Virtualization Technologies and Cloud Security: Advantages, Issues, and Perspectives

Roberto Di Pietro[1](✉) and Flavio Lombardi[2]

[1] Information and Computing Technology Division, College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar
rdipietro@hbku.edu.qa

[2] Istituto per le Applicazioni del Calcolo, Consiglio Nazionale delle Ricerche, Rome, Italy
flavio.lombardi@cnr.it

Abstract. Virtualization technologies allow multiple tenants to share physical resources with a degree of security and isolation that cannot be guaranteed by mere containerization. Further, virtualization allows protected transparent introspection of Virtual Machine activity and content, thus supporting additional control and monitoring. These features provide an explanation, although partial, of why virtualization has been an enabler for the flourishing of cloud services. Nevertheless, security and privacy issues are still present in virtualization technology and hence in Cloud platforms. As an example, even hardware virtualization protection/isolation is far from being perfect and uncircumventable, as recently discovered vulnerabilities show. The objective of this paper is to shed light on current virtualization technology and its evolution from the point of view of security, having as an objective its applications to the Cloud setting.

Keywords: Virtualization · Security · Cloud

1 Introduction

The advances in virtualization technology of the past decade have rendered the Cloud approach feasible and convenient. Nevertheless, the main limitation of virtual machines is that they were born as a means to easily migrate from physically deployed services to more compact and manageable images. In fact, each and every VM runs its own full operating system together with the various libraries required by the application (see Fig. 1) [35]. Such an approach multiplicates the usage of RAM, CPU, and storage with respect to simply hosting multiple services as separate processes on a single piece of bare metal.

Containerization technology is intended to replace hypervisor and VMs, and deploys each application in its own process-like environment running on the physical machine on a single operating system [42]. Containers can be provisioned (and deprovisioned) in a few seconds and make a more efficient usage

© Springer Nature Switzerland AG 2018
P. Samarati et al. (Eds.): Jajodia Festschrift, LNCS 11170, pp. 166–185, 2018.
https://doi.org/10.1007/978-3-030-04834-1_9

WIZ, Inc. EXHIBIT - 1028
WIZ, Inc. v. Orca Security LTD.

of resources, achieving a much higher application density (orders of magnitude [37]) than virtualization. This renders containers much more convenient than virtual machines.

Nevertheless, as we will show along this paper, virtualization is not on a dead path. In fact, virtual machines provide additional security mechanisms and isolation benefits in many application scenarios that are often worth the additional resource usage [28,39].

A virtualization environment generally consists of three core components: an hypervisor or Virtual Machine Manager (also VMM in the following), management tools, and Virtual Machines (VMs). In particular, the infrastructure-as-a-service (IaaS) Cloud layer directly leverages and exposes powerful virtualization technologies and resources to a remote user [3]. Nevertheless, virtualization technologies also introduce additional security concerns. The size of the attack surface for the virtualization approach is directly proportional to the amount of emulated physical resource or functionality that must be provided in software. As regards containers, they can leverage all services offered by the host OS, so the issue here is to enforce effective security and isolation among processes. This is actually more difficult to do, since OSes have not been designed with this in mind. Further, the partitioning/virtualization modes and ISAs[1] of recent CPU and GPU cannot be used by containers, as they are inherently part of virtualization and introduce the actual performance penalties of traditional VMs. Unikernels can be considered an alternative to both containerization and virtualization. They maintain some of the benefits of other approaches (lightweight and isolated) but introduce further issues such as manageability, monitoring and reliability.

In this paper, we survey various aspects of virtualization, analyze their impact on security, and discuss future perspectives. In particular, we provide technology background for most widespread virtualization tools in order to highlight features, advantages and potential security flaws, with a focus on their application to Cloud. Further, discussions and comparisons with containerization and unikernel approaches are introduced throughout the paper.

The sequel of this paper is organized as follows: a technology background is provided in Sect. 2; most relevant virtualization security issues are introduced in Sect. 3; virtualization-based security approaches are presented in Sect. 4; novel enclave technology is discussed in Sect. 5; virtualization-based use cases, together with some future research trends, are presented in Sect. 6; and, finally, conclusions and hints for future work are given in Sect. 7.

2 Technology Background

Various different virtualization technologies are currently deployed in the Cloud, mostly for x86_64 architectures (e.g., Xen, KVM, VMware, VirtualBox, and HyperV). Most relevant details on virtualization frameworks and on supporting hardware (CPU/GPU) features are given and discussed in the following sections.

[1] Instruction Set Architecture(s).

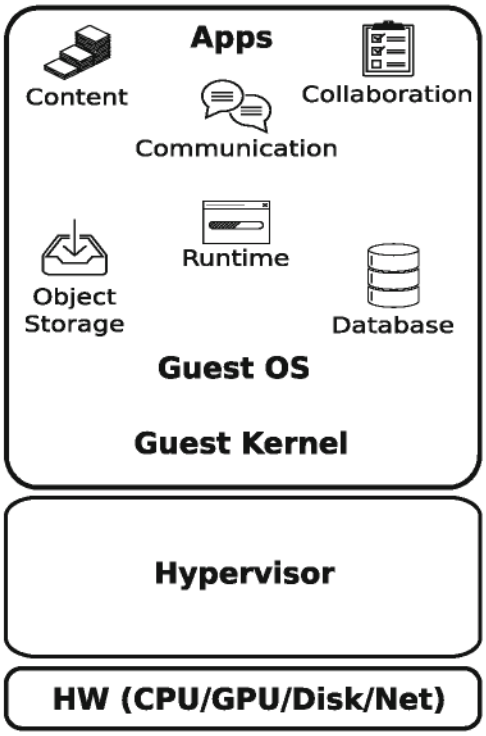

Fig. 1. Cloud layers and virtualization

2.1 Virtualization Frameworks

The essential characteristics of the most widespread virtualization environments are summarized in Table 1. It is worth noting that all present hypervisors support full virtualization (also hardware-assisted virtualization in the following), as it offers relevant performance and isolation benefits. In fact, hardware virtualization allows the CPU to detect and possibly block unauthorized or malicious access to virtual resources. Nevertheless, no virtualization framework is immune to bugs. The virtualization platform can be an additional attack surface.

