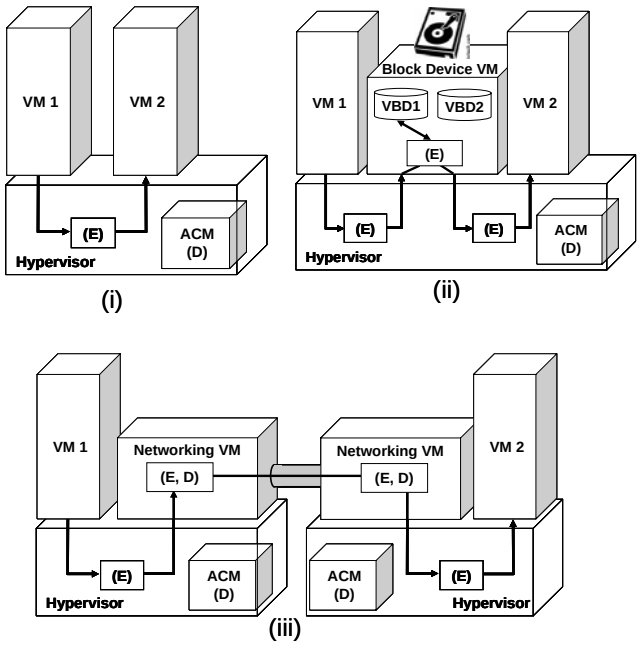

Figure 4. Controlled Sharing: (i) direct VM-to-VM, (ii) VM-to-VM through local peripheral resources through resource VM, (iii) VM-to-VM through distributed networking resources

sHype enforces mediated sharing between VMs by labeling both VMs and resources that can be shared among virtual machines (e.g., inter process communication, shared memory, virtual block devices, virtual LANs). sHype then controls information flow between virtual machines by a mandatory access control policy (MAC) based on these labels. While direct VM-to-VM communication and cooperation through Inter Process Communication (IPC) and Shared Memory

(SHMEM) are fully controlled by the hypervisor, the extent to which shared peripheral resources are controlled by the hypervisor differs greatly from VMM to VMM implementation. Figure 4 summarizes the three major ways to share virtual resources and the access control enforcement (E) and decision (D) points relative to those resources.

sHype mediates security-sensitive operations that might enable sharing of resources between virtual machines and authorizes those operations—all operations depicted in Figure 4.i-4.iii—according to the security policy. The security enforcement inside the hypervisor itself is protected from attacks such as modification since the hypervisor is protected against virtual machines.

The sHype security enforcement inside of I/O device virtual machines (ii, iii in Figure 4) is protected against other virtual machines by conventional isolation properties of the hypervisor. sHype implements access controls when virtual machines bind to resources (bind-time authorization) and revokes access if the bind-time authorization conditions cease to hold.

In effect, sHype acts as a reference monitor, leveraging existing isolation between virtual resources offered by the VMM. It achieves enterprise-grade assurance guarantees with minimal changes to the underlying VMM infrastructure and minimal performance overhead.

B. TPM Virtualization

The Trusted Platform Module [3] is an emerging security building block, introduced to offer a system-wide hardware root of trust that cannot be compromised by the system software. The sHype architecture includes a design that virtualizes the hardware TPM by creating software TPM instances that are assigned to virtual machines. Based on this TPM device virtualization, TPM-based attestation [16] provides the ability to generate and report run-time measurements of executable code on the hypervisor and virtual machines and to infer properties of the running system.

Figure 5 shows the generic TPM virtualization built on top of (i) a cryptographic coprocessor or (ii) a hardware Trusted Platform Module. Either creates a hardware root-of-trust [19]. The hardware root of trust is extended through the hypervisor to build trust into the individual software TPM instances (virtual TPMs), which in turn serve as roots of trust for the individual guest virtual machines.

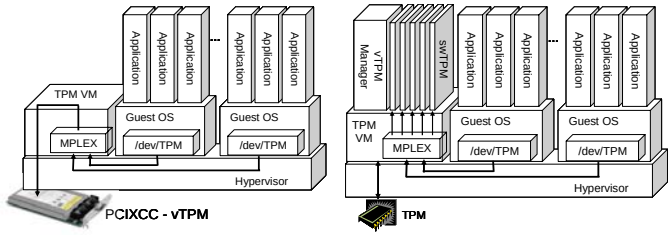

Figure 5. TPM Virtualization Based on PCIX-Crypto Coprocessor (left) and TPM VM (right)

The TPM VM must start first on such a system, even before any privileged I/O-VM. The TPM VM communicates either with a dedicated TPM instance on the PCIX-Crypto Coprocessor (Figure 5 left) or with the hardware TPM (Figure 5 right). Guest virtual TPM instances are created on demand whenever a guest virtual machine with configured TPM support is created. The guest virtual TPM instance is contained within (i) the coprocessor or (ii) the TPM VM.

This architecture enables the use of a Trusted Platform Module (TPM) on systems where multiple Operating Systems are running concurrently and require TPM support. The TPM is designed to support a single operating system at a time. Therefore, virtual TPM management extensions have been proposed [17] that specify the creation, deletion, and secure migration of independent instances of virtual TPMs, based on the current configuration requirements of the platform. In this model, each created instance of a virtual TPM is associated with a single virtual machine and is securely migrated with its associated virtual machine.

The tamper-sensing and responding IBM Cryptographic Coprocessor (PCIXCC [20]) offers an ideal platform for hosting virtual TPM functionality (cf. Figure 5 left) where the highest degree of security is required. The built-in tamper sensitivity of the PCIXCC makes it impossible for intruders with physical access to the device to gain access to sensitive data (e.g., private keys) on the device. It is powerful enough to run multiple virtual TPM instances at the same time. It includes hardware acceleration for cryptographic operations such as RSA key generation, encryption and decryption. In this case, the multi-purpose PCIXCC replaces the hardware TPM.

Figure 5 (right) shows virtual TPMs running in a TPM VM. The TPM VM itself is associated with the system's hardware TPM. In this solution, the software TPM instances rely on the security of the TPM VM.

Another approach leveraging virtualization and trusted computing to create protected systems is Terra [22]. Terra partitions a tamper-resistant hardware platform into multiple, isolated virtual machines, providing the appearance of multiple boxes on a single, general-purpose platform. Compared to Terra, sHype defines TPM virtualization and a complete MAC enforcement mechanism and basic MAC policies for mediated sharing in distributed systems

V. FUTURE DIRECTIONS

The purpose of virtualization and hypervisor security, as discussed in this paper, is the development of secure computing foundations – combining coarse grained isolation and trusted computing technologies in order to provide verifiable containment and trust properties across large distributed environments. In such environments, hardware provides the basis for these properties. The promise of realizing quantifiable security and simplified operational security management for business and IT services will drive progress across the spectrum, from low-level hardware-related developments, to high-level distributed systems management.

At the processor and chip-set level, continued acceleration of security and virtualization features are expected in the near term. Examples include the acceleration of exits and the propagation of page table entry changes when access bits are modified in order to maintain consistency between virtual machine and hypervisor shadow page table entries. Multi-core processors are emerging in the high-volume market and security and trusted computing functions such as cryptographic acceleration engines and trusted platform modules will be integrated into the processor complex. This trend may well continue into the embedded processor and microcontroller space in support of peripheral as well as mobile and pervasive devices that must interact with the rest of the virtualized infrastructure. As hypervisors mature, processor support for recursive virtualization, the ability for a hypervisor to operate within a virtual machine that is itself supported by a hypervisor, may be necessary to preserve the investment made in these mature solutions.

It is likely that an increasing number of peripherals will support self-virtualization, capable of supporting multiple logical adaptor instances within one physical adaptor. Additionally, peripherals will support trusted computing authentication and integrity goals with the incorporation of Trusted Platform Modules and attestation capabilities. The combination of these developments will lead to the emergence of peripherals that are capable of enforcing system-wide access control and information flow security policies in a verifiable manner, thus becoming an extended part of the trusted computing base while relieving the hypervisor of the policy enforcement obligations associated with these resources.

Utility and other distributed computing models, such as cloud computing, will also continue to gain acceptance. This is due in part to the economic advantages of having access to essentially unlimited computing resources, paying for only what is used and being able to stipulate service level agreements or quality of service guarantees with little or no up-front investment. Such infrastructure usage scenarios will require reliable and secure resource monitoring and metering that can be trusted by both the owner of the workloads and the owner of the infrastructure. Hardware support for low-level monitoring and metering, such as virtual processor cycle or storage bandwidth, will be necessary to support business-level requirements with minimum overhead. This same hardware support for resource control will be required in order to enforce resource usage limits that will defend against denial of service attacks in mixed use environments – those with workloads from competing interests that are potentially hostile to each other. Hardware support for reliable sanitization of resources that are frequently reused, such as accelerated zeroization of memory pages and various system buffers, is also required for fast and efficient provisioning of workloads into virtual environments.

Enabling and managing what will essentially become a distributed trusted computing base, built upon the secure hardware and virtualization foundations discussed throughout this paper, is the greatest promise of these technologies. The academic research and industry communities must leverage emerging trusted computing technologies and virtualization capabilities to further bridge the middleware-to-systems gap and to relieve application developers from the burden of implementing and verifying security-related functionality.

ACKNOWLEDGEMENTS

The authors would like to thank their colleagues Stefan Berger, Kenneth Goldman, and Ray Valdez for their valuable discussions and support as well as their contributions in implementing and improving sHype and vTPM for the Xen Opensource hypervisor.

REFERENCES

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

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

[3] Trusted Computing Group: TCG Specification Architecture Overview. Specification, Revision 1.2, April 2004.

