# Evolution of Attacks, Threat Models, and Solutions for Virtualized Systems

DANIELE SGANDURRA and EMIL LUPU, Imperial College London

Virtualization technology enables Cloud providers to efficiently use their computing services and resources. Even if the benefits in terms of performance, maintenance, and cost are evident, however, virtualization has also been exploited by attackers to devise new ways to compromise a system. To address these problems, research security solutions have evolved considerably over the years to cope with new attacks and threat models. In this work, we review the protection strategies proposed in the literature and show how some of the solutions have been invalidated by new attacks, or threat models, that were previously not considered. The goal is to show the evolution of the threats, and of the related security and trust assumptions, in virtualized systems that have given rise to complex threat models and the corresponding sophistication of protection strategies to deal with such attacks. We also categorize threat models, security and trust assumptions, and attacks against a virtualized system at the different layers—in particular, hardware, virtualization, OS, and application.



Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms: Security, Algorithms, Measurement

Additional Key Words and Phrases: Virtualization, threat models, Cloud computing, integrity attacks

**ACM Reference Format:**
Daniele Sgandurra and Emil Lupu. 2016. Evolution of attacks, threat models, and solutions for virtualized systems. ACM Comput. Surv. 48, 3, Article 46 (February 2016), 38 pages.
DOI: http://dx.doi.org/10.1145/2856126

## 1. INTRODUCTION

Virtualization increases the efficient use of computing services and resources in terms of their performance, maintenance, and cost by enabling multiple environments, such as operating systems (OSes), to share the same physical resources. While its advantages are well documented [Creasy 1981; Uhlig et al. 2005a; Rosenblum 2004], virtualization also gives rise to several security concerns [Sahoo et al. 2010; Garfinkel and Rosenblum 2005; Pearce et al. 2013]. Some of these concerns are not entirely novel, whereas others, such as multi-tenancy or high privileges of the hypervisor, are specific to virtualization and require novel solutions [Azab et al. 2010; Szefer et al. 2011; Zhang et al. 2011a; Butt et al. 2012; Xia et al. 2013; Wu et al. 2013; Hofmann et al. 2013].

Addressing security concerns in a virtualized system requires a careful consideration of the threats; however, when the virtualized environment is "outsourced to the Cloud,"

---

This work is supported by FP7 EU-funded project Coco Cloud under grant no. 610853 and EPSRC Project CIPART, grant no. EP/L022729/1.
Authors' addresses: D. Sgandurra and E. C. Lupu, Computer Science Department, Imperial College London, Department of Computing, Huxley Building 180 Queen's Gate, South Kensington Campus, London SW7 2AZ, UK; emails: {d.sgandurra, e.c.lupu}@imperial.ac.uk.

DOI: http://dx.doi.org/10.1145/2856126

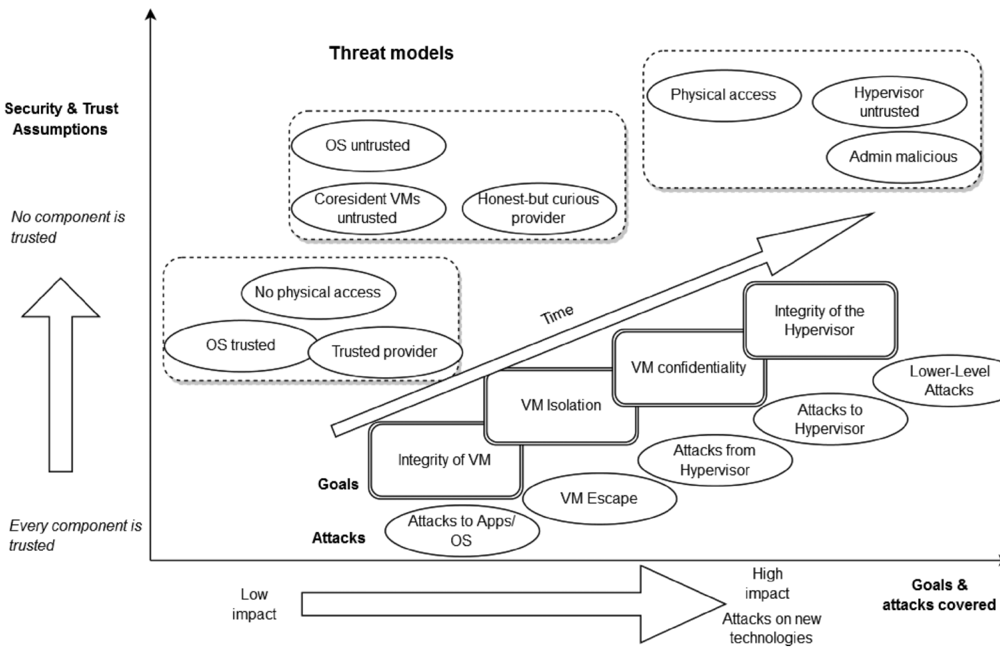


Fig. 1. Evolution of threat models and corresponding solutions over time.