2.2 CPU Virtualization

The introduction of virtualization-enabling extensions in Intel and AMD CPUs dates back to 2005 [1,25]. VT-x and AMD-V were developed to add an additional more privileged execution ring where an hypervisor or virtual machine manager (VMM) could supervise actual access to physical resources from less privileged execution rings, as depicted in Fig. 2.

CPUs are required to support some advanced extensions in order to allow the hypervisor to leverage them, as can be seen in Table 1. More in detail:

- **Intel VT-x AMD-V**: These two CPU capability sets are the basic ingredient of hardware-supported virtualization. They introduce *Ring –1* allowing a guest virtual machine to run its kernel at standard privilege level (i.e., Ring 0);

Table 1. CPU-related virtualization features

X86_64 hypervisor	Open source	Hypervisor type	Supported extension(s)
Xen	Y	Native	VT-x, AMD-V, EPT, RVI, VT-d, AMD-Vi
KVM	Y	Hosted	VT-x, AMD-V, EPT, RVI, VT-d, AMD-Vi
VMWare ESX	N	Native	VT-x, AMD-V, EPT, RVI, VT-d, AMD-Vi
Hyper-V	N	Native	VT-x, AMD-V, EPT, RVI, VT-d, AMD-Vi
VirtualBox	Y	Hosted	VT-x, AMD-V

- **Intel EPT, AMD RVI**: Rapid Virtualization Indexing and Extended Page Tables, i.e. the Support for Second Level Address Translation (SLAT) that can significantly improve performance;
- **Intel VT-d, AMD-Vi**: These CPU capabilities (directed I/O) allow faster I/O resource virtualization.

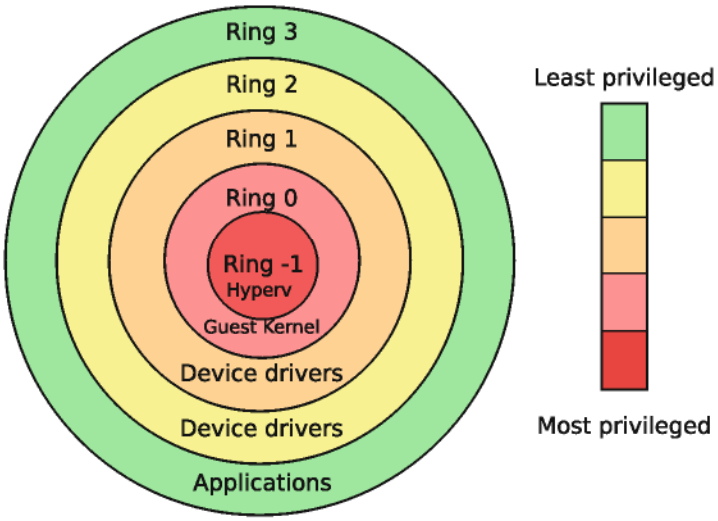

Fig. 2. Execution rings for the x86_64 architecture. See also [19]

2.3 GPU Virtualization

The virtualization paradigm also applies to Graphics Processing Units (GPUs). Virtual machines can be given mediated or full access to GPU computing and memory resources. This allows offering a GPU-based Cloud similar to what is in place already for CPU-based computing resource sharing. Hypervisor support for GPU virtualization features (see Table 2) is still somehow limited as relevant GPU technology is still reserved for high-end GPUs. In fact, GPU virtualization is usually implemented following one of these main approaches [24]:

- **time-sharing**: a single VM at a time is given direct access to the GPU. Time-slots are handled by the hypervisor;

- **passthrough**: the GPU is directly and permanently connected to a single VM that has direct access to it;
- **partitioned**: the GPU resources are split into smaller virtual GPUs, assigned to single VMs.

Once VMs have access to the GPU, the interaction between the guest and the real resource can be achieved in two different ways: backend virtualization or frontend virtualization [17]. Backend virtualization gives a direct connection between the VM and the GPU hardware. Frontend virtualization poses an intermediate layer between the guest and the hardware that has to leverage some kind of intermediate APIs to access the GPU. Some frontend virtualization examples are gVirt [56], vCUDA [53], GViM [22] and VOCL [59].

Table 2. GPU-related virtualization features

X86_64 hypervisor	Open source	Supported GPU virtualization technologies
Xen	Y	Intel GVT-g, AMD MxGPU
KVM	Y	Intel GVT-g, AMD MxGPU
VMWare ESX	N	Intel GVT-g, AMD MxGPU
Hyper-V	N	-
Virtualbox	Y	-

Particularly relevant here is AMD MxGPU technology [58], a partitioning strategy allowing users to have an equal share of the GPU. This hardware-based virtualization solution helps guaranteeing some isolation among different workloads and users.

Intel GVT-g [56] is a full GPU virtualization solution with mediated passthrough (VFIO[2] mediated device framework based). A virtual GPU instance is maintained for each VM, with part of performance critical resources directly assigned. The capability of running native graphics driver inside a VM, without hypervisor intervention in performance critical paths, achieves a good balance among performance, feature, and sharing capability.

As GPUs are mainly used for computation tasks, security concerns about GPU virtualization are mainly focused on data leakage [16]. This can occur either by directly access data owned by the victim and stored within the GPU memory or by exploiting side channels. In [41], Christin et al. have depicted two adversary models:

- **serial adversary**: this attacker has access to the same GPU or to the same GPU memory of the victim, before or after the victim. Hence, it can seek for traces previously left by the victim in different GPU memories;
- **parallel adversary**: this attacker has access to the same GPU or GPU memory of the victim but in the same moment.

[2] Virtual Function I/O.

3 Virtualization Security Issues

Virtualization technologies underlying Cloud computing infrastructure themselves constitute vulnerable surface. In a Cloud scenario, we can observe the following major security challenges [35]:

- **privileged user access**: access to sensitive data in the Cloud has to be restricted to a subset of trusted users (to mitigate the risk of abuse of high privilege roles);
- **lack of data/computation isolation**: one instance of customer data has to be fully isolated from data belonging to other customers;
- **reliability/availability**: the Cloud provider has to setup an effective replication and recovery mechanism to restore services, should a security issue occur;

Virtualization potentially widens Cloud computing attack vectors such as:

- **hypervisor**: the hypervisor is the software element sitting in between the host and guests to allow mediated access to physical resources. This layer should be transparent to a non-privileged user running into the guest. Unfortunately, its presence cannot be fully hidden [46]. As such, an attacker can exploit hypervisor vulnerabilities to gain access to both the host system and other guests. Hypervisors also provide emulation capabilities for missing hardware elements. However, this is a potential attack surface, as demonstrated by Ray [47] and Geffner [26];
- **pivoting**: users can often login into specific services hosted by a VM. Once inside, the attacker could also exit the virtual machine she accessed, to damage the underlying physical system and/or sibling VMs.
- **migration**: virtual machines can be moved over different hosts for load balancing or disaster recovery. This "migration" is performed by copying the VM image over the network. An attacker can potentially eavesdrop data and perform a man in the middle attack if the channel is not encrypted.
- **resource allocation**: virtual machines are usually executed on-demand at run-time, thus making the resource allocation and management process as dynamic as possible. Resource sharing can thwart the security of the host system as well as of its virtual machines. In fact, negligence in cleaning resources before releasing them to others can lead to severe data leakage. As an example, data written by a VM into volatile or persistent storage can be accessed by others who have access to the same elements [50];

The above attacks show how virtual machines and the physical machines hosting them can be thwart by attackers targeting the host or just the virtual machine. Some mitigating approaches can be as follows:

- **host side**: vulnerabilities in the implementation of the hypervisor can somewhat be mitigated by frequently updating the hypervisor to reduce 0-days vulnerability window;

- **network monitoring**: monitoring and analyzing internal communications between sibling guests can help; nevertheless, malicious network behavior is difficult to detect by means of traditional intrusion detection systems and intrusion prevention systems;
- **encryption**: to mitigate such migration attacks encryption of the data in transit can be used; nevertheless, this proves quite demanding on performance, and consequently on costs.
- **on allocation**: this attack can be dealt with by carefully deleting/cleaning resources either persistent or volatile that have been previously assigned to other VMs;

3.1 Co-location Issues

Co-location of virtual machines by different tenants on the same physical host is particularly frequent in Cloud computing. Virtual resources assigned to a tenant might get hacked by other virtual resources assigned to different tenants that are co-located within the same physical machine. Co-location can lead to different issues as follows:

- **information leakage**: by reusing the same physical hardware to allocate virtual resources, tenants might be able to exploit forensic tools to recover sensitive data from previous tenants;
- **performance degradation**: malicious tenants co-located in the same physical host might be able to make an uneven/widely varying use of computational power with high cpu-intensive co-located virtual machines with the final goal of degrading victim's performances;
- **service disruption**: malicious tenants sharing physical resources with their victim might be able to lead the hardware to unexpected behaviors thus causing a service disruption against the victim.

A large number of research results have highlighted the actual existence of co-location vulnerabilities [48,61]. Such papers show that completely preventing tenants from sharing the same physical resources is practically unfeasible (due to rising costs). A viable solution [3] might be an attribute-based approach where tenants can express constraints over both virtual and physical resource allocation. Tenants would be able to indicate an high data sensitivity, thus requesting to avoid co-location. In this way, co-location will not be allowed for virtual resources working on high sensitive information thus lowering the chance of data leakage. As a consequence, virtual resource cost would be increased. This could be an acceptable trade-off in most sensitive scenarios.

3.2 Randomness and Virtualization

Cloud providers usually deploy identical VM clones when needed to satisfy request load. As such, it often happen that the very same images are used for different tenants. As a consequence, the internal random pool for clone VMs is

most probably the same/very similar for different VMs [20]. An adversary might exploit this weakness and try to guess the value of VM cryptographic keys [49]. In order to address such issue, the Cloud or Service providers should try to increase the number of events fed to the entropy pool of VM operating systems as soon as they are deployed, so as to provide an adequate level of security.

3.3 Container Security

The need for cost savings and shorter development cycles induced the succes of containers in the Cloud. Containers are lighter than virtual machines and provide near-native performance. Docker [18] is the current market leader, providing a fully-featured packaging tool. Nevertheless, as introduced above, Containers provide much less isolation to applications, as such mechanisms are not based on hardware features but on process isolation approaches. Among other interesting works, Martin et al. [10] discuss Docker security real-world implications define an adversary model and describe several vulnerabilities affecting current Docker usage. The very same authors [40] detail Docker vulnerabilities and identify several vulnerabilities present by design or introduced by some original use-cases. Albeit some practical countermeasures are proposed, it is clear the containerization approach cannot guarantee an adequate level of security and protection in many multi-tenant scenarios.

3.4 Unikernel Security

The container limitation in providing actual isolation can be addressed by Unikernels, leveraging hardware virtualization to provide a potentially better alternative to containers (at least from the security point of view). Unikernels are specialized lightweight virtual machines (VMs) that squeeze the guest operating system and userspace layers together into one single VM layer [38]. This provides a smaller footprint, and a minimal attack surface. However, managing the privileges of thousands of unikernels is often difficult and error prone. An interesting approach is proposed in VirtusCap [52], a multi-layer access control architecture and mechanism leveraging unikernels. VirtusCap limits privileges of unikernels using the Principle of Least Privilege to create unikernels that have only the privileges they need to accomplish their task.

3.5 Virtualization and Spectre/Meltdown

Spectre [30] and Meltdown [34] are recently discovered CPU vulnerabilities stemming from hardware-implemented performance optimizations aimed at reducing CPU-memory access latencies. Spectre leverages the fact that the speculative execution resulting from a branch misprediction may leave observable side effects that may reveal private data to attackers. In fact, when the memory access pattern depends on private data, the resulting state of the data cache constitutes a side channel an attacker can leverage to extract information about the private data.

Meltdown allows a userspace process to read all memory, even beyond its access scope. Like Spectre, the problem lies with speculative machine code execution that allows cache-timing attacks to leak data from any existing memory address.

Both Spectre and Meltdown are serious security vulnerabilities, in particular since they have been proven to even bypass CPU isolation features guaranteed by hardware-assisted virtualization. The reason why is that they are tied to hard-coded CPU optimizations that involve reusing (i.e. not deleting) cached values even though they belong to different (even security) contexts. Nevertheless, Containers and Unikernels are also vulnerable. As such, mitigating such hardware/firmware bugs is mandatory for any kind of co-location and multi-tenancy of the same physical CPU.

