A Case for Hardware Protection of Guest VMs from Compromised Hypervisors in Cloud Computing

Jakub Szefer and Ruby B. Lee
Dept. of Electrical Engineering
Princeton University
{szefer, rblee}@princeton.edu

***Abstract*—Cloud computing, enabled by virtualization technologies, is becoming a mainstream computing model. Many companies are starting to utilize the infrastructure-as-a-service (IaaS) cloud computing model, leasing guest virtual machines (VMs) from the infrastructure providers for economic reasons: to reduce their operating costs and to increase the flexibility of their own infrastructures. Yet, many companies may be hesitant to move to cloud computing due to security concerns. An integral part of any virtualization technology is the all-powerful hypervisor. A hypervisor is a system management software layer which can access all resources of the platform. Much research has been done on using hypervisors to monitor guest VMs for malicious code and on hardening hypervisors to make them more secure. There is, however, another threat which has not been addressed by researchers – that of compromised or malicious hypervisors that can extract sensitive or confidential data from guest VMs. Consequently, we propose that a new research direction needs to be undertaken to tackle this threat. We further propose that new hardware mechanisms in the multicore microprocessors are a viable way of providing protections for the guest VMs from the hypervisor, while still allowing the hypervisor to flexibly manage the resources of the physical platform.**

I. Introduction

Today's virtualization extensions to microprocessors along with hypervisor software allow cloud computing providers to run multiple VMs on a single physical server. An IaaS provider, such as the Amazon EC2 service [1], can run its many servers at a lower cost than smaller companies can run their own servers, so these companies have incentive to lease VMs: to reduce their operating costs and increase the flexibility of their own infrastructures. Mutually distrusting companies may lease VMs from the same provider, attack the virtualization layer to compromise it and gain information about other VMs (their competitors) through abusing the hypervisor's privileges once it has been compromised. A sample attack vector could be a cloud customer leasing a VM, installing a malicious guest OS which attacks and compromises the hypervisor in order to gain access to the memory contents (data and code) of other VMs. Figure 1 depicts such a scenario. Different companies leasing VMs from the same IaaS provider may try to gain access to other VMs: to steal proprietary code, to access personal patient data, etc. Recent work [14] has shown that it is possible to determine whether VMs are co-located on the same physical server. This makes it easier for malicious customers to find the target server, increasing the chances for a successful attack.

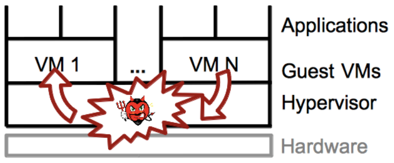

Fig. 1. A compromised hypervisor can attack guest VMs.

To counter this threat, we propose *hypervisor-secure virtualization* as a new research direction for security in cloud computing: to protect guest VMs from attacks by a compromised hypervisor. Specifically, we would like the hypervisor to initiate VMs and manage resources as is done today, but not be allowed to snoop on any confidential contents of the VM during its lifetime. Memory is the key asset which should be protected as the potentially sensitive, private or proprietary code or data is stored in the memory which has been assigned to a VM.

We argue that hardware enhancements can provide a promising solution. Because the hardware is logically below the hypervisor software, it can store data and have functionality which cannot be altered by the hypervisor. Furthermore, changing functionality implemented by hardware, and probing the microprocessor chip to recover secrets, are extremely difficult. Consequently, hardware enhancements are a good approach for tackling the threat. Already, virtualization extensions to the microprocessor hardware are widely deployed and major microprocessor vendors provide them in their commodity products [4], [2]. Extensions in other parts of the servers are also already available, for example the I/O MMU [3] or the SR-IOV (Single Root I/O Virtualization) extension for the PCI devices [5]. These extensions have been added to improve virtualization performance. This indicates that the overheads of modifying the hardware do not outweigh the benefits these extensions can provide for virtualization. The same should be done for security – new features to help protect the virtualized environments need to be added. While many researchers may feel that hardware modifications would incur too much overhead, we see that commercial microprocessors do add new extensions. Adding new features is not only realistic, but even desired by hardware vendors who want to create faster and more secure virtualization environments and attract more customers. Also, because the hardware trend is towards multicore microprocessor chips, we propose that research should focus on adding security features to multicore chips to protect VMs from a compromised hypervisor.

