Virtualization and Hardware-Based Security

Hypervisors allow virtualization at the hardware level. These technologies have security-related strengths as well as weaknesses. The authors examine emerging hardware and software virtualization technologies in the context of modern computing environments and requirements.

Ronald Perez and Reiner Sailer
IBM T.J. Watson Research Center

Leendert van Doorn
Advanced Micro Devices

Virtualization is the process of presenting something as being genuine when in fact it isn't. Virtualization in the computer architecture domain is the presentation of an environment to one layer in an information technology stack that abstracts or represents a lower layer. System architects typically insert this layer of indirection between existing layers in a hardware or software stack 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 might involve language-level runtimes that provide high-level abstract architectures for applications, or a thin hardware virtualization layer of software situated between system hardware and the operating system layer that provide 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. Although virtualization comes in many forms, including process, storage, and network virtualization, here we focus on security and hardware support for this thin hardware virtualization layer, often termed a *virtual machine monitor* (VMM) or *hypervisor.*

Hypervisor virtualization

Early virtual machines (VMs) were essentially computing environments that simulated or emulated the host system's existing hardware while arbitrating access to shared system resources. This let multiple instances of operating systems run on the same system, where each operating system was originally designed to be the sole arbiter of system resources. Because systems of the time—largely mainframes designed in the 1960s and 1970s, such as the IBM VM/370's predecessors[1]—were expensive, these early VMs gave companies distinct economic advantages by letting them partition one physical system into multiple logical systems. The same advantages exist today for the IBM VM/370's heirs, such as the IBM PR/SM and z/VM VM monitors (VMMs) and the S/390 and zSeries mainframes.[2]

Although hypervisors associated with high-end systems have continued to evolve and remain critical to maximizing these systems' usage, increasing the use of mid-range and high-volume systems has become an additional factor driving hypervisor development and deployment. These systems' increased electrical power requirements rival increases in computing power and overall performance. The expense involved in housing and managing these systems hasn't 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. Hypervisors are therefore an attractive option for large data centers 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 apparent. By checkpointing and repli-

cating VMs in a distributed systems environment, hypervisors can potentially achieve continuous or high-availability requirements with greater ease and at less cost. Emerging tool chains are enabling the rapid provisioning of workloads to VMs. The tool chains enable the capability to load-balance an enterprise's entire infrastructure by migrating existing VMs 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.

Hypervisors are particularly well suited for a few basic but strong security primitives—namely, separation and controlled sharing.

A system can achieve separation in several ways:

- using different hardware facilities for different workloads (physical separation),
- running workloads at different times (temporal separation),
- cryptographically protecting workload-specific data (cryptographic separation), and
- using a reference monitor[3] or security kernel to separate workloads and their resources (logical separation or isolation).

Hypervisors function as reference monitors, providing workload isolation on an operating system instance granularity (as opposed to operating systems, which strive to provide process-level isolation).

In secure systems designs, the reference monitor mediates all security-sensitive operations, such as access to objects or communications between subjects. In hypervisors, objects are system resources and subjects are VMs. Reference monitors must be small enough to be fully tested and analyzed and relatively immune to compromise. In addition, the reference monitor is always invoked, making it impossible to bypass its mediation functionality. A hypervisor is also a key element in the entire system's trusted computing base—the hardware, firmware, and software components in layered-system architectures that must be correct to enforce the explicit or implicit system security policy.

Because the hypervisor reference monitor mediates access to and between coarse-grained entities—such as processors, memory (in addition to the memory management performed at the operating-system level), disks, and VMs—hypervisors are often orders of magnitude smaller in size and less complex than modern operating systems. They're therefore less difficult to test and analyze for correctness and to protect from compromise. Hardware features, such as privileged operating modes and protected memory support, often provide additional protection.