4 Virtualization Benefits for Security

Virtualization technologies also constitute a privileged point of view for observing and tracing VM activity. This can be used to collect useful data, analyze them, and act accordingly.

4.1 Virtual Machine Monitoring

A core set of requirements that a security monitoring system for the Cloud should meet can be summarized as follows [35]:

- **effectiveness**: the system should be able to detect attacks and integrity violations.
- **accuracy**: the system should be able to avoid false-positives, i.e, mistakenly detecting malware attacks where authorized activities are taking place.
- **transparency**: the system should minimize detectability from inside guests, i.e., potential intruders should not be able to detect the presence of the monitoring system.
- **robustness**: the host system, Cloud infrastructure and the sibling VMs should be protected from attacks proceeding from a compromised guest and it should not be possible to disable or alter the monitoring system itself.
- **reactivity**: the system should either be able to take action against both the attempt and the compromised guest, or notify other security-management components.
- **accountability**: the system should not interfere with Cloud and Cloud application actions, but collect data and snapshots to enforce accountability policies.

Nevertheless, satisfying these requirements is quite difficult, as there is a clear trade-off between transparency and reactivity. Possible mitigation approaches include:

- **hiding reaction**: i.e. leveraging regular guest maintenance actions as a reaction. E.g., halting the guest, restarting a fresh image, migrating the VM instance.
- **delaying reaction**: snapshotting the current status and delaying performing reactive activity. Nevertheless, the adversary might be able to perform further activity before being stopped.

In fact, a viable approach to achieve integrity protection is to continuously monitor key components that would most probably be targeted by attacks. We have shown (see also [35]) that by either actively or passively monitoring kernel or middleware components, it is actually possible to detect modifications to kernel data and code, thus guaranteeing that kernel and middleware integrity have not been compromised. A fully asynchronous monitoring system can be a viable solution [15] to provide protection and advanced transparent introspection capabilities to an hypervisor, as detailed in the following.

4.2 Semantic Introspection and Modeling VM Behavior

Monitoring key Cloud components that would be targeted or affected by attacks is vital in order to protect the VMs and the Cloud infrastructure [2]. By either actively or passively monitoring key VM components any possible modification to VM data and code can be traced and recorded.

In fact, virtual machine introspection is a process that allows observing the state of a VM from outside of it. Syringe [7] is one example of a monitoring system making use of virtualization to observe and monitor guest kernel code integrity from a privileged VM or from the VMM. However, it is quite simple for guest code to realize it is running inside a VM that can potentially be a honeypot VM [33].

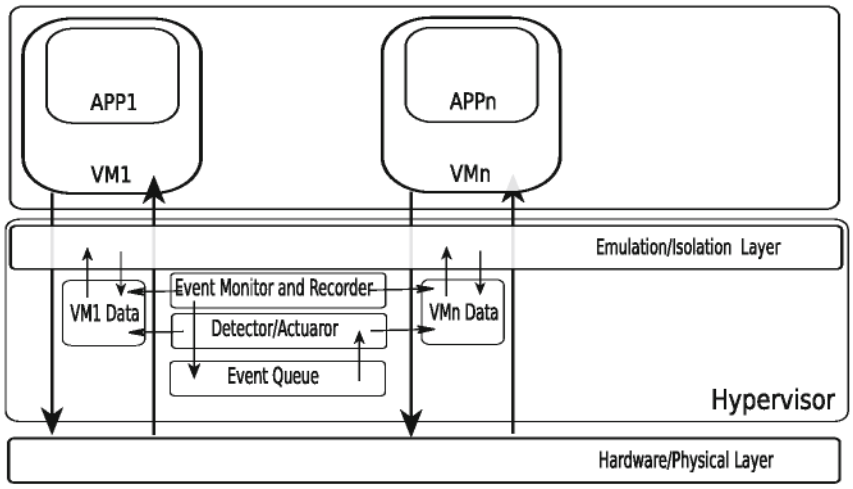

Fig. 3. Virtualization: introspection components

The approach depicted in Fig. 3 is an example of advanced transparent passive tracing and recording of VM events from the hypervisor [35]. Any relevant event or status change is recorded by an event interceptor and it is then stored in a pool of recorder warnings where the collected information is asynchronously evaluated (evaluator) and, if needed, a reaction is triggered (act) according to a chosen policy.

An interesting VM-introspection-based approach is CloRExPa [15], providing various kinds of customizable resilience service solutions for Cloud guests, using execution path analysis. CloRExPa can trace, analyze and control live VM activity, and intervened code and data modifications, possibly due to either malicious attacks or software faults. Execution path analysis allows the VMM to trace the VM state and to prevent such a guest from reaching faulty states, leveraging scenario graphs.

This trend towards semantic introspection of VM activity is a very active field also as regards mobile devices in the Cloud [27]. This is the way to go for enabling control over possibly untrusted mobile Cloud nodes/applications. In fact, as discussed above also for BYOD untrusted devices, either they have to be banned altogether from the enterprise or enhanced semantics-aware introspection has to be put in place to prevent them from leaking sensitive information. Outside of the enterprise, semantic introspection allows legitimate users to regain control over their device internals. This approach will help detect and react to malware and to backdoors that are put in place even by trusted software or apps.

The main problem with introspection is that it requires knowing the internals and semantics of guest operating systems and running applications. This is especially difficult in case of closed-source OS and application such as in Windows and Mac environments. In fact, Windows OSes have always been the main target of malware that have exploited numerous bugs and vulnerabilities exposed by its implementations [36]. Recent trusted boot technology plus additional integrity checks have rendered the Windows OS less vulnerable to kernel-level rootkits. Nevertheless, guest Windows Virtual Machines are becoming an increasingly interesting attack target. HyBIS [14] is the only example of introspection system protecting present Windows OS Guests from malware and rootkits.

4.3 Finer-Grained Security

Some other approaches are available that can enhance a general advanced protection system or be considered as a standalone solution.