We briefly discuss related work in section II. In section III we discuss *hypervisor-secure virtualization*. In section IV we describe three concrete research directions for protecting the guest VMs. In section V we discuss challenges presented by our new computing model and the opportunities that this model presents. We conclude in section VI.

II. Related Work

With the growing deployments of virtualization, much security related work has been done in two broad areas. First, work has been done on using the virtualization extensions and hypervisors to inspect and monitor the guest VMs. Researchers have looked, for example, at leveraging the hypervisor's most privileged status to monitor VMs for rootkits [19]. Such work typically makes use of the hypervisor's ability to inspect a guest's memory or its emulation of devices. However, this adds more code to the already large TCB (trusted computing base). Second, work has also been done on hardening hypervisors, e.g. [18], and "minimizing" or even removing the hypervisor all together, e.g. NoHype [9]. Such work aims to limit the possible attack vectors.

Outside of virtualization-related research, projects such as [17], [12], [11], [8], [7] have looked at secure processor architectures and protecting trusted software modules through automatic encryption and hashing. Also, the Cell Broadband Engine processor vault [15] provides hardware isolated execution environments where code executing on a synergic processing element (SPEs) inside the Cell processor can be protected from code executing in the rest of the system.

III. Hypervisor-secure Virtualization

In this paper, we hope to bring attention to an un-addressed area of virtualization-related research: protecting the guest VMs from a compromised or malicious hypervisor. To our knowledge all the previous research has concentrated on the guest VM $\rightarrow$ hypervisor attacks and their prevention (either by inspecting the guest VMs or hardening the hypervisors). The hypervisor $\rightarrow$ guest VM attacks have not been addressed. Secure processor architectures which protect and isolate software modules come closest to tackling this problem but have not been designed with virtualization in mind.

In our target scenario, the virtualization layer is provided by the IaaS provider. The provider has no incentive to be malicious nor to supply a malicious hypervisor. We assume, however, that the hypervisor can nevertheless be buggy or misconfigured which can lead to a compromise. Securing hypervisors is a very difficult problem. It is highlighted by the various vulnerabilities in commercial hypervisors (e.g. [6] from the Common Vulnerabilities and Exposures (CVE) database) and also numerous attacks on hypervisors which have been presented in the literature (e.g. [13], [21]). Consequently we propose that the new direction of protecting the VMs from the hypervisor, rather than the other way around, should be studied. We define the hypervisor-secure property as a property of the VM indicating that the VM is protected from a malicious attack by the hypervisor (or other software which gains hypervisor-level privileges) and that the confidentiality or integrity of the VM's data or code is preserved in spite of the attacker having hypervisor privileges.

We propose that the virtualization functionality and flexibility of cloud computing can be maintained when portions or even the whole hypervisor kernel is removed from the TCB. New microprocessor hardware modifications can provide facilities for protecting the guest VMs, while the hypervisor is still able to manage the resources. We call this *hypervisor-secure virtualization*, akin to the root-secure idea for operating systems [16].

A. Threat Model

When undertaking research on hypervisor-secure virtualization, we propose the following assumptions (justified below):

- we assume the hardware is secure (e.g. there are no hardware trojans which can leak data);
- securing the guest OS itself or the applications is not in the scope;
- we do not consider denial-of-service (DoS) attacks.

By introducing new hardware, we have to assume that the hardware works. If there has been some malicious modification to the hardware, then the trust in the system has been lost. We propose that trust evidence needs to be generated by the hardware so that customers, i.e., owners of guest VMs, know that they are communicating with a genuine system. The reliability and track record of the manufacturer will have to be assessed by each customer to decide whether to trust the manufacturer's microprocessors and systems.