Together with hardware-architected hypervisor calls (similar to operating system calls), these components and features ensure that VMs cannot bypass the hypervisor mediation functionality. Hypervisors have traditionally supported strong isolation or separation of VMs and their workloads, including fault isolation—limiting an application or operating system fault's effects within a VM. Administrators have been able to explicitly configure these systems to support system resource sharing and communication between VMs. However, hypervisors that support sharing based on explicit security policies and labels associated with each VM and its resources, such as the DEC VAX VMM[4] and the KVM/370[5] security-enhanced version of the IBM VM/370, weren't available on a large scale. This is perhaps partly because of the commercial requirements of the day and the systems' high-assurance government-/military-oriented design targets. Because hypervisors must maintain isolation while maximizing resource utilization and support for the more distributed and interconnected nature of contemporary workloads, policy-driven controlled-sharing requirements for commercial hypervisors on mid-range and high-volume systems are increasingly becoming an issue.

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

Hardware virtualization support

Virtualization's major disadvantage is its large performance overhead. This is especially true when using interpretative or emulation techniques. A single emulated machine instruction can easily expand to thousands of real instructions and can cause significant performance degradation. To counter this, CPU manufacturers have been developing hardware support for virtualization in which part or all of the emulation occurs in the CPU itself. Major vendors are also working on virtualization capabilities for the CPU, I/O, and some specialized devices.

Traditional computer systems consist of memory

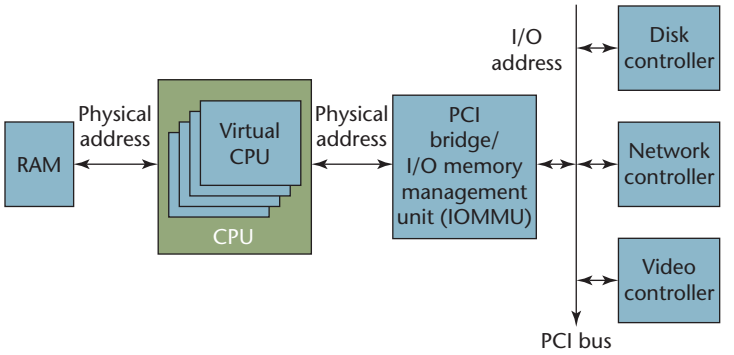

Figure 1. Virtualized system architecture overview. Virtualizing a typical computer system involves the system's shared resources, in particular the CPU and I/O.

(RAM), a CPU, an I/O controller (typically a PCI bridge), and one or more I/O devices, such as a disk and network and video controllers. Virtualizing the shared resources in this architecture, as Figure 1 shows, involves changing the CPU and the bus controller that arbitrates accesses to the I/O bus. The I/O devices themselves might also be virtualization aware, but that's beyond this article's scope.

A typical CPU consists of many shared resources—such as interrupt vectors, page tables, interrupt controllers, timers, and special descriptor tables. The hypervisor must virtualize these resources, but its job can be considerably easier when the right processor abstractions are available.[6] It must give the illusion of a VM to the guest operating system that's running inside it. This involves managing physical memory, interrupts, faults, and I/O devices.

Intel Vanderpool and AMD Pacifica Technology

Until recently, hardware virtualization support was available on only mainframe computers. Some of these capabilities are now integrated into modern commodity processors; for example, both Intel and Advanced Micro Devices have designed and implemented their own virtualization extensions—Intel's Vanderpool Technology[7] and AMD's Secure VM.[8] Both VT and SVM provide roughly the same functionality. Both create a container that's a virtual CPU. Inside the container, you can run an unmodified operating system that's unaware that it's operating in a controlled environment.

The only way out of this container is under well-defined conditions, or *exits*. Exits are generated whenever the code in the container executes privileged instructions, such as changing the CPU state or the page tables, or causing a page fault. (Operating systems use privileged instructions to control critical system resources; only the operating system kernel can execute them.) These exits cause a trap into the underlying hypervisor that executes in the root container. The hypervisor must then emulate the correct behavior such that the operating system running inside the container is unaware that it's being virtualized. For example, to access a specific physical memory location, the operating system creates a mapping from a virtual to a physical address in the page table structure. The operating system must then 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. The hypervisor must then 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.