that is, run by an external provider, the trust placed in that provider (*trust assumptions*) also needs to be examined [Butt et al. 2012; Szefer et al. 2011; Li et al. 2013; Santos et al. 2012]. It is important to realize that such trust assumptions, as well as other *security assumptions*, are not fixed, but change in time owing to changes in the technology, the discovery of new vulnerabilities, and so on. Threat models for virtualized systems have evolved considerably over the years; therefore, research solutions have also become increasingly sophisticated to cope with them[Garfinkel and Rosenblum 2003; Payne et al. 2007; Jiang et al. 2007]. For example, early studies assumed the hypervisor of a remote host to be trusted, and used this assumption to build a trusted chain of enforcement [Keller et al. 2010; Hofmann et al. 2011; Azab et al. 2009; Xiong et al. 2011]. More recent studies consider this assumption to be too strong [Wu et al. 2013; Xia et al. 2013; Ding et al. 2013a], especially in light of new attacks against the hypervisor that have since been discovered [Wang and Jiang 2010; Ding et al. 2013a] and the correspondingly updated threat models.

Figure 1 shows the evolution of the threat models and of the security goals as a function of the attacks discovered and the threat assumptions. We can see that, over the years, threat models have evolved from considering only attacks on the tenant OS, such as control-flow attacks [Abadi et al. 2005; Petroni and Hicks 2007], to lower-level attacks on the hypervisor, such as virtualization-based rootkits [King and Chen 2006; Dai Zovi 2006; Wojtczuk 2008] or cross virtual machines attacks [Kortchinsky 2009; Xu et al. 2011]. As a consequence, the trust assumptions have also changed by reducing the size of the trusted computing base (TCB). To address new threats, with weaker assumptions, research solutions have also evolved, by first proposing protection models that assume a trusted monitoring application, then a trusted kernel, to solutions adding the enforcement into an administrative virtual machine. They have further evolved by proposing solutions in which only the hypervisor is trusted to, finally, solutions that protect the (untrusted) hypervisor from lower levels. In parallel, technologies for root-of-trust have evolved to cope with these new protection needs.

This survey arises from the observation that many publications in this area have become narrower in focus, and often rely on implicit and different assumptions. Threat models are often presented in different ways, making it difficult to evaluate the efficacy of solutions: which threats do they address and under which assumptions? Moreover, to the best of our knowledge, the evolution of threat models, attacks, and research solutions for virtualized systems is not presented in the existing literature. Hence, in this survey, we categorize threat models, security and trust assumptions, and attacks against a virtualized system at the different layers: hardware, hypervisor, virtualization, OS, and application. We review protection strategies proposed in the literature and show why some of the solutions have been invalidated by new attacks or by threat models previously not considered. We focus, in particular, on attacks and security solutions of the *compute* virtualization area, and do not cover all possible attacks against network and storage virtualization, as well as overall virtualization infrastructure security. We also aim to provide a first guide towards presenting in one uniform framework the *threat models* (security and trust assumptions), *security properties* (goals and TCB) and *implementation strategies* (methodology and features) for the proposed solutions in the hope that future papers can use a standard approach to define their assumptions and goals. Other recent surveys cover related topics, thus sometimes refer to the same literature, in particular:

- Ryan [2013] surveys four generic solutions for Cloud security: homomorphic encryption, key translation, hardware-anchored security, and CryptDB. The work also outlines the weaknesses of each approach and discusses their applicability in real scenarios. These solutions are orthogonal to the ones discussed in this article.
- Bouchenak et al. [2013] survey existing studies towards verifying that Cloud-based services behave as expected. Their work focuses on verifying the identity of the service and of the nodes that the service runs on, on verifying the functional correctness of a service with respect to service-level agreement parameters (e.g., performance and dependability) and on the compliance of the service with security requirements as specified by a security policy. In contrast, we focus here on virtualization-based solutions to protect the integrity of software running on the Cloud.
- Xiao and Xiao [2013] survey studies on Cloud security and privacy by clustering problems and solutions in five areas: *Cloud confidentiality*, *Cloud integrity*, *Cloud availability*, *Cloud accountability*, and *Cloud privacy*. In this survey, we also consider the evolution of threat models, the taxonomy of attacks, and protection solutions, with more emphasis on integrity and confidentiality rather than availability and accountability.
- Pearce et al. [2013] focus on threats and solutions in virtualized scenarios. Among the related work, this article is probably closest to ours. However, in contrast with this work, we provide a taxonomy of attacks and solutions by considering the level at which they apply; furthermore, we discuss the threat models and their evolution, as well as the different TCB presented in the papers. Finally, we consider attestation techniques for virtualized environments and provide a first guide for a uniform framework for presenting threat models and protection solutions.
- Pék et al. [2013] focus on low-level attacks on the virtualization layers and other issues introduced by such layers. We also consider higher-level issues and discuss in detail existing solutions to cope with these attacks.