Since the goal is for computing in a VM in the cloud to be as secure as if the OS and applications were running on a dedicated server in the customer's own office, bugs, covert channels or other problems with the customer's OS or applications are out of scope since they are already present when the customer is running the OS or applications in his or her own office. Furthermore, previous work has addressed securing OSes and applications. Hence, the additional security vulnerability posed by cloud computing is protecting the guest VMs from the hypervisor.

Denial of Service (DoS) attacks are not in our threat model because they can easily be achieved by just turning off power to the physical server, or by the hypervisor refusing to schedule a guest VM. This can, however, easily be mitigated by the customer having multiple VMs (possibly spread across many providers) such that if one system goes down, the other VMs are not affected. What we want to protect against, however, is that even in the case of a DoS attack, the guest VM's data and code are protected.

B. Model of Operation

Figure 2 depicts the parties in our target model: the cloud customers, the IaaS infrastructure provider and the end users. In this model, the guest VMs run on the cloud provider's servers. We also depict the attack surface between the guest

VMs and the hypervisor. Our proposal is to tackle the unaddressed attacks from the hypervisor to guest VMs.

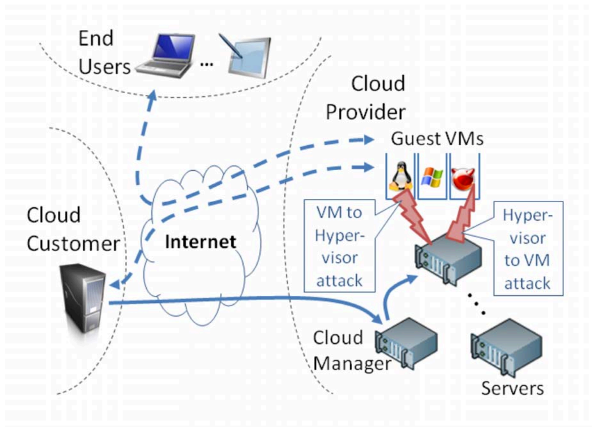

Fig. 2. A cloud customer leases VMs from a third party infrastructure provider; the leased VMs in turn are used either by the cloud customer (e.g. for batch accounting jobs) or end users (e.g. web site visitors).

In our suggested model of operation, the cloud customer is given the means to provide some specification of the confidentiality and integrity protection he wants for the data and code which will run inside the VM. The customer would supply his requested protections together with the OS and applications that he currently provides to an IaaS cloud provider when requesting a new VM. The cloud provider would then use all this information to create a new VM, assign it resources, signal the hardware to initialize the required protections for this VM, and launch the VM. Subsequently, the required protections would be enforced by the hardware and could not be maliciously changed by the hypervisor. Naturally, a mechanism is needed for the hypervisor to update the resource assignment, but the original protections would be enforced throughout the lifetime of the VM so that any update does not violate the customer's requested protections.

The requested protections should be provided in plaintext so that the hypervisor can inspect them and make sure the request does not violate the IaaS provider's rules. Since the hypervisor could modify the protections before it passes the request onto the hardware, the hardware should also provide some trust evidence, e.g., measurements, which can be sent back to the customer to show that it has initialized protections as requested. A private key inside the processor could be used along with on-chip cryptographic routines to sign measurements of the protections which the hardware has initialized. The customer could ask the hypervisor to retrieve the measurements throughout the lifetime of the VM and the signature could be verified using the hardware manufacturer's certificate.

Now that the VM is running and its protections have been verified, the protected memory resources are off-limits to the hypervisor, other VMs or hardware devices (except ones explicitly assigned to the VM). When the VM is terminated, the hypervisor would need to signal the hardware so the hardware could zero out any memory which is being protected to prevent data leakage and only then remove the access restrictions, so the hypervisor can again access the memory (e.g. to assign it to another VM).

IV. Protecting the Guest VMs

There are three paths for leveraging the hardware to protect guest VMs, which we discuss in the following paragraphs. These approaches and the tradeoffs among them should be explored to find the best solution for realizing hypervisor-secure virtualization.