IBM's approach in its Power Architecture server processors (such as Power5)[9] differs from that taken by Intel and AMD. Rather than introduce a heavyweight container and exit concept, the Power processor duplicates certain key control registers in a new hypervisor state (akin to user and supervisor state) that operate independently from their supervisor-state counterparts. In a way, it's the complex instruction set computer (CISC) versus reduced instruction set computing (RISC) approach. Power introduced a few lightweight concepts to support virtualization.

I/O memory management unit

I/O virtualization aims to give a VM direct device access such that it can't overwrite other VMs. This isn't a problem for most devices, but some have bus-master capabilities that must 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 isn't assigned to the VM controlling the device. This is a common problem. CPUs therefore have an indirection layer and provide a virtual-to-physical memory map abstraction. The CPU's memory management unit (MMU) provides this mapping. I/O uses something analogous—the IOMMU[9,10]—which is typically part of the bus controller (see Figure 1). The details will depend on the specific design, but each I/O device typically has its own address translation map.

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

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 company quickly renamed the initiative the Next-Generation Secure Computing Base. NGSCB aimed to provide the following guarantees:

- process isolation,
- sealed storage,
- platform attestation, and
- secure I/O paths.

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

Intel's secure computing platform—Lagrande Technology[11]—consists of a VT core to provide process isolation, special keyboard and video capabilities for the secure I/O paths, a direct memory access (DMA) exclusion vector to isolate I/O devices from the security kernel, and tight integration with the trusted platform module (TPM) version 1.2 specification to provide sealed storage and platform attestation. Although AMD's Presidio[12] secure computing platform provides the same high-level functionality, its technology roadmap differs significantly from Intel's. Unlike Intel, AMD integrates the TPM capabilities and DMA exclusion capabilities directly into SVM. Consequently, Presidio consists of an SVM and a set of secure I/O capabilities.

Both Intel and AMD added two new features to their CPUs and chipsets. The first, 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 Trusted Computing Group's *dynamic root of trust*.[13] 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 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's Senter instruction[11] and AMD's Skinit instruction.[8] Both are conceptually similar. Here we describe the Skinit behavior.

Upon executing a Skinit instruction, the processor is reinitialized into a well-known state in which it can execute a secure loader such that the loader can't be tampered with during its execution. In this state, the processor

- disables the interrupts
- inhibits DMA to the memory area containing the secure loader, and
- initializes the special-purpose registers controlling memory accesses to safe values.

This environment guarantees that other programs running on the CPU or external devices can't modify the loader while it's running.

Once the processor has been reinitialized, it sends a secure hash of the 64-Kbyte loader to the TPM, which stores it in the platform configuration register (PCR) of the TCM.[7] Only the CPU, using special locality bus cycles that can't be generated from software, can write this PCR. This ensures that only the CPU Skinit instruction can generate this hash value. The secure loader's hash value constitutes the dynamic root of trust. As soon as the loader's hash is stored inside the TPM, control transfers 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, or resume execution where the operating system that invoked Skinit left off. Applications running on the operating system can then use these measurements to unseal storage or attest to remote parties of the software stack that's running and the hardware platform that it's running on.

The secure I/O capabilities are the more challenging aspects of LT and Presidio. The first—secure input—is straightforward. The perceived threat is one in which an adversary intercepts or modifies the communication from the keyboard and mouse into the secure environment. To prevent this, trusted components in the keyboard encrypt communication from the keyboard to the data recipient—a keyboard device driver in the secure kernel. Secure output, such as video, suffers from the same communication threats, and also suffers from Trojan horse attacks. How can a user distinguish between output from a secure environment and output from an insecure environment? Despite good research in this area, no practical solutions yet exist. Intel has proposed a solution in which the secure environment always displays on top of the current screen and can be activated through a secure attention key mechanism. Others have suggested rendering a pass phrase that's only available to secure kernels and users into the window's background to convey to the user that he or she is interacting with a secure environment. This too might prove too cumbersome for the end user. Whereas everyone agrees

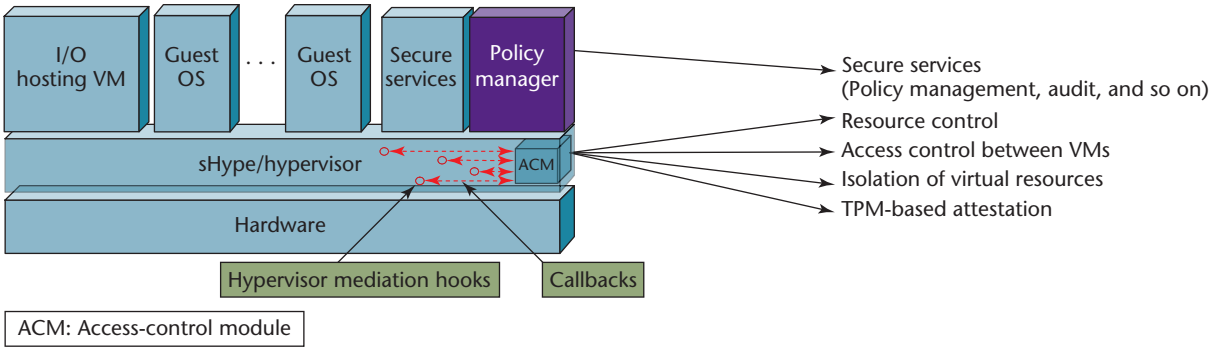

Figure 2. sHype hypervisor security architecture. The architecture's major components are the policy manager, which oversees the access-control policy; the access-control module (ACM); and the mediation hooks.

that secure user I/O is critical to securing client systems, little consensus exists in the industry as to what form that should take.

sHype security architecture

Figure 2 illustrates the sHype[14] security architecture and its integration into a VMM environment. We implemented sHype in various stages for multiple hypervisors, including the Xen[15] open source hypervisor.

Building on the previously described hardware support, our major design goal for sHype is to establish a secure foundation for server platforms, providing functions such as:

- *strong isolation and mediated sharing* between VMs, strictly controlled by a flexible access-control enforcement engine;
- *attestation and integrity guarantees* for the hypervisor and its VMs, supported by a virtual TPM architecture (TPM-based attestation[16] lets VMs generate and report the running system's properties);
- *resource control and accurate accounting guarantees*, letting the hypervisor enforce quality-of-service agreements between service provider and consumer; and
- *secure services*, providing the base infrastructure in sHype for refining complex monolithic runtimes 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 form the basis of IBM Research's Trusted Virtual Data Center,[17] which simplifies consistent and strong isolation guarantees in distributed virtualized data centers.

Access-control architecture

The sHype access-control framework enforces a formal security policy (mandatory access control, or MAC) on information flow between VMs, independently of generic user VMs. It moves the security state from being defined by various ad hoc system administrator decisions to a state that's formally defined by the security policy and enforced by sHype, independently of guest VMs.

Few VM monitors offer sufficient information inside the hypervisor to distinguish the information flow's direction or the semantics of higher-level operations inducing such an information flow. sHype therefore aims to provide coarse-grained, robust, and simple access control on VMs and resources within the hypervisor and defers finer-grained access control to higher layers (guest operating systems, middleware, and applications). It promotes a layered security approach over a monolithic one. Higher-layer policies exploit lower-layer security policies and focus on refinement rather than reimplementation. This strategy is comparable to communication stacks, where higher layers usually rely on lower-layer functions to bridge physical differences (medium access or layer 2) or to limit network exposure (IP firewalling or layer 3). Yet they sometimes decide to reimplement some of the lower-layer functions (for example, error checks) in higher layers based on additional information that isn't available to the lower-layer functions. The access-control architecture's major components are the policy manager, which creates and updates the access-control policy; the access-control module (ACM); and the mediation hooks (see Figure 2).