As an example, Cloudvisor [60] is a transparent, backward-compatible approach protecting the privacy and integrity of cloud VMs. Cloudvisor separates the resource management from security protection in the virtualization layer. A small security monitor hidden under the VMM and using nested virtualization [55] is leveraged to protect the VMM and VMs. This approach is claimed of not affecting the security of users' data inside the VMs.

In NestCloud [44] nested virtualization can be used in several usage models such as debugging and live migration. NestCloud is a three-level nested virtualization architecture minimizing the overhead caused by the additional level.

NestCloud is a very effective approach for detailed introspection of VMs at the cost of increased latency and reduced performance.

Albeit not directly applied to cloud computing, Payer and Gross [45] presented an interesting work on virtualization for safe execution of applications based on software-based fault isolation and policy-based system call authorization. A running application is encapsulated in an additional layer of protection using dynamic binary translation in user-space. This virtualization layer dynamically recompiles the machine code and adds multiple dynamic security guards that verify the running code to protect and contain the application. The binary translation system implemented in [45] redirects all system calls to a policy-based system call authorization framework. This interposition framework validates every system call based on the given arguments and the location of the system call. Depending on the user-loadable policy and an extensible handler mechanism the framework decides whether a system call is allowed, rejected, or redirect to a specific user-space handler in the virtualization layer.

Also Lee et al. [31] discuss how new hardware architectural features for cloud servers can help protect the confidentiality and integrity of a cloud customer's code and data in leased Virtual Machines, even when the powerful underlying hypervisor may be compromised. They use a non-bypassable form of hardware access control leveraging the hardware trend towards manycore chips and hardware virtualization features to enhance Cloud Security. They aim at exploring software-hardware co-design for security to design future trustworthy systems that provide security protections, at the levels needed, when needed, even when malware is in the system.

Another interesting work is by Cazalas et al. [8]. They study whether integrity of execution can be preserved for process-level virtualization protection schemes in the face of adversarial analysis. Their approach considers exploits that target the virtual execution environment itself and how it interacts with the underlying host operating system and hardware. Results indicate that such protection mechanisms may be vulnerable at the level where the virtualized code interacts with the underlying operating system, undermining security and calling for additional mitigation techniques using hardware-based integration or hybrid virtualization techniques that can better defend legitimate uses of virtualized software protection.

5 Secure Enclaves and Virtualization

In Cloud computing environments, hardware resources are shared, and parallel computation widespread that can produce privacy and security issues when isolation is not enforced. In fact, the hypervisor is an important cornerstone of Cloud computing that is not necessarily trustworthy or bug-free. To mitigate this threat Intel and AMD introduced respectively SGX[3] [9] and SEV[4] [29], which transparently encrypt a virtual machines memory. Intel introduced the

[3] Software Guard Extensions.

[4] Secure Encrypted Virtualization.

SGX [11] hardware extensions to create a trusted execution environment (secure enclave or isolation container) within its CPUs. SGX claims runtime protection of a running process/VM even if the host OS and software components are malicious. Isolation containers are a primitive to minimize trusted software, leveraging trusted hardware and having a small performance overhead [11]. This is a smart idea though present implementations (AMD SEV and Intel SGX) do still have some limitations, as we detail in the following.

5.1 Intel SGX

Intel SGX [54] is an hardware technology aimed at protecting guest code and data from the hypervisor. It is an architecture extension designed to increase the security of software through an "inverse sandbox" mechanism. Legitimate software can be sealed inside an "enclave" and protected from unauthorized access, even when malware has hypervisor privileges. SGX was designed to comply with some clear requirements/objectives [9]:

- **protecting sensitive data** from unauthorized access or modification by rogue software running at higher privilege levels;
- **supporting legitimate software** allowing them to continue using platform resources;
- **maintaining consumer freedom** allowing them to retain control of their platforms and the freedom to install and uninstall applications and services as they choose;
- **allow certifying** an application's trusted code and produce a signed attestation, rooted in the processor, that includes this measurement and other certification that the code has been correctly initialized in a trustable environment;
- **supporting legacy** (development) tools, processes, and software distribution channels;
- **allowing scalability** of the performance of trusted applications in order to scale with the capabilities of the underlying hardware;
- **protecting applications** allowing them to define secure regions of code and data that maintain confidentiality even when an attacker has physical control of the platform and can conduct direct attacks on memory.

SGX minimizes the amount of code that provides support for the protected-module architecture, whereas module state persistence is delegated to the untrusted operating system. Nevertheless, state continuity must be guaranteed since an attacker should not be able to cause a module to use stale states (i.e. a rollback attack), and while the system is not under attack, a module should always be able to make progress, even when the system could crash or lose power at unexpected random points in time [54]. Providing state-continuity support is non-trivial as many algorithms are vulnerable to attack, require on-chip non-volatile memory, wear-out existing off-chip secure non-volatile memory and/or

are too slow for many applications. ICE [54] is an interesting architecture providing state-continuity guarantees to protected modules by means of a machine-checked proof. ICE does not rely on secure non-volatile storage for every state update (e.g., the slow TPM chip) and is resilient to power losses.

5.2 SGX Security Issues

Albeit beneficial and promising in theory, the SGX approach has proven vulnerable to (mostly side-channel) attacks from its early days. As an example, CacheZoom [43] can track all memory accesses of SGX enclaves with high spatial and temporal precision. AES key recovery attacks have been proven possible.

Hertzelt et al. [23] analyse to what extent the proposed features can resist a malicious hypervisor and discuss the tradeoffs imposed by additional protection mechanisms. They developed a model of SEV's security capabilities and found three design shortcomings. First the virtual machine control block is not encrypted and handled directly by the hypervisor, allowing it to bypass VM memory encryption by executing conveniently chosen gadgets. Secondly, the general purpose registers are not encrypted upon vmexit, leaking potentially sensitive data. Finally, the control over the nested pagetables allows a malicious hypervisor to closely monitor the execution state of a VM and attack it with memory replay attacks.

Schwarz et al. [51] have found SGX can be used to Conceal Cache Attacks. They demonstrate software-based side-channel attacks from a malicious SGX enclave targeting co-located enclaves, and abusing SGX protection features to conceal itself. The attack is fully functional even across multiple Docker containers. In fact the real issue with cache attacks lies with stealing information (such as private keys) rather that controlling a system.

Cloak [21] is another technique leveraging hardware transactional memory to prevent adversarial observation of cache misses on sensitive code and data. Cloak provides protection against cache-based side-channel attacks for SGX enclaves.

Constan's Sanctum [12] achieves stronger security guarantees under software attacks than SGX with an -h equivalent programming model. In fact, Sanctum offers the same promise as Intel's Software Guard Extensions (SGX), namely strong provable isolation of software modules running concurrently and sharing resources, but protects against an important class of additional software attacks that infer private information from a program's memory access patterns. Sanctum reduces attack surface through isolation, rather than plugging attack-specific privacy leaks. Most of Sanctum's logic is implemented in trusted software, which does not perform cryptographic operations using keys, and is easier to analyze than SGX's opaque microcode. Sanctum prototype leverages a RISC-V [57] core but is quite flexible in that it adds hardware at the interfaces between generic building blocks, replacing SGX's microcode with a software security monitor that runs at a higher privilege level than the hypervisor and the OS. On RISC-V, the security monitor runs at machine level, leveraging one privileged enclave, similarly to SGX's Quoting Enclave. The really interesting

idea behind Sanctum is that it leverages a principled, transparent, and well-scrutinized approach to secure system design.

Various recent research efforts are actively seeking countermeasures to SGX side-channel attacks. It is widely assumed that SGX may be vulnerable to other side channels, such as cache access pattern monitoring, as well. However, prior to our work, the practicality and the extent of such information leakage was not studied. [5] show that cache-based attacks are indeed a serious threat to the confidentiality of SGX-protected programs. They mount our attack without interrupting enclave execution. This approach has major technical challenges, since the existing cache monitoring techniques experience significant noise if the victim process is not interrupted.

The SGX-based branch shadowing attack is described in [32] which can reveal fine-grained control flows (i.e., each branch) of an enclave program running on real SGX hardware. In fact, SGX does not clear the branch history when switching from enclave mode to non-enclave mode, leaving the fine-grained traces to the outside world through a branch-prediction side channel. They developed two exploitation techniques: Intel PT- and LBR-based history-inferring techniques and APIC-based technique to control the execution of enclave programs in a fine-grained manner. As a result, their attack could brake ORAM, Sanctum, SGX-Shield, and T-SGX. A software-based countermeasure, called Zigzagger, was introduced by [32] to mitigate the branch shadowing attack in practice.

Brasser et al. [4] propose a data location randomization as a novel defensive approach against side-channel attacks. Their compiler-based tool called DR.SGX instruments enclave code to permute data locations at the granularity of cache lines. Brasser's solution protects most, but not all enclaves from typical SGX cache attacks.

6 Use Cases for Virtualization

This section introduces increasingly common Use Cases and Technological scenarios. One relevant topic is mobile virtualization for small devices such as smartphones, smart watches, and tablets, that are carried everywhere. They are referred to as Bring Your Own Device (BYOD) since their owner usually carries them even inside the secure perimeter of companies, and in general at work. This section also highlights the usage of virtualization honeypots for malware collection and computer forensics purposes. In fact, malware can be analyzed and dissected based on the interaction with the emulated virtual environment.

6.1 BYOD and Virtualization

Personal mobile devices often enter enterprise boundaries. They can potentially hide malware or eavesdrop sensitive data to the outside world. At present, there is little or no control over an enterprise personnel mobile device data and application content and integrity. Banning such devices altogether from within enterprise boundaries does not seem a viable approach. A better one would imply

remote attestation of the integrity and compliance of the employee's mobile device to the desired security policies. Secure virtualization mechanisms based on a trusted transparent monitoring hypervisor would help. In fact, software integrity attestation future perspectives are good, given that ARM CPUs increasingly support virtualization extensions that allow implementing hypervisors that can run and monitor trusted VMs even on mobile/handheld devices [13]. The hypervisor would be able to enforce the exclusive execution of an enterprise VM when the device is inside well defined boundaries. The same VM can be disabled outside such boundaries in order to limit/prevent data breaches.

6.2 Virtualization and Smartphones

Increasingly often, smart mobile phones are relevant sources of information for investigations. Most currently available tools able to acquire forensic evidence from smartphones require destructive physical access to the device. This is one use case where secure virtualization can be used to access live data without interfering with regular phone activity and thus allowing live mobile forensics. LiveSD Forensics [6] is an example of on-device live data acquisition of the RAM and the EEPROM of Windows mobile devices. LiveSD Forensics uses a standard SD-card equipped with tailored code to perform the data acquisition. Unfortunately, LiveSD generates a memory alteration, albeit small.

In addition, virtualization allows creating mobile honeypots able to study and classify malware in a controlled way. In fact, similarly to mobile forensics, mobile virtualization can be used to collect malware and study its behavior, in a mostly transparent way. As mobile hardware is increasingly capable of running multiple VMs in parallel, different levels of security can be associated to different VMs to limit malware activity.

6.3 Future Research Directions

Future virtualization trends are mostly related to novel technological developments that aim at better isolation and performance. One such example is represented by ARM CPUs that, apart from being dominant in the mobile market, are increasingly present in the server arena. A second example is represented by Cloud-provided GPU access that is increasingly common. Finally, novel x86_64 processors integrate both CPU and GPU cores. Nevertheless, they have to provide additional security guarantees. Efficiently virtualizing distributed heterogeneous computing in the Cloud is an opportunity to improve Cloud security and reliability. Further, in order to allow efficient secure usage of multicores, such resources have to be constantly monitored for anomalous usage patterns, since sharing resources also introduces additional security and privacy issues. Finally, the availability of an increasingly large amount of computing cores allows using them for a number of novel applications, such as computation replication for reliability and availability or proactive computing for most different possible scenarios.

7 Conclusion

Virtualization is at the heart of Cloud computing. Albeit more lightweight approaches such as Containerization and Unikernels exist, hardware-supported isolation mechanisms provide beneficial in many different scenarios where security requirements are relevant. Nevertheless, security vulnerabilities are still a major issue, as highlighted by recently discovered exploits. Enhanced virtualization approaches and more effective isolation and monitoring technologies, that can also leverage additional computing resources of recent CPUs and GPUs, are still in their infancy. Such advances, coupled with appropriate software counterparts, will possibly improve the integrity and security of resources in Cloud, server farms, and in mobile scenarios.

Acknowledgements. Roberto Di Pietro would like to thank Sushil Jajodia for the guidance and support received when he was a young PhD student visiting his Center for Secure Information Systems at GMU—a pivotal experience in Roberto's professional life—and, above all, for Sushil's life-long example of dedication and commitment to pursue research excellence.

References

1. AMD: Secure virtual machine architecture reference manual. http://www.0x04.net/doc/amd/33047.pdf. Accessed 02 Feb 2018 (2005)
2. Baiardi, F., Maggiari, D., Sgandurra, D., Tamberi, F.: Transparent process monitoring in a virtual environment. Electr. Notes Theor. Comput. Sci. **236**, 85–100 (2009). https://doi.org/10.1016/j.entcs.2009.03.016
3. Bijon, K., Krishnan, R., Sandhu, R.: Mitigating multi-tenancy risks in IaaS cloud through constraints-driven virtual resource scheduling. In: Proceedings of the 20th ACM Symposium on Access Control Models and Technologies, SACMAT 2015, pp. 63–74. ACM, New York (2015)
4. Brasser, F., et al.: DR.SGX: hardening SGX enclaves against cache attacks with data location randomization. CoRR abs/1709.09917 (2017)
5. Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., Sadeghi, A.: Software grand exposure: SGX cache attacks are practical. CoRR abs/1702.07521 (2017)
6. Canlar, E.S., Conti, M., Crispo, B., Di Pietro, R.: Windows mobile livesd forensics. J. Netw. Comput. Appl. **36**(2), 677–684 (2013)
7. Carbone, M., Conover, M., Montague, B., Lee, W.: Secure and robust monitoring of virtual machines through guest-assisted introspection. In: Balzarotti, D., Stolfo, S.J., Cova, M. (eds.) RAID 2012. LNCS, vol. 7462, pp. 22–41. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33338-5_2
8. Cazalas, J., McDonald, J.T., Andel, T.R., Stakhanova, N.: Probing the limits of virtualized software protection. In: Proceedings of the 4th Program Protection and Reverse Engineering Workshop. PPREW-4, pp. 5:1–5:11. ACM, New York (2014)
9. Chakrabarti, S., et al.: Intel software guard extensions (Intel; SGX) architecture for oversubscription of secure memory in a virtualized environment. In: Proceedings Hardware and Architectural Support for Security and Privacy. HASP 2017, pp. 7:1–7:8. ACM, New York (2017)

10. Combe, T., Martin, A., Di Pietro, R.: To docker or not to docker: a security perspective. IEEE Cloud Comput. **3**(5), 54–62 (2016)
11. Costan, V., Lebedev, I., Devadas, S.: Secure processors part I: background, taxonomy for secure enclaves and intel SGX architecture. Found. Trends® Electron. Des. Autom. **11**(1–2), 1–248 (2017)
12. Costan, V., Lebedev, I.A., Devadas, S.: Sanctum: minimal hardware extensions for strong software isolation. In: USENIX Security Symposium, pp. 857–874 (2016)
13. Dall, C., Nieh, J.: KVM/ARM: the design and implementation of the Linux arm hypervisor. SIGARCH Comput. Archit. News **42**(1), 333–348 (2014)
14. Di Pietro, R., Franzoni, F., Lombardi, F.: HyBIS: advanced introspection for effective windows guest protection. In: De Capitani di Vimercati, S., Martinelli, F. (eds.) SEC 2017. IAICT, vol. 502, pp. 189–204. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58469-0_13
15. Di Pietro, R., Lombardi, F., Signorini, M.: CloRExPa: cloud resilience via execution path analysis. Future Gener. Comput. Syst. **32**, 168–179 (2014)
16. Di Pietro, R., Lombardi, F., Villani, A.: CUDA leaks: a detailed hack for CUDA and a (partial) fix. ACM Trans. Embed. Comput. Syst. **15**(1), 15:1–15:25 (2016)
17. Dowty, M., Sugerman, J.: GPU virtualization on VMware's hosted I/O architecture. SIGOPS Oper. Syst. Rev. **43**(3), 73–82 (2009)
18. Dua, R., Raja, A.R., Kakadia, D.: Virtualization vs containerization to support PaaS. In: 2014 IEEE International Conference on Cloud Engineering, pp. 610–614, March 2014
19. By Hertzsprung at English Wikipedia, C.B.S.: Execution rings. https://commons.wikimedia.org/w/index.php?curid=8950144
20. Fernandes, D.A.B., Soares, L.F.B., Freire, M.M., Inácio, P.R.M.: Randomness in virtual machines. In: 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing, pp. 282–286, December 2013
21. Gruss, D., Lettner, J., Schuster, F., Ohrimenko, O., Haller, I., Costa, M.: Strong and efficient cache side-channel protection using hardware transactional memory. In: 26th USENIX Security Symposium (USENIX Security 17), pp. 217–233. USENIX Association, Vancouver, BC (2017)
22. Gupta, V., et al.: GViM: GPU-accelerated virtual machines. In: Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing. HPCVirt 2009, pp. 17–24. ACM, New York (2009)
23. Hetzelt, F., Buhren, R.: Security analysis of encrypted virtual machines. SIGPLAN Not. **52**(7), 129–142 (2017)
24. Hong, C.H., Spence, I., Nikolopoulos, D.S.: GPU virtualization and scheduling methods: a comprehensive survey. ACM Comput. Surv. **50**(3), 35:1–35:37 (2017)
25. Intel: Intel virtualization technology specification for the ia-32 intel architecture (2005). http://dforeman.cs.binghamton.edu/~foreman/550pages/Readings/intel05virtualization.pdf. Accessed 02 Feb 2018
26. Geffner, J.: VENOM: Virtualized Environment Neglected Operations Manipulation. Available from MITRE, CVE-ID CVE-2015-3456, May 2015
27. Jia, L., Zhu, M., Tu, B.: T-VMI: trusted virtual machine introspection in cloud environments. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. CCGrid 2017, pp. 478–487. IEEE Press, Piscataway, NJ, USA (2017)
28. Jian, Z., Chen, L.: A defense method against Docker escape attack. In: Proceedings of the 2017 International Conference on Cryptography, Security and Privacy. ICCSP 2017, pp. 142–146. ACM, New York (2017)

29. Kaplan, D., Powell, J., Woller, T.: AMD memory encryption. White paper (2016). https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf
30. Kocher, P., et al.: Spectre attacks: Exploiting speculative execution. ArXiv e-prints 1801.01203, January 2018
31. Lee, R.B.: Hardware-enhanced access control for cloud computing. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies. SACMAT 2012, pp. 1–2. ACM, New York (2012)
32. Lee, S., Shih, M., Gera, P., Kim, T., Kim, H., Peinado, M.: Inferring fine-grained control flow inside SGX enclaves with branch shadowing. CoRR abs/1611.06952 (2016)
33. Lengyel, T.K.: Malware collection and analysis via hardware virtualization. Doctoral dissertations, 964 (2015). https://opencommons.uconn.edu/dissertations/964
34. Lipp, M., et al.: Meltdown. ArXiv e-prints 1801.01207 (2018)
35. Lombardi, F., Di Pietro, R.: Secure virtualization for cloud computing. J. Netw. Comput. Appl. **34**(4), 1113–1122 (2011)
36. Lombardi, F., Pietro, R.D., Soriente, C.: Crew: cloud resilience for windows guests through monitored virtualization. In: Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems. SRDS 2010, pp. 338–342. IEEE Computer Society, Washington, DC, USA (2010)
37. Joy, A.M.: Performance comparison between Linux containers and virtual machines. In: International Conference on Advances in Computer Engineering and Applications, pp. 342–346, March 2015
38. Madhavapeddy, A., et al.: Unikernels: library operating systems for the cloud. SIGPLAN Not. **48**(4), 461–472 (2013)
39. Manu, A.R., Patel, J.K., Akhtar, S., Agrawal, V.K., Murthy, K.N.B.S.: A study, analysis and deep dive on cloud PAAS security in terms of Docker container security. In: 2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT), pp. 1–13, March 2016
40. Martin, A., Raponi, S., Combe, T., Di Pietro, R.: Docker ecosystem - vulnerability analysis. Comput. Commun. **122**, 30–43 (2018)
41. Maurice, C., Neumann, C., Heen, O., Francillon, A.: Confidentiality issues on a GPU in a virtualized environment. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 119–135. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_9
42. Merkel, D.: Docker: lightweight Linux containers for consistent development and deployment. Linux J. **2014**(239) (2014). Article no. 2. http://dl.acm.org/citation.cfm?id=2600239.2600241
43. Moghimi, A., Irazoqui, G., Eisenbarth, T.: CacheZoom: how SGX amplifies the power of cache attacks. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 69–90. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_4
44. Pan, Z., He, Q., Jiang, W., Chen, Y., Dong, Y.: Nestcloud: towards practical nested virtualization. In: Proceedings of the 2011 International Conference on Cloud and Service Computing. CSC 2011, pp. 321–329. IEEE Computer Society, Washington, DC, USA (2011)
45. Payer, M., Gross, T.R.: Fine-grained user-space security through virtualization. SIGPLAN Not. **46**(7), 157–168 (2011)
46. Perez-Botero, D., Szefer, J., Lee, R.B.: Characterizing hypervisor vulnerabilities in cloud computing servers. In: Proceedings of the 2013 International Workshop on

Security in Cloud Computing. Cloud Computing 2013, pp. 3–10. ACM, New York (2013)
47. Ray, E., Schultz, E.: Virtualization security. In: Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies. CSIIRW 2009, pp. 42:1–42:5. ACM (2009)
48. 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. CCS 2009, pp. 199–212. ACM, New York (2009)
49. Ristenpart, T., Yilek, S.: When good randomness goes bad: virtual machine reset vulnerabilities and hedging deployed cryptography. In: NDSS, pp. 212–224 (2010)
50. Sabahi, F.: Cloud computing security threats and responses. In: 2011 IEEE 3rd International Conference on Communication Software and Networks, pp. 245–249, May 2011
51. Schwarz, M., Weiser, S., Gruss, D., Maurice, C., Mangard, S.: Malware guard extension: using SGX to conceal cache attacks. CoRR abs/1702.08719 (2017)
52. Sfyrakis, I., Großß, T.: Virtuscap: capability-based access control for unikernels. In: 2017 IEEE International Conference on Cloud Engineering (IC2E), pp. 226–237. IEEE (2017)
53. Shi, L., Chen, H., Sun, J.: vCUDA: GPU accelerated high performance computing in virtual machines. In: IEEE International Symposium on Parallel Distributed Processing, pp. 1–11, May 2009
54. Strackx, R., Jacobs, B., Piessens, F.: ICE: a passive, high-speed, state-continuity scheme. In: Proceedings of the 30th Annual Computer Security Applications Conference. ACSAC 2014, pp. 106–115. ACM, New York (2014)
55. Suzaki, K., Yagi, T., Tanaka, A., Oiwa, Y., Shibayama, E.: Rollback mechanism of nested virtual machines for protocol fuzz testing. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing. SAC 2014, pp. 1484–1491. ACM, New York (2014)
56. Tian, K., Dong, Y., Cowperthwaite, D.: A full GPU virtualization solution with mediated pass-through. In: 2014 USENIX Annual Technical Conference (USENIX ATC 14), pp. 121–132. USENIX Association, Philadelphia, PA (2014)
57. Waterman, A., Asanovic, K.: The RISC-V instruction set manual. https://riscv.org/specifications. Accessed 02 Feb 2018
58. Wong, T.: AMD multiuser GPU (2016). https://www.amd.com/Documents/Multiuser-GPU-White-Paper.pdf
59. Xiao, S., et al.: VOCL: an optimized environment for transparent virtualization of graphics processing units. In: Innovative Parallel Computing, pp. 1–12, May 2012
60. 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. SOSP 2011, pp. 203–216. ACM, New York (2011)
61. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-tenant side-channel attacks in paas clouds. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. CCS 2014, pp. 990–1003. ACM, New York (2014)