A. Protection Through Isolation

Isolation and resource partitioning is not a new concept, but in today's commodity systems, the hypervisor software is relied on to provide and enforce the isolation. A powerful attack by the hypervisor is to give itself (or some VM) the ability to snoop on some target VM's memory. The first approach is then for the *hardware* to enforce the isolation more strictly and flexibly than it does today. In addition, the hardware should provide the customers with evidence that the correct protections are being enforced.

Because the hardware is logically below the hypervisor software, it can store data and contain functionality which cannot be altered by the hypervisor. Consequently, information about memory regions and their protections could be stored in a dedicated portion of the system not accessible to the hypervisor (e.g. specially reserved parts of DRAM). New mechanisms could consult the stored protections to enforce isolation during memory accesses. Extra checks on each memory access would naturally lead to unacceptable overheads, and a number of methods could be used to minimize the performance impact. For example, the checks could be done when memory address translation is performed (e.g. when updating the translation lookaside buffer, TLB). Once the address translation is performed and validated, future accesses to that address would not incur overhead.

B. Protection Through Cryptography

The hypervisor can also be isolated from accessing the guest VM's data or code if such data and code is encrypted and the hypervisor does not posses the proper keys for decryption. Many previous projects have looked at secure processor architecture and protecting trusted software modules through encryption and hashing, e.g. [17], [12], [11], [8], [7]. Different from our proposal, they only protected small, specially crafted software modules and not whole VMs. Also, some of these proposals include the OS or the hypervisor in the TCB to manage the modules and the software modules have to be encrypted and hashed with specific keys of each processor (incurring excessive re-encryption overhead if the same code is to execute on different processors).

Again, the fact that hardware is logically below the hypervisor can be leveraged so that memory encryption keys are securely stored in new hardware registers, out of reach of the hypervisor. A different key can be generated for each VM and used to protect that VM's memory by encrypting the memory when the VM is created. After VM creation time, the VM's memory would be encrypted whenever it leaves the microprocessor chip. Hardware mechanisms can swap the keys depending on which VM is executing, or when the hypervisor is executing, so that memory is able to be unencrypted only if the right VM is executing.

C. *Protection Through Isolation and Cryptography*

For defense in depth, a third approach can be to combine both isolation and cryptography. The principles from the isolation-based approach can be taken so that the hypervisor can be prevented from accessing memory resources as specified by the customer. The encryption can further defend against hardware attacks and protect data at rest. Because of the isolation, the key management complexity would be greatly reduced and the new hardware may only need one key for protecting whole physical memory.

V. Challenges and Opportunities

The key to remember is that hypervisor-secure virtualization aims to create an environment where a compromised or malicious hypervisor is not able to extract private or confidential information from the guest OS and the applications. On the surface this seems to restrict some of the hypervisor's abilities. In this section we examine both the challenges (and some initial resolutions which remove most of these restrictions) and the new opportunities that hypervisor-secure virtualization presents. Research should focus on providing better resolutions for the challenges and on building on the opportunities.

A. *Challenges*

Allowing the guest VM to specify its protections can create a number of challenges. We list some challenges and for each challenge we propose an initial resolution.

Guest VM Introspection: A challenge will be to allow all existing VMI techniques to continue to work. If this is essential, the customer can always specify that none of the VM's memory is made inaccessible to the hypervisor, and use a trusted hypervisor. However, since hypervisor-secure virtualization allows the guest VMs to set some portion of their memory as inaccessible to the hypervisor, this prevents the hypervisor from being able to monitor all of the guest VM's memory using VM introspection (VMI) techniques [19]. The hypervisor, however, can still use intrusion detection techniques to scan network packets for potential compromise. Also, the guest VM protections can allow the hypervisor to scan the kernel's memory, while not giving it access to the proprietary data or code portion of memory. Alternatively, a trusted software shim, or nested hypervisor, could be used to perform the introspection.