This article is organized as follows. In Section 2, we introduce some key concepts of virtualization and Cloud computing. In Section 3, we present a taxonomy of attacks that exploit virtualization. In Section 4, we survey and categorize research solutions aimed at protecting virtualized environments. In Section 5, we discuss the threat models on virtualization resources at different levels, that is, hardware, virtualization and the Cloud. We also propose a methodology to define, categorize and evaluate the security solutions under a common framework. In Section 6, we analyze the results of this survey by showing the trend of the protection solutions. In particular, we show that solutions usually consider only a specific threat model; for this reason, they are highly sensitive to variations to that model, for example, if an assumption is removed, or a new threat is considered, the solution is no longer valid. In Section 7, we present our conclusions.

## 2. BASICS OF VIRTUALIZATION

We start by recalling some of the key concepts of virtualization. Virtualization enables the software emulation of the physical properties of a physical computer, which is encapsulated in a *Virtual Machine (VM)* that can be used and managed independently from other VMs. This allows the resources of a physical computer, including processor, memory, storage, and I/O channels, to be shared between several concurrent VMs, while preserving isolation. In turn, this allows for a more efficient use of the physical resources and the *on-demand* allocation of resources to the tenants. For example, in current virtualized environments, a new VM can be created in a matter of seconds. Usually, an administrative VM (*Admin VM*[1]) takes care of configuring and initializing the VMs. The replacement of a physical system with a VM is, in theory, transparent to the applications and to the OS, which can run unchanged. Therefore, the tenants' applications can run in a VM exactly as they do on a physical architecture. A *virtual machine monitor (VMM)* [Goldberg 1973, 1974] is the software component that creates, manages, and monitors the VM, and can be of two types: (i) a type I VMM[2], which is a thin software layer that runs on top of the hardware/firmware layer; and (ii) a type II VMM, which runs on top of a *host OS* and implements each tenant VM as a *guest OS* process that emulates a physical architecture. VMs allow distinct OSes and applications in them to run concurrently. From a security point of view, the VMM must guarantee the isolation of the VMs to ensure that any erroneous or malicious behavior within a VM is confined within it. This isolation is fundamental: it must be ensured even for processes running with administrative permissions within a VM, and requires mediating each access from a VM to shared resources.

### 2.1. Cloud Computing

Virtualization is the key foundation for realizing *Cloud computing*—a form of computing similar in several respects to *utility computing* [Rappa 2004]—in which services, including storage, computation and applications, are provided by large pools of remote resources. We recall the major service models of Cloud computing, which are known as [Mell and Grance 2011]: (i) *software-as-a-service* (SaaS), in which tenants are provided with the capability to use the provider's applications but do not manage or control the infrastructure; (ii) *platform-as-a-service* (PaaS), in which tenants can deploy onto the Cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider; in PaaS, the consumer has control only over the deployed applications and possibly their hosting configurations; and (iii) *infrastructure-as-a-service* (IaaS), in which tenants can access virtualized physical resources such as processing, storage, and networks, in which they can deploy and

[1]The VM used by administrators to manage the platform that has privileged access to the VMM interfaces and the VMs, for example, to create them, stop them, or use the introspection interface. An example is Dom0 in Xen. In literature, it is also referred to as *Privileged VM*, *Control VM*, *Root VM*, and so on.
[2]A type I VMM is also called a *hypervisor*: we will use the term *hypervisor* interchangeably with any type of VMM.

run arbitrary software. Tenants manage the use of the OS and the resources they have purchased. These Cloud services may be offered in four different deployment models:

—*Private Cloud*: operated solely for an organization and managed by it (although the management may be subcontracted);
—*Community Cloud*: shared by several organizations and supports a specific community that has shared concerns. It may be managed by the organizations or a contracted third party;
—*Public Cloud*: available to the general public or a large industry group and is owned by an organization selling Cloud services;
—*Hybrid Cloud*: a composition of two or more Clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability.

There is clearly a trade-off for tenants between flexibility and control (and security and trust assumptions) among these service and deployment models. In fact, tenants have more control (and better security) and flexibility (e.g., in terms of configuration options) when moving from SaaS to IaaS and, analogously, when moving from a public Cloud to a private Cloud. The drawback in these cases is that, when moving from SaaS to IaaS, the user interface to the Cloud, that is, to allocate services or resources, places an additional burden to the user in terms of ease of use.

## 3. ATTACKS IN VIRTUALIZED ENVIRONMENTS

While virtualization offers many advantages, as discussed in the previous section, it also introduces new security concerns linked mainly to the loss of control arising from the usage of third-party owned resources, the sharing of resources among VMs, