Survey of Security Problems in Cloud Computing Virtual Machines

Ivan Studnia, Eric Alata, Yves Deswarte, Mohamed Kaâniche, Vincent Nicomette

Survey of Security Problems in Cloud Computing Virtual Machines

Ivan Studnia[1,2], Eric Alata[1,3], Yves Deswarte[1,2], Mohamed Kaâniche[1,2], and Vincent Nicomette[1,3]

[1] CNRS, LAAS, 7 Avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ. Toulouse, LAAS, F-31400 Toulouse, France
[3] Univ. Toulouse, INSA, LAAS, F-31400 Toulouse, France
{studnia,ealata,deswarte,kaaniche,nicomett}@laas.fr

Abstract. Virtualization techniques are at the heart of Cloud Computing, and these techniques add their own vulnerabilities to those traditional in any connected computer system. This paper presents an overview of such vulnerabilities, as well as possible counter-measures to cope with them.

Keywords: virtual machine, hypervisor, vulnerabilities, virtualization-based monitoring, isolation enforcement.

1 Introduction

Virtualization has become an attractive and widely used technology in today's computing. Indeed, the ability to share the resources of a single physical machine between several isolated virtual machines (VM) enabling a more optimized hardware utilization, as well as the easier management and migration of a virtual system compared to its physical counterpart, have given rise to new architectures and computing paradigms. In particular, virtualization is a key element in cloud computing.

However, adding another abstraction layer between hardware and software raises new security challenges. This paper gives an overview of some security problems related to the use of virtualization and shows that the widely used virtual machine managers cannot be considered fully secure. This observation, added to the ever growing popularity of virtualized architectures has led to an important number of studies aiming at enforcing security in virtual systems.

This paper presents first the virtualization principles, then discusses some vulnerabilities related to virtual systems before describing various attempts to address the security challenges raised by virtualization.

2 Context

2.1 Definition

In [21], Popek and Goldberg define a virtual machine as *"an efficient, isolated duplicate of a real machine"*. They also give three necessary conditions to reach this goal.

– Efficiency: Virtualization shall not induce a significant decrease in performance. Therefore, the greatest amount of instructions must not require an intervention from the virtual machine manager (VMM).
– Resource control: The VMM must have a complete control over the virtualized resources.
– Equivalence: A program must behave the same way on a virtual machine as it would do on its physical counterpart.

However, this definition, given in 1974, does no longer cover the whole range of possibilities offered by virtualization. For example, it is now possible to emulate (parts of) systems which do not have a physical equivalent on the host machine. Therefore, new definitions have been established. We will use a definition given in [22]: *"Virtualization is defined as a framework dividing the resources of the device from the execution environment, allowing environment plurality by using one or more techniques such as time-sharing, emulation, partitioning."*

This definition allows us to include emulation or paravirtualization (cf. 2.3) into virtualization and therefore covers a broader range of currently used techniques.

2.2 Classification

Virtualization offers many possibilities, and according to the needs, various kinds of virtualization can be used:

– Process virtualization: virtualizing this layer consists in providing an interface between an application and the underlying system. This allows to create an application without concerning about the specificities of the OSes it will run on, as long as they possess the required virtualization layer. The Java Virtual Machine is an example of process virtualization.
– Server virtualization: here, the virtualization is applied to the hardware. This will allow many OSes to run simultaneously on a physical machine. In this paper, we focus on server virtualization techniques.
– Network virtualization: VPNs (Virtual Private Networks, which enable the interconnection of distinct private networks through a public infrastructure such as the Internet) and VLANs (Virtual LANs, distinct local networks sharing the same physical infrastructure) are examples of network virtualization.
– Storage virtualization: SANs (Storage Area Networks) fall into this category.

2.3 Server virtualization

There are several ways of implementing server virtualization, either by interacting with the hardware, (nPar[4] for example), the OS (Linux-VServer[5]) or through a hypervisor, that is to say a program dedicated to the management of virtual machines. Our study will focus on the latter[6]. These hypervisors are often categorized within two groups:

– Type 1: Type 1 managers are installed directly above the hardware and run with the highest level of privileges. Xen [4] and VMWare ESX [1] are type 1 hypervisors.
– Type 2: Type 2 managers are installed above an operating system, like any other program. QEMU and VirtualBox are type 2 hypervisors.

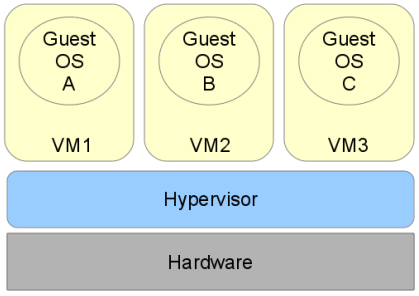

Fig. 1. Type 1 hypervisor

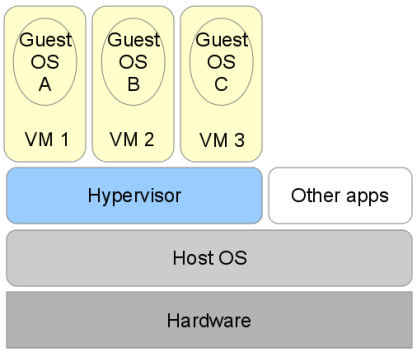

Fig. 2. Type 2 hypervisor

Furthermore, we can distinguish three major techniques of hypervisor based virtualization: full virtualization, paravirtualization and emulation.

Full virtualization In this case, one or more unmodified operating systems (called "guests") are sharing hardware resources from the host system. The presence of the hypervisor is transparent from the guests viewpoint.

Paravirtualization The kernels of the guest systems are modified to make them able to communicate with the hypervisor below them via *hypercalls*. Hypercalls can be seen as equivalents to system calls of a non virtualized OS. Paravirtualization is historically the standard method of virtualization used by Xen.

Emulation Like full virtualization, emulation allows unmodified operating systems to run. However, in that case, the resources seen by the guest OS are completely simulated by software. This allows to execute an operating system compiled on an architecture different from the architecture of the host. This results in lower throughput than with the previously mentioned techniques. QEMU is an example of an emulator.

2.4 Hardware assisted virtualization

Leveraging hardware capabilities to support virtualization has been done for a long time (IBM System/370, 1972). However, hardware assisted virtualization was not implemented on x86 CPUs. This prevented Popek and Goldberg's conditions from being met because some privileged instructions were not correctly trapped. Therefore, the hypervisor had to perform extra tasks in order to address those lacks. This increased the overall complexity and impacted the performance. However, since 2006, Intel VT[7]and AMD-V[8] technologies implement such techniques. A new CPU state was

[4] `http://en.wikipedia.org/wiki/HP_nPar_%28Hard_Partitioning%29`
[5] `http://linux-vserver.org/`
[6] We will use indifferently the terms "hypervisor" and "virtual machine manager" in this paper.
[7] `http://www.intel.com/technology/itj/2006/v10i3/1-hardware/6-vt-x-vt-i-solutions.htm`
[8] `http://sites.amd.com/us/business/it-solutions/virtualization/Pages/virtualization.aspx`

introduced, orthogonal to privilege rings 0-3, called `root mode` on Intel chips and `guest mode` on AMD chips. This state is accessed whenever the hypervisor needs to take (resp. give back) the control over (to) a virtual machine. The guest OS can actually run in ring 0 but not in root mode. Furthermore, the handling of I/O memory virtualization allows to prevent DMA (Direct Memory Access) requests issued from a virtual machine to tamper with unauthorized zones of the host memory.

Although these various features were introduced to facilitate the work of the hypervisor, they can also be used to carry out new kinds of attacks. An example of such an attack is presented in 3.2.

3 Vulnerabilities and attacks

Virtualization technologies offer new economical and technical possibilities. However, the addition of a new layer of software introduces new security concerns. Garfinkel and Rosenblum give in [9] a list of challenges raised by virtualisation that are discussed hereafter.

Scaling. Virtualization enables quick and easy creation of new virtual machines. Therefore, security policies of a network (setup, updates...) have to be flexible enough to handle a fast increase in the number of machines.

Transience. With virtualization, machines are often added to or removed from a network. This can hinder the attempts to stabilize it. For example, if a network gets infected by a worm, it will be harder to find precisely which machines were infected and clean them up when these machines exist only during brief periods of time on the network. Similarly, infected machines or still vulnerable ones can reappear after the infection was thought to be wiped out.

Software lifecycle. The ability to restore a virtual machine into a previous state raises many security concerns. Indeed, previously patched vulnerabilities (programs flaws, deactivated services, older passwords...) may reappear. Moreover, restoring a virtual machine into a previous state can allow an attacker to replay some sequences, which renders obsolete any security protocol based on the state of the machine at a given time.

Diversity. In an organization where security policies are based on the homogeneity of the machines, virtualization increases the risk of having many versions of the same system at the same time on the network.

Mobility. A virtual machine is considered like any other file on a hard drive. It can be copied and moved to another disk or another host. This feature, cited as a benefit of virtualization, also adds security constraints because guaranteeing the security of a virtual machine becomes equivalent to guaranteeing the security of every host it has been on.

Identity. Usual methods used to identify machines (like MAC addresses) are not necessarily efficient with virtual machines. Moreover, mobility increases even more the difficulties to authenticate the owner of a virtual machine (as it can be copied or moved).

Data lifetime. A hypervisor able to save the state of its VMs can counter the efforts made by a guest to delete sensitive data from its memory. Indeed, there may always be a backup version of the VM containing the data.

If many of these challenges can be addressed with good use policies (for cloud computing, examples can be found in [2]), attackers may still exploit flaws in the system to perform their attacks. The rest of this section will focus on these malicious attempts to break into a virtualized system.

3.1 Vulnerabilities

Technically, virtualization has enabled the emergence of new attack scenarios. First, a virtual machine being the equivalent of a physical machine, an attacker willing to take control over it can use the same tools in both cases. On the other hand, the addition of a virtualization layer can give an attacker new ways of subverting its target. Indeed, as shown in figure 3, for a type 2 hypervisor, vulnerabilities in the virtualization layer may allow attacks to be performed from a virtual machine against another VM, the VMM or the host operating system. The picture would be similar for a type 1 hypervisor, except that there is no host OS underneath the hypervisor, so the corresponding attack surface is also removed. If there are any system management tools, these are also considered as a potential attack vector.

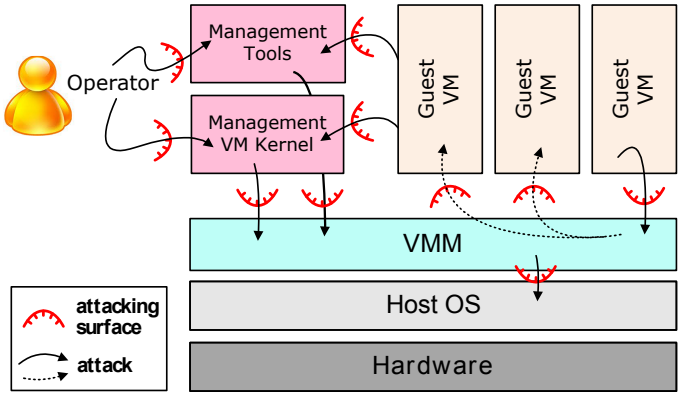

Fig. 3. (after figure 1 from [30]) Attack surfaces on virtual machines

3.2 Attack examples

In this part, we will describe some typical examples of attacks targeting virtualized systems or using virtualization properties to corrupt a machine.

Detecting a virtualized environment. Virtualization technologies (except paravirtualization) are supposed to provide the guest OS with an identical duplicate of a real system. Actually, this

is not completely true and it is therefore possible to detect whether there is a hypervisor running underneath the OS.

Such techniques are useful for both attackers and defenders. On the one hand, an attacker can check if the targeted system is virtualized and acts accordingly. On the other hand, a user willing to check the integrity of his machine can check for the presence of a hypervisor installed against his will under his OS (cf 3.2). [8] gives some generic examples of detection. First, a hypervisor can be detected by checking the execution time of some instructions because the processing of such instructions by the VMM requires more time than when the system is not virtualized. However, such comparisons are possible only if the measurements were previously done on an identical, known to be safe system. [8] also describes a method based on the measurements of the time needed to access the *Translation Lookaside Buffers* (TLB): once those buffers have been filled with some known data, calling the CPUID instruction triggers the cleaning of at least one portion of the TLB if a hypervisor trapped it. Then, by comparing the access times to the TLB measured before and after calling CPUID, one can establish the presence of a hypervisor. One advantage of this approach is that it does not require preliminary baseline measurements.

Identifying the hypervisor. Ferrie describes in [8] the processes he used to identify with precision which of six hypervisors (VMWare, VirtualPC, Parallels, Bochs, Hydra and QEMU) was used. To do so, he used specific instructions that are not handled by some hypervisors the same way as they are on a real system (this can lead to hypervisor-specific exceptions or, on the contrary, to the absence of exception whereas some would have been raised on a real system). Among the known applied examples of such methods, we can cite RedPill[9] and ScoopyDoo[10].

Once a VMM is correctly identified, an attacker can adapt his attack scenario to the vulnerabilities known as characteristic of this VMM. Such techniques also allow the creation of viruses targeting only the systems where a specific kind of hypervisor is running. It should be noted that some of these methods allow to detect and identify various popular hypervisors. However, in most cases they are legitimately installed on a machine, so it is understandable that they do not try to hide themselves.

Breach in the isolation. One of the main goals of a hypervisor is to ensure that the hosted virtual machines are isolated, which means that one guest system is not able to reach more resources than it has been granted, especially the memory used by the other guests or by the host system. However, bad configuration or design flaws within the VMM can allow an attacker to break out of the isolation. This can lead to:

- Denial of service: A virtual machine uses all the computing capacities of the real host, preventing the other VMs from running correctly.
- System halt: A specifically crafted instruction causes the VM or the hypervisor to crash
- *VM escape*: This category covers the situations where an attacker gains access to memory located outside the region allocated to the corrupted VM. The attacker can access the memory of other systems (guest or even host) and can read, write or execute its content. This can lead to a complete takeover of another VM, of the hypervisor or of the entire host system (for a type 2 hypervisor).

[9] `http://invisiblethings.org/papers/redpill.html`
[10] `http://www.trapkit.de/research/vmm/scoopydoo/index.html`

VM escape documented attacks targeting Xen, VMWare and Linux KVM are respectively described in [28], [15] and [7]. In these three situations, an attacker was able to run some custom code with the highest privileges and therefore took over the whole system. This kind of attacks exploits some vulnerabilities in the source code or the design of the hypervisor. In [19], random sequences of code are sent (by fuzzing techniques) to several VMMs. For every tested hypervisor, some sequences have caused a system crash. This is often the first step to discover new exploitable vulenrabilities. Moreover, the amount of code shipped with every new version of a hypervisor seems to be increasing (cf. Table 1), which raises the probability of new vulnerabilities.

Accurate targeting of VMs in a cloud. In this paragraph, we will focus on the works done by Ristenpart et al. [23]. Their research work consists in identifying techniques to ensure a co-residence in a public cloud (Amazon EC2 in their experiments). This means that the authors are able to obtain a VM that is hosted on the same real host as another VM they want to attack. Thanks to this co-residence, they are able to monitor some activities of the targeted VM through a covert channel.

Let us note that, in these experiments, only "legitimate" tools were used and no modification of the targeted systems has been done.

The authors started by establishing a map of the cloud: they examined how the IPs were distributed according to the characteristics of the corresponding VMs (number of cores, storage capacity, ...). Then, they identified several techniques for checking the co-residence between an attacker's VM and his target, first through a network analysis (if the first node is the same, or if the IPs are within a certain range, they are co-resident), then by measuring the host activity. To do so, they compared the time required for reading a given portion of the memory before and during an intense activity imposed to the target (by sending multiple requests to the targeted server). Results of this experiment are shown in figure 4. The target is a web server to which multiple `HTTP` get requests were periodically sent. In the first two graphs, there is a clear distinction between the activity recorded during the `HTTP` gets and the "standard" activity ("`No HTTP gets`"): the two machines are co-resident. In the third graph, no distinction can be made: two machines are not resident on the same real host.

Once that co-residence is confirmed, an attacker can obtain more information about the target activities with a similar method (by monitoring the host activity without interacting with the target) or try to take over it by other means.

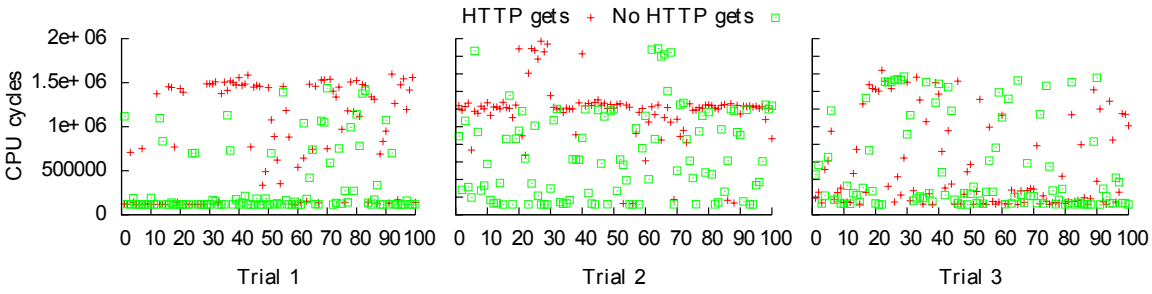

Fig. 4. (Figure 5 from [23]) Establishing co-residence in a cloud through load measurements

Virtual Machine Based Rootkits. The previously described attacks were based on some characteristic features or design flaws of the VMMs. *Virtual Machine Based Rootkits* (VMBRs) are different. Here, the attacker uses some features of hardware-assisted virtualization of x86 processors (cf 2.4) in order to install a hypervisor underneath the targeted OS, putting it into a virtual machine. As the attacker totally controls the hypervisor, he can monitor the whole virtualized OS. Some VMBR implementations can virtualize the OS on the fly (BluePill [24]), others can survive a reboot (SubVirt [13]). Finally, J. Rutkowska [25] and her team were able to install BluePill under Xen after subverting it from one of its VM, therefore putting the hypervisor itself into a VM (*nested virtualization*).

4 Virtualization and security

As seen previously, virtualized systems have many advantages over their physical counterparts. However, they also have their own vulnerabilities (in addition to the "traditional" vulnerabilities inherited from physical systems). Although an important portion of the threats can be prevented by following some rules (for the administrator as well as the users), an attacker may still be able to reach his goal by exploiting some flaws. Many works are therefore published in order to improve the security of virtual environments. We will describe some of them hereafter, but this list is not comprehensive. Like [6], we split up these works in two categories: VM monitoring and isolation enforcement.

4.1 VM monitoring

The hypervisor having higher privileges than the virtualized operating systems, it can be used to monitor their activities "from the outside". This section describes applied examples of this idea.

Virtual machine introspection. Virtual machine introspection (VMI) techniques are based on the ability of the hypervisor to access VM-allocated memory space in order to analyze its content from outside. Data retrieval and analyses can be done directly within the hypervisor or from a dedicated VM (which is recommended because this will require less modifications in the hypervisor code). However, the VMM has basically no knowledge of the abstractions used by the guest OS and is therefore unable to understand the meaning of the retrieved memory zones. This is the *semantic gap* that VMI programs have to fill. There are many VMI implementations, from frameworks (like LibVMI[11] for Xen and KVM) to entire hypervisors, according to one's needs. [18] classifies them according to three characteristics:

- Monitoring or interfering: is the system only watching and reporting any anomaly or can it counter attacks by itself ?
- Semantic awareness: the amount of knowledge the system possesses about the guest OS. For example, Lares[20], using hooks located into the guest system, needs a strong knowledge of this system. On the other hand, Cloudsec [11] knows nothing about the guest OS and tries to fill the semantic gap according to the gathered information.
- Event replay: the ability of the system to record the state of the monitored VM for further analyses.

[11] http://code.google.com/p/vmitools/

Kernel protection. Virtualizing an OS puts it in a less privileged state, the hypervisor getting the highest privileges. The latter can then enforce security in the OS from outside.

SecVisor [26] is a hypervisor designed to ensure the integrity of an OS. It is a small VMM (less than 5000 lines of code) aimed at allowing only the execution of user approved code. The design of SecVisor had to fulfill three criteria:

1. A small amount of code in order to allow a formal checking or a manual audit.
2. A minimal exterior interface in order to reduce the attack surface.
3. The lowest possible amount of changes made into the monitored kernel in order to ease portability.

SecVisor does not prevent code from being added to the kernel, although it forbids its execution unless the code has been approved by the user. To do so, SecVisor relies on hardware features (cf. 2.4) such as hardware memory protection.

The `Hytux` prototype[16], developed at LAAS, is also a lightweight hypervisor that implements protection mechanisms in a more privileged mode than the Linux kernel. It is especially designed to counter kernel rootkits, malware that aims at corrupting the kernel of operating systems.

Rootkit detection. Similarly, Patagonix [17] is a hypervisor designed to detect rootkits hiding into the monitored OS, without relying on the OS capacities. To do so, the code of the virtualized OS is tagged as non executable, which triggers an action from the hypervisor the first time this code is executed of when it has been modified from the previous execution. Then, the code is identified by comparing it with a database of known binaries. Therefore, it becomes possible to detect modified binaries or programs trying to hide their presence to the OS. By using a database to deal with the semantic gap, Patagonix can theoretically monitor any OS as long as it possesses a sufficient database of its correct binaries.

4.2 Isolation enforcement

Some other studies focus on means to prevent a breach in the isolation property. Corrupting the hypervisor (by exploiting design flaws or other vulnerabilities) is a way to break this isolation. Again, various approaches are possible to deal with this problem.

Security modules. This category contains the modules developed to provide security features into a hypervisor. For example, Xen Security Modules [5] is a framework implementing new security rules into Xen that an administrator can include while compiling it.

Another example would be sVirt[12], which uses Mandatory Access Control (MAC) to set a stronger isolation between the VMs in Linux-based virtualization solutions (such as KVM). sVirt assigns a distinct MCS (Multi Category Security) label to each VM but gives the same label to a VM process and its corresponding VM image. Therefore, a VM process will be allowed to read and modify only its corresponding image but any access from one VM process to another or to another image will be prevented. Thus, if one VM becomes compromised, it will not be able to access other VMs (images or processes) hosted on the same computer.

[12] http://selinuxproject.org/page/SVirt

Protecting the VMM. As seen in section 4.1, a hypervisor can be used to monitor the virtualized systems it is hosting. However, as seen in 3.2, the hypervisor can in turn be targeted and modified by an attack. As the hypervisor possesses every privilege on its guest systems, it is crucial to preserve its integrity. However, while it is possible to ensure the integrity of a system during boot (we can for example cite the Trusted Computing Group[13] which works on this topic), it is much harder to ensure runtime integrity. So, to ensure runtime integrity, one could think of installing a second hypervisor under the initial hypervisor dedicated to monitoring it, similarly to 4.1. However, one would have to guarantee that the most privileged hypervisor cannot in turn be corrupted. Several studies have therefore focused on using other means to ensure the integrity of the most privileged element.

First, HyperSentry [3] aims at providing a secure environment to tools willing to enforce the integrity of the most privileged element. For that purpose, it leverages both the SMM (*System Management Mode*) specificities to protect its code from the hypervisor and the Trusted Boot as defined by the TCG [14] to ensure the integrity of its code at startup. Moreover, through the use of communication channels uncontrolled by the hypervisor (like IPMI[15] in [3]), an analysis can be executed without the hypervisor noticing it. Therefore, HyperSentry can access the state of a hypervisor at a given time without making it aware of the process, theoretically preventing it from tampering with the data before it is sent to the analysis tools.

Other works, like HyperSafe [27], try to protect the hypervisor from modifications through the strict enforcement of certain principles. HyperSafe patches an existing hypervisor to enforce two features (*memory lockdown* and *restricted pointer indexing*) aiming at guaranteeing control flow integrity. The objective is to make the hypervisor protect itself against unwanted modifications or executions of its code. HyperSafe has already been ported on Bitvisor and (not yet completely, according to [27]) on Xen.

However, if such tools allow to enhance the ability of a hypervisor to protect itself, using them implies increasing the amount of code running with high privileges (since it is part of the hypervisor). Thus, any design mistake present into these modules can become an entry point for new attacks. Such an example is detailled in [29], where an attack is made possible only if the FLASK module for XSM is installed, which is not the default configuration for Xen.

Protecting the VMs against their VMM. The purpose of CloudVisor [30] is to ensure data confidentiality and integrity for the VM, even if some elements of the virtualization system (hypervisor, management VM, another guest VM) are compromised[16]. The idea is that data belonging to a VM but accessed by something else than this VM appears encrypted. To reach its goal, CloudVisor virtualizes the monitored hypervisor (realizing nested virtualization), therefore removing the latter from the most privileged zone while still giving it the illusion of the opposite. This means that the monitored VMM is now running in *guest mode* while CloudVisor is the only one in *root mode*. Any access, requested by the VMM, to some memory belonging to a VM is then trapped by CloudVisor. If the access is not requested by the owner of the requested page, CloudVisor encrypts its content. Such a concept seems particularly interesting within a cloud context, where multi-tenancy

[13] `http://www.trustedcomputinggroup.org`

[14] `http://www.trustedcomputinggroup.org/resources/trusted_boot`

[15] *Intelligent Platform Management Interface* : `http://download.intel.com/design/servers/ipmi/IPMIv2_0rev1_0.pdf`

[16] Let us note that CloudVisor does not try to protect a VM from any other kind of attacks.

(unrelated users sharing the same physical resources) can be the norm and where privacy therefore represents one of the main concerns of cloud customers.

Microkernels. Microkernels result from the will to design an operating system kernel in another way: a feature shall be allowed in kernel space only if its removal (and execution in user space) prevents the system from working correctly. Everything else is designed as user space modules, communicating between themselves through IPC (*Inter Process Communication*). Figure 5 (source Wikipedia[17]) shows that with such a principle, there is much less code running in kernel space in a microkernel than in a so-called monolithic kernel (such as the Linux kernel). This makes a microkernel critical code easier to audit.

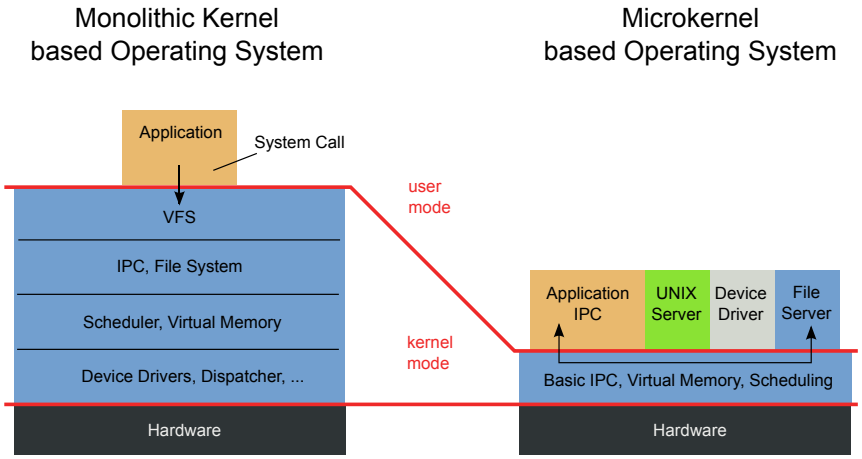

Fig. 5. Overview of the differences between a monolithic kernel and a microkernel