Malicious Guest VMs: A malicious guest VM could attempt to overuse its protections or hog resources. Before creating the VM, however, the hypervisor can freely scan the plain-text request from the customer to make sure that the requested protections are in line with the infrastructure provider's expectation. Also, at any time the guest VM can be terminated and resources reclaimed if a VM is suspected to be misbehaving. In such a situation, the hardware would have to ensure that the protected state is erased so the hypervisor can not, for example, force a crash dump to recover secrets.

Performance Overheads: The isolation approach will result in the hardware intervening on memory accesses from the hypervisor to check the access rights. This performance overhead can be overcome by caching the access rights. A sample implementation could be to modify the TLB update mechanisms such that a memory translation is only placed in the TLB if access is allowed; future accesses will hit in the TLB thus requiring no extra overhead. This requires flushing the TLB whenever any of the protections are updated. On systems where the TLB functionality is disabled, however, each memory access from the hypervisor will have to incur the overhead.

For the encryption approach, there is no overhead for checking access rights during address translation, but there will be overhead for encryption during a write, decryption during a read, and overhead for the verification of hashes. This overhead can be minimized by using, for example, counter-mode encryption. In counter-mode, counter values are encrypted and then logically XORed with the data (both to encrypt and decrypt). The counter values could be encrypted ahead of time so the read and write latency will only increase by the latency of an XOR operation.

Side-channel Attacks: Cache-based side channels, for example, could be used to try to recover cryptographic keys from a VM. Such attacks can be defeated by incorporating new cache designs, e.g. [20], in the microprocessor core.

Hardware Attacks: Hardware attacks were not excluded by our threat model as they are very interesting and challenging. If a customer uses hypervisor-secure virtualization because he or she does not trust the hypervisor provided by the infrastructure provider, he or she may also be in doubt about the facilities where the hardware resides. Could someone sneak into a server room and install a hardware probe on a memory bus? While we believe this is unlikely as it is in the IaaS provider's interest to secure their facilities, it is possible in an insider attack. If cryptography-based hypervisor-secure virtualization is used, then hardware probing of the memory buses or DRAM chips is automatically prevented since all data outside of the microprocessor core is encrypted.

Misconfiguration: Both configuration of new hardware features and protection policies specified by the customers could be sources of misconfiguration that lead to violation of a guest VM's confidentiality or integrity. Designing a trusted hardware or software mechanism for configuring the new hardware is an open challenge. Policy-related issues may be easier to solve: a default policy configuration could be provided by the OS vendors, just as the OS vendors provide default security settings today. The trust evidence provided by fast on-chip cryptographic hardware could further be used to verify that the enforced policy matches the customer's original request.

Device Virtualization: Protecting the guest VM may pose a challenge for device virtualization as the hypervisor may need access to processor registers and the guest's memory to virtualize devices. This issue is simplified for PCI devices, e.g. NIC, with SR-IOV functionality. The SR-IOV configuration registers are memory-mapped so they can easily be configured by the hypervisor; this memory-mapped region can then be protected to prevent information leaking to the hypervisor.

Accounting and QoS: Accounting and QoS do not require the hypervisor to inspect the guest's memory contents. For example, in our model the hypervisor is still able to see the amount of memory the VM uses (just not its plaintext contents), and its usage of network connections.

Live VM Migration: If the hypervisor is not able to inspect the memory contents of the VM, then it will not be able to perform the live migration tasks which require the hypervisor to forward the memory contents from a source to a target server. We propose this can easily be overcome using guest-assisted migration [10]. This gives the customer more control over how the guest VM is migrated and can even be used to avoid migration altogether. For example, "migrating" a web server could simply be done by draining connections from an old VM instance and sending new requests to a new VM.

B. Opportunities

Being able to protect the guest VMs from a compromised or malicious hypervisor creates a number of opportunities as compared to the current IaaS cloud computing model.

Private Data and Proprietary Code: With hypervisor-secure virtualization, now guest VMs have truly private memory they can use to store data and proprietary code. The VMs could download encrypted data or code from the customers once the VM protections are enabled and verified; the hypervisor or other VMs cannot access the plaintext of this data or code. This would enable a whole new set of companies, for example ones dealing with health records, to freely use cloud computing. What new services are possible by enabling private information to be securely stored in the cloud? What benefits can be gained from securely running proprietary code in the cloud?