[4] R. Sailer, T. Jaeger, E. Valdez, R. Cáceres, R. Perez, S. Berger, J. Griffin, L. van Doorn: Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. 21st Annual Computer Security Applications Conference (ACSAC), December 5-9, Tucson, Arizona, 2005

[5] Intel: Intel® Virtualization Technology Specification for the IA-32 Intel® Architecture. C97063-002, April 2005

[6] AMD: AMD64 Virtualization Codenamed 'Pacifica' Technology, Secure Virtual Machine Architecture Reference Manual. Publication no. 33047, revision 3.01, May 2005.

[7] D. Grawrock: LaGrande Architecture SCMS – 18. Intel Developer Forum, September 2003.

[8] M. LaPedus: AMD tips 'Pacifica' and 'Presidio' processors for '06. http://www.eetimes.com/semi/news/showArticle.jhtml?articleID=52601317, Nov. 2004.

[9] W. Armstrong, R. Amdt, D. Boutcher, R. Kovacs, D. Larson, K. Lucke, N. Nayar, and R. Swanberg: Advanced Virtualization Capabilities of POWER5 Systems. IBM Journal of Research and Development, July/September 2005. 49(4/5): p. 523-532.

[10] 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.

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

[12] R. J. Creasy: The Origin of the VM/370 Time-Sharing System. IBM Journal of Research and Development, 25(5): 483-490, September 1981

[13] IBM Processor Resource/Systems Managsement (PR/SM) Planning Guide, SB10-7036-01, eServer zSeries 990.

[14] 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.

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

[16] 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, August 2004.

[17] S. Berger, R. Cáceres, K. Goldman, R. Perez, R. Sailer and L. van Doorn: vTPM – Virtualizing the Trusted Platform Module. 15th USENIX Security Symposium, July 2006, Vancouver, Canada.

[18] AMD I/O Virtualization Technology (IOMMU) Specification, 2006, http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf.

[19] S. Berger, R. Cáceres, D. Pendarakis, R. Perez, R. Sailer, W. Schildhauer, D. Srinivasan, E. Valdez. TVDc: Managing Security in the

Trusted Virtual Datacenter. ACM SIGOPS Operating Systems Review, Vol 42, Issue 1, January 2008.
[20] J. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. Smith, S. Weingart. Building the IBM 4758 Secure Cryptographic Coprocessor. IEEE Computer, October 2001, pp. 57-66.
[21] J.S. Robin, C.E. Irvine. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. 9th USENIX Security Symposium, August 2000, Denver.
[22] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In Proceedings of the ACM Symposium on Operating System Principles, October 2003.
[23] B. D. Payne, R. Sailer, R. Cáceres, Ron Perez, and W. Lee. A Layered Approach to Simplified Access Control in Virtualized Systems. ACM SIGOPS Operating Systems Review, Vol. 41, No. 3, July 2007.
[24] J. McCune, S. Berger, R. Cáceres, T. Jaeger, R. Sailer: Shamon -- A System for Distributed Mandatory Access Control. 22nd Annual Computer Security Applications Conference (ACSAC), Miami Beach, Florida, December 2006.

Ronald Perez is a Senior Manager and Senior Technical Staff Member at the IBM T. J. Watson Research Center where he currently leads the Systems Solutions and Architecture Department, multiple teams of research scientists and engineers pursuing advances in a diverse set of systems technologies including virtualization and systems management, next generation memory subsystems, stream processing, multimedia and information theory. His research interests also include systems security, with over ten years in this field and having led the Secure Systems Department at Watson prior to his current position. Ronald currently represents IBM as Vice President of the Trusted Computing Group, a not-for-profit organization formed to develop, define, and promote open standards for hardware-enabled trusted computing and security technologies. Ronald received his degree in Computer Science from The University of Texas at Austin.

Leendert van Doorn is a Senior Fellow at AMD where he runs the Software Technology Office. There he and his team actively drive AMD's virtualization, platform manageability and manycore strategies. Before joining AMD he was a senior manager at IBM's T.J. Watson Research Center, where he managed the secure systems and security analysis departments. He received his Ph.D. from the Vrije Universiteit in Amsterdam where he worked on the design and implementation of microkernels. In his former job at IBM he was also actively involved in IBM's virtualization strategy, created and lead IBM's secure hypervisor and trusted virtual data center initiatives, and was on the board of directors for the Trusted Computing Group. Despite all these distractions, he continued to contribute code to the Xen open-source hypervisor, such as the integrated support code for AMD-V and Intel®VT-x. When conference calls and meetings are getting too much for him, he finds refuge at Rice University where he is an adjunct professor.

Reiner Sailer received his PhD degree in Electronic Engineering from the University of Stuttgart, Germany in 1999. He joined the IBM T.J. Watson Research Center in 1999, where currently he is a Research Staff Member and the Manager of the Security Services (GSAL) group. He previously led the Trusted Virtual Datacenter project at the T. J. Watson Research Center. His current research interests include systems security, trusted computing, virtualization infrastructure security, and virtualization-based security services.