Conceptually, since a privileged module deals with isolation and communication between the processes running above it, microkernels are close to hypervisors. However, only the formers have a claim on minimal size. Those similarities [10] allow some microkernels to work as virtual machine managers. Among these is L4 [18] for which one implementation, called seL4, was formally proven to be secure [14]. This means that seL4 is theoretically flawless (unless a flaw is found in the proof system itself, or in the underlaying layers: memory protection, context switching, hardware management, etc.). Running such a microkernel with the highest level of privileges theoretically prevents their compromission. The use of microkernels, with their smaller amount of critical code, as a basis for virtualization can also help to counter the trend of the increasing code size into some hypervisors (example with Xen in Table 1).

Removing the VMM. NoHype [12] represents yet another case. The hypervisor being a potential attack surface between two virtual machines, one solution to remove this surface is to get rid of

[17] `http://en.wikipedia.org/wiki/Microkernel`
[18] `http://os.inf.tu-dresden.de/L4/overview.html`

Table 1. Amount of lines of critical code into Xen, after [30]

	VMM	Dom0 Kernel	Tools	Total
Xen 2.0	45K	4,136K	26K	4,207K
Xen 3.0	121K	4,807K	143K	5,071K
Xen 4.0	270K	7,560K	647K	8,477K

the hypervisor. This is made possible by using hardware assisted virtualization and putting several constraints on the conditions of virtualization: each virtual machine is linked to one unique processor core, the portion of memory allowed to it is fixed and managed by the hardware (see 2.4) and each (virtual) device is dedicated to only one VM. Figure 6 gives an overview of NoHype implementation. MMC stands for *Multi-core Memory Controller*. It is the device dedicated to managing the allocation of memory to each core. The system manager (on the left) also runs on one dedicated core, which is the only one allowed to send IPIs (*Inter Processor Interrupts*) to other cores, enabling the startup, shutdown or migration of a VM on them. For example, to start a VM, the system manager prepares the required resources and sends an IPI to the core on which the VM is supposed to run. This core runs a program responsible for configuring it (by mapping the allocated resources) then starts the VM image. The effective virtualization of the guest OS is realized by a tiny hypervisor, called *core manager* (the little grey squares on the schema). The system manager is then able to communicate exclusively with the core managers and only to trigger the migration or termination of a VM. However, due to its design, NoHype loses some features offered by more traditional virtual machine managers, such as the ability to share resources (devices, memory buffers) between several VMs.

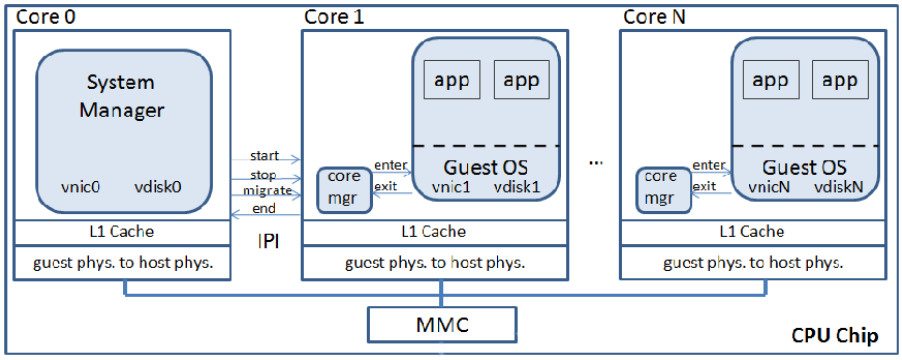

Fig. 6. NoHype architecture (excerpt from [12])

5 Conclusion

Virtualization technologies can bring many interesting new features (optimized use of hardware resources, eased restauration, machine migration, ...), but they also introduce new means to perform attacks. These attacks may either be directed against virtualized systems or leverage some features

related to virtualization in order to take over a system. Therefore, especially if a system is shared between many users, as is the case in cloud computing, strong security of virtual machines is crucial to protect their data and gain the customer's trust. We have presented various implementations dealing with those concerns on two main topics:

- Leveraging virtualization to secure a system located on top of a hypervisor.
- Securing the VMM. This also raised issues concerning the security of the most privileged element of a system.

The research on these topics is very active today, following a great diversity of possibilities, going from devising ways to secure popular systems like Xen, KVM or VMWare solutions, to creating new models such as the microkernel-based virtualization. However, even if these solutions are satisfying security-wise, one should still consider the possible issues of their large-scale deployment. Indeed, additional control procedures will cause a decrease in efficiency. One must therefore find the right balance between performance and security, according to their needs.

Acknowledgements

This research has been partially supported by the French ANR SOP project (ANR-11-INFR-0001), and by the French project Investissements d'Avenir SVC (Secure Virtual Cloud).