Attestation: The proposed new hypervisor-secure virtualization includes the ability to provide trust evidence, i.e., attest, to the customer that a correct guest OS was initialized and protections are being enforced. Today, the customers have to trust the infrastructure provider's record of (reported) security breaches to judge how secure the leased environment is. How can attestation empower customers to migrate their facilities to the IaaS services?

Untrusted Hypervisors: We propose that the whole hypervisor kernel can be removed from the TCB. Excluding the DoS attacks, can this radical idea work?

VI. CONCLUSION

In this paper, we have described a new research direction for security in cloud computing. By investigating new microprocessor features which can be added to multicore processor chips, the guest VMs can be provided protections from a compromised or malicious hypervisor. The key to *hypervisor-secure virtualization* is that the hypervisor is able to dynamically and freely manage the resources of the physical platform. However, once a VM is started, new hardware features protect the VM from the hypervisor, per the customer's specification, for the lifetime of the VM. This creates a new situation where running a guest OS and applications in the cloud is as secure as running the OS and applications on one's own server in one's office.

REFERENCES

[1] Amazon EC2. http://aws.amazon.com/ec2/.

[2] AMD Virtualization (AMD-V) Technology. http://sites.amd.com/us/business/it-solutions/virtualization/Pages/amd-v.aspx.

[3] Intel Corporation: Intel Virtualization Technology for Directed I/O. http://download.intel.com/technology/itj/2006/v10i3/v10-i3-art02.pdf.

[4] Intel Virtualization Technology. http://www.intel.com/technology/itj/2006/v10i3/1-hardware/6-vt-x-vt-i-solutions.htm.

[5] PCI SIG: PCI-SIG Single Root I/O Virtualization. http://www.pcisig.com/specifications/iov/single_root/.

[6] Vulnerability Summary for CVE-2007-4993. http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-4993.

[7] D. Champagne and R. Lee. Scalable architectural support for trusted software. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1 –12, 2010.

[8] J. S. Dwoskin and R. B. Lee. Hardware-rooted trust for secure key management and transient trust. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 389–400, New York, NY, USA, 2007. ACM.

[9] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. NoHype: virtualized cloud infrastructure without the virtualization. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 350–361, New York, NY, USA, 2010. ACM.

[10] M. A. Kozuch, M. Kaminsky, and M. P. Ryan. Migration without virtualization. In *12th Workshop on Hot Topics in Operating Systems (HotOS)*, 2009.

[11] R. B. Lee, P. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang. Architecture for protecting critical secrets in microprocessors. pages 2–13, Madison, Wisconsin, USA, June 2005.

[12] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural support for copy and tamper resistant software. *SIGPLAN Not.*, 35:168–177, November 2000.

[13] T. Ormandy. An empirical study into the security exposure to hosts of hostile virtualized environments. CanSecWest Applied Security Conference, 2007.

[14] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[15] K. Shimizu, H. P. Hofstee, and J. S. Liberty. Cell broadband engine processor vault security architecture. *IBM Journal of Research and Development*, 51(5):521 –528, 2007.

[16] S. W. Smith and D. Safford. Practical server privacy with secure coprocessors. *IBM Systems Journal*, 40(3):683 –695, 2001.

[17] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Aegis: architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th annual international conference on Supercomputing*, ICS '03, pages 160–171, New York, NY, USA, 2003. ACM.

[18] Z. Wang and X. Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 380 –395, May 2010.

[19] Z. Wang, X. Jiang, W. Cui, and P. Ning. Countering kernel rootkits with lightweight hook protection. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 545–554, New York, NY, USA, 2009. ACM.

[20] Z. Wang and R. B. Lee. A novel cache architecture with enhanced performance and security. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, pages 83–93, Washington, DC, USA, 2008. IEEE Computer Society.

[21] R. Wojtczuk. Subverting the Xen hypervisor. Black Hat USA, 2008.