Towards an Implementation of a Blind Hypervisor

Mehdi Aichouch and Moha Ait Hmid
CEA, List, Software Modules for System Security and Dependability Laboratory
Point courrier 172, F-91191 Gif-sur-Yvette, FRANCE
{mehdi.aichouch@cea.fr, moha.ait-hmid}@cea.fr

Abstract

One major fear of many virtual machines users is a corrupted hypervisor that might violate the privacy of their VM's data. One possible idea to tackle such a problematic situation is to protect the privacy of a virtual machine even though the hypervisor is not trustworthy. The blind hypervisor approach define a set of hardware extensions and software adaptation to prevent a hypervisor from accessing virtual machine's private data even if it has the most privileged access to hardware.

1. Introduction

Running a virtual machine on a cloud computing infrastructure require from an end user to be more and more confident with the distant platform. In particular, a formally verified hypervisor that satisfies the security properties may guarantee that a virtual machine will be protected from any malicious software attack. While formally verifying a microkernel with a small *trusted computing base* (TCB) have been achieved [1], it is still difficult to realize in the case of hypervisors usually used in the cloud computing domain due to their big TCB.

In the absence of a formally verified hypervisor, it is necessary to provide another way of protection to guarantee that a virtual machine will not be prone to malicious attacks not only from the other software managed by the hypervisor but also from the hypervisor itself. This way, if a malicious software that may take control over the hypervisor for example through a privilege escalation breach, could not be able to attack the virtual machine as the hypervisor itself is not authorized to access the VM private memory partition.

2 Blind Hypervisor

A blind hypervisor[2] guarantees the confidentiality and integrity of a virtual machine's data and code. That is, a blind hypervisor does not have read and write accesses to a memory partition reserved for a virtual machine even though it possesses the highest CPU execution privileges.

A virtual machine's data and code are encrypted when stored on a hard disk. A virtual machine's content should be decrypted before its loading into memory. When a VM is preempted the CPU's context and state registers, and the memory caches should be flushed and invalidated prior to the execution of any other code in order to avoid memory leakage.

Implementing a blind hypervisor require a set of hardware extensions. Specifically, to prevent the hypervisor from accessing a virtual machine's data and code the hardware architecture might be extended by a "secure" *memory management unit* (MMU) in addition to a "regular" MMU already present in the architecture. This way, even if a hypervisor that has all privileges to configure the first MMU, the second MMU prevent it from accessing a virtual machine's memory partition based on identification mechanism. For each created software component (hypervisor or VM) a memory partition is reserved and an exclusive access is granted to that partition based on the component identification.

The second required component is a *trusted loader*. It is in charge of the encryption/decryption of the VM's content before its loading from a hard disk into memory and storing it back on a hard disk. It is the only component in the system that has access to the private key to encrypt/decrypt the virtual machine content.

Moreover, a hardware architecture should be extended by two CPU execution modes in addition to the two regular *user/supervisor* execution modes usually used to execute an operating system kernel and applications. A third *hypervisor* mode is added to control the execution of virtual machines, but does not allow to access the memory partitions reserved for the VMs. A fourth mode is an *initialization* mode of the processor that allow the hardware reset and creation of memory partitions. Exiting from the *initialization* mode to *hypervisor* mode is triggered by an execution of a particular instruction that is illegal in all other modes. There is no other way to enter the initialization mode except the reset of the hardware. We note that the three first execution modes are available in today architectures with support to full virtualization. These fourth execution levels imposes an adaptation to a hypervisor. The context-switching mechanism has to be modified whether by implementing it in hardware as a micro-code in the processor, or in a protected software.

3 Implementation on a TSAR Architecture

TSAR[3] is a *cache-coherent non-uniform memory access* many-core architecture (cc-NUMA). It is composed by a set of n clusters with 4 CPUs per cluster, as illustrated in Figure 1.

Each virtual machine is allocated a set of clusters and accesses a private memory segment depending on its set of clusters. Each cluster in TSAR architecture have been extended by a set of hardware units called *Hardware Address Translators* (HAT) that translate the physical memory addresses generated by all the CPUs in a cluster into machine memory addresses in the physical memory space reserved to a VM. The HATs prevent a VM from any illegal access to other VM memory address spaces. An HAT is an implementation of a "secure" MMU component required by a blind hypervisor.

A TSAR cluster includes a cryptographic processor (H-Crypt) responsible of the ciphering/deciphering of the VM content and loading/storing from hard disk into memory. It represents an implementation of a "trusted loader" component. A virtual machine has direct access to the available I/O devices connected to the machine. Once a virtual machine is started the hypervisor does not intervene in the execution of a virtual machine.

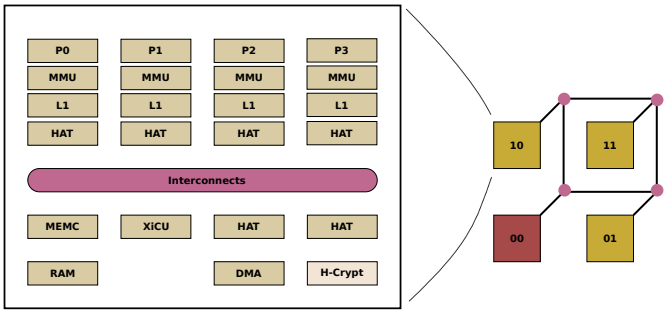

Figure 1. Example of standard cluster in TSAR architecture.

4. Conclusions

In this paper, we presented a blind hypervisor design that guarantee the confidentiality and integrity of a virtual machine data and code by preventing non permitted access to virtual machine private memory. We described a set of hardware extensions to a many-core architecture to support this design. We are currently implementing a blind hypervisor on a 16-clusters TSAR machine and experimenting the execution of multiple protected operating systems.

References

[1] The proof. http://ssrg.nicta.com.au/projects/TS/l4.verified/proof.pml, 2015.

[2] P. Dubrulle, R. Sirdey, E. Ohayon, P. Dore, and M. Aichouch. Blind Hypervision To Protect Virtual Machine Privacy Against Hypervisor Escape

Vulnerabilities. *IEEE International Conference on Industrial Informatics*, june 2015.

[3] A. Greiner. TSAR : a scalable shared memory many-cores architecture with global cache coherence. *In 9th Int. Forum on Embedded MPSoC and Multicore*, 2009.