The policy manager maintains the hypervisor security policy, which defines the rules the ACM uses to decide which VMs can access which resources. The policy manager is implemented inside a special-purpose VM to keep related complexity out of the hypervisor. It provides the ACM inside the hypervisor with a precompiled security policy. Security mediation hooks mediate access to resources inside the hypervisor that enable information flow between VMs. Security hooks are located both within the hypervi-

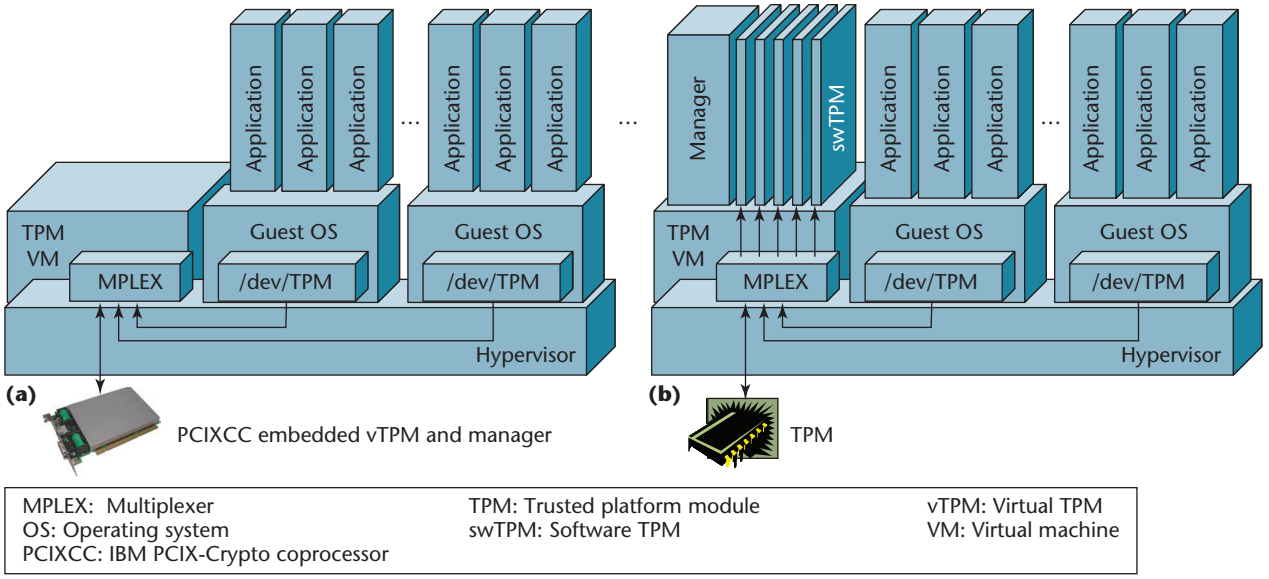

Figure 3. Trusted platform module virtualization built on top of (a) an IBM PCIX-Crypto coprocessor and (b) a hardware TPM. The hypervisor extends the hardware root of trust created by these systems to build trust into the virtual TPMs.

sor to mediate direct VM-to-VM sharing, and within the I/O hosting VM and VMM to mediate access to virtualized resources. A security hook implements a redirection of access of VMs 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 ACM considers the labels on VMs and resources to control information flow between VMs as well as the collocation of VMs.

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

TPM virtualization

The TPM[13] is an emerging security building block offering system-wide hardware roots of trust that the system software can't compromise. The sHype architecture virtualizes the hardware TPM by creating software TPM instances that are assigned to VMs.

Figure 3 shows the generic architecture built on top of a cryptographic coprocessor (Figure 3a) and a hardware TPM (Figure 3b). Both create a hardware root of trust. The hypervisor extends the hardware root of trust to build trust into the individual software TPM instances (virtual TPMs), which in turn serve as roots of trust for the individual guest VMs.