References

1. VMWare ESX homepage. `http://www.vmware.com/products/vsphere/esxi-and-esx/index.html`
2. Alliance, C.S.: Security guidance for critical areas of focus in cloud computing v3.0. Cloud Security Alliance (2011)
3. Azab, A., Ning, P., Wang, Z., Jiang, X., Zhang, X., Skalsky, N.: HyperSentry: enabling stealthy in-context measurement of hypervisor integrity. In: Proceedings of the 17th ACM conference on Computer and communications security. pp. 38–49. ACM (2010)
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. ACM SIGOPS Operating Systems Review 37(5), 164–177 (2003)
5. Coker, G.: Xen Security Modules (XSM). Xen Summit (2006)
6. Douglas, H., Gehrmann, C.: Secure virtualization and multicore platforms state-of-the-art report. Tech. rep., SICS, Swedish Institute of Computer Science (2009)
7. Elhage, N.: Virtunoid: Breaking out of KVM. Black Hat USA (2011)
8. Ferrie, P.: Attacks on more virtual machine emulators. Symantec Technology Exchange (2007)
9. Garfinkel, T., Rosenblum, M.: When virtual is harder than real: Security challenges in virtual machine based computing environments. In: Proceedings of the 10th conference on Hot Topics in Operating Systems-Volume 10. p. 20. USENIX Association (2005)
10. Heiser, G., Uhlig, V., LeVasseur, J.: Are virtual-machine monitors microkernels done right? ACM SIGOPS Operating Systems Review 40(1), 95–99 (2006)
11. Ibrahim, A., Hamlyn-Harris, J., Grundy, J., Almorsy, M.: Cloudsec: A security monitoring appliance for virtual machines in the IaaS cloud model. In: Network and System Security (NSS), 2011 5th International Conference on. pp. 113–120. IEEE (2011)
12. Keller, E., Szefer, J., Rexford, J., Lee, R.: Nohype: virtualized cloud infrastructure without the virtualization. In: ACM SIGARCH Computer Architecture News. vol. 38, pp. 350–361. ACM (2010)
13. King, S., Chen, P.: Subvirt: Implementing malware with virtual machines. In: Security and Privacy, 2006 IEEE Symposium on. pp. 14–pp. IEEE (2006)

14. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., et al.: sel4: Formal verification of an os kernel. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. pp. 207–220. ACM (2009)
15. Kortchinsky, K.: Cloudburst—a vmware guest to host escape story. Black Hat USA (2009)
16. Lacombe, E., Nicomette, V., Deswarte, Y.: Enforcing kernel constraints by hardware-assisted virtualization. Journal in computer virology 7(1), 1–21 (2011)
17. Litty, L., Lagar-Cavilla, H., Lie, D.: Hypervisor support for identifying covertly executing binaries. In: Proceedings of the 17th conference on Security symposium. pp. 243–258. USENIX Association (2008)
18. Nance, K., Bishop, M., Hay, B.: Virtual machine introspection: Observation or interference? Security & Privacy, IEEE 6(5), 32–37 (2008)
19. Ormandy, T.: An empirical study into the security exposure to hosts of hostile virtualized environments. In: Proceedings of CanSecWest Applied Security Conference (2007)
20. Payne, B., Carbone, M., Sharif, M., Lee, W.: Lares: An architecture for secure active monitoring using virtualization. In: Security and Privacy, 2008. SP 2008. IEEE Symposium on. pp. 233–247. IEEE (2008)
21. Popek, G., Goldberg, R.: Formal requirements for virtualizable third generation architectures. Communications of the ACM 17(7), 412–421 (1974)
22. Ramos, J.: Security challenges with virtualization (2009)
23. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: 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. pp. 199–212. ACM (2009)
24. Rutkowska, J.: Subverting vista™ kernel for fun and profit. BlackHat Briefings USA (2006)
25. Rutkowska, J., Tereshkin, A.: Bluepilling the xen hypervisor. Black Hat USA (2008)
26. Seshadri, A., Luk, M., Qu, N., Perrig, A.: Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In: ACM SIGOPS Operating Systems Review. vol. 41, pp. 335–350. ACM (2007)
27. Wang, Z., Jiang, X.: Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In: Security and Privacy (SP), 2010 IEEE Symposium on. pp. 380–395. IEEE (2010)
28. Wojtczuk, R.: Subverting the Xen hypervisor. Black Hat USA 2008 (2008)
29. Wojtczuk, R., Rutkowska, J., Tereshkin, A.: Xen 0wning trilogy. In: Black Hat conference (2008)
30. Zhang, F., Chen, J., Chen, H., Zang, B.: Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 203–216. ACM (2011)