The TPM VM must start on such a system first, even before any privileged I/O VM. The TPM VM communicates with a dedicated TPM instance on the IBM PCIX-Crypto coprocessor (PCIXCC)[18] or the hardware TPM. The vTPM manager[19] in the TPM VM creates guest virtual TPM instances on demand whenever the hypervisor creates a guest VM with configured TPM support. The guest virtual TPM instance is contained within the coprocessor or the TPM VM.

This architecture lets you use a TPM on systems running multiple operating systems concurrently and requiring TPM support. The TPM supports one operating system at a time. Virtual TPM management extensions specify the creation, deletion, and secure relocation of independent instances of virtual TPMs, based on the platform's current configuration requirements. In this model, the vTPM manager associates each created instance of a virtual TPM with a single VM and securely relocates them.[19]

The tamper-sensing and responding PCIXCC offers an ideal platform for hosting virtual TPM functionality when the highest degree of security is required. PCIXCC's built-in tamper sensitivity prevents intruders with physical access to the device from gaining access to sensitive data (for example, private keys) on the device. It's 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 multipurpose PCIXCC replaces the hardware TPM.

Figure 3b shows virtual TPMs running in a TPM VM. The TPM VM is associated with the system's hardware TPM. In this solution, the software TPM

instances rely on the TPM VM's security. Terra, another approach leveraging virtualization and trusted computing to create protected systems, partitions a tamper-resistant hardware platform into multiple, isolated VMs, presenting multiple boxes on a single, general-purpose platform.[20] Unlike Terra, sHype defines TPM virtualization and a complete MAC enforcement mechanism and basic MAC policies for mediated sharing in distributed systems.

Future directions

Virtualization and hypervisor security aim to develop secure computing foundations, combining coarse-grained isolation and trusted computing technologies 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, acceleration of security and virtualization features is expected in the near term. Examples include the acceleration of exits and the propagation of page table entry changes when a guest operating system running in a VM modifies access bits to maintain consistency between VM and hypervisor shadow page table entries. Heterogenous multicore processors are emerging in the high-volume market and chip designers will integrate security and trusted-computing functions, such as cryptographic acceleration engines and TPMs, into the processor complex. This trend may well continue into the embedded processor and microcontroller space. As hypervisors mature, processor support for recursive virtualization (that is, a hypervisor's ability to operate within a VM that is itself supported by a hypervisor) might be necessary to preserve the investment made in these mature solutions.

An increasing number of peripherals will likely support self-virtualization (that is, be able to support multiple logical adaptor instances within one physical adaptor). Peripherals will also support trusted-computing authentication and integrity goals with the incorporation of TPMs and attestation capabilities. The combination of these developments will lead to the emergence of peripherals that can enforce 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 increase is partly due to the economic advantages of having access to essentially unlimited computing resources, paying only for what you use, and being able to stipulate service-level agreements or quality-of-service guarantees with little or no upfront investment. Such infrastructure usage scenarios will require reliable and secure resource monitoring and metering that both workload and infrastructure owners can trust. We'll need hardware support for low-level monitoring and metering, such as virtual processor cycle or storage bandwidth, to support business-level requirements with minimum overhead. We'll need this same hardware support for resource control 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 frequently reused resources, 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 we've discussed, 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 relieve application developers from the burden of implementing and verifying security-related functionality. □

Acknowledgments

We thank our 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 open source hypervisor.

References

1. R.J. Creasy, "The Origin of the VM/370 Time-Sharing System," *IBM J. Research and Development*, vol. 25, no. 5, Sept. 1981, pp. 483–490.
2. *IBM Processor Resource/Systems Management (PR/SM) Planning Guide*, SB10-7036-01, eServer zSeries 990.
3. J.P. Anderson et al., *Computer Security Technology Planning Study*, tech. report ESD-TR-73-51, vols. I and II, Air Force Systems Command, USAF, 1972.
4. P.A. Karger et al., "A Retrospective on the VAX VMM Security Kernel," *IEEE Trans. Software Eng.*, vol. 17, no. 11, Nov. 1991, pp. 1147–1165.
5. B.D. Gold, R.R. Linde, and P.F. Cudney, "KVM/370 in Retrospect," *Proc. IEEE Symp. Security and Privacy*, IEEE CS Press, 1984, pp. 13–23.

6. J.S. Robin and C.E. Irvine, "Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor," *Proc. 9th Usenix Security Symp.*, Usenix Assoc., 2000, p. 10.
7. Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide*, www.intel.com/products/processor/manuals/.
8. AMD, *AMD64 Architecture Programmer's Manual, Volume 2: System Programming*, http://developer.amd.com/documentation/guides/Pages/default.aspx.
9. W.J. Armstrong et al., "Advanced Virtualization Capabilities of Power5 Systems," *IBM J. Research and Development*, vol. 49, nos. 4/5, 2005, pp. 523–532.
10. Advanced Micro Devices, *AMD I/O Virtualization Technology (IOMMU) Specification*, 2006; www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/34434.pdf
11. D. Grawrock, *The Intel Safer Computing Initiative: Building Blocks for Trusted Computing*, Intel Press, 2006.
12. M. LaPedus, "AMD Tips 'Pacifica' and 'Presidio' Processors for '06," Nov. 2004; www.eetimes.com/semi/news/showArticle.jhtml?articleID=52601317.
13. Trusted Computing Group, *TCG Specification Architecture Overview*, revision 1.2, Apr. 2004; www.trustedcomputinggroup.org/downloads/TCG_1_0_Architecture_Overview.pdf.
14. R. Sailer et al., "Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor," *Proc. 21st Ann. Computer Security Applications Conf.* (ACSAC), IEEE CS Press, 2005, pp. 276–285.
15. P. Barham et al., "Xen and the Art of Virtualization," *Proc. 19th ACM Symp. Operating Systems Principles*, ACM Press, 2003, pp. 164–177.
16. R. Sailer et al., "Design and Implementation of a TCG-based Integrity Measurement Architecture," *Proc. 13th Usenix Security Symp.*, Usenix Assoc., 2004, pp. 223–238.
17. S. Berger et al., "TVDc: Managing Security in the Trusted Virtual Datacenter," *ACM SIGOPS Operating Systems Rev.*, vol. 42, no. 1, 2008, pp. 40–47.
18. J. Dyer et al., "Building the IBM 4758 Secure Cryptographic Coprocessor," *Computer*, Oct. 2001, pp. 57–66.
19. S. Berger et al., "vTPM—Virtualizing the Trusted Platform Module," *Proc. 15th Usenix Security Symp.*, Usenix Assoc., 2006, pp. 305–320.
20. T. Garfinkel et al., "Terra: A Virtual Machine-based Platform for Trusted Computing," *Proc. ACM Symp. Operating System Principles*, ACM Press, 2003, pp. 193–206.

Ronald Perez *is a senior manager and senior technical staff member at the IBM T.J. Watson Research Center, where he 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. Perez has a BA in computer science from the University of Texas at Austin. Contact him at ronpz@us.ibm.com.*

Leendert van Doorn *is a senior fellow at Advanced Micro Devices, where he runs the Software Technology Office, System Manageability Organization, and he is an adjunct professor at Rice University. His research interests include virtualization, operating systems, security, and system manageability. van Doorn has a PhD in computer science from the Vrije Universiteit in Amsterdam. He is a senior IEEE member. Contact him at leendert@ieee.org.*

Reiner Sailer *is a research staff member and manager of the Security Services (GSAL) group at the IBM T.J. Watson Research Center. His research interests include systems security, trusted computing, virtualization infrastructure security, and virtualization-based security services. Sailer has a PhD in electronic engineering from the University of Stuttgart, Germany. Contact him at sailer@us.ibm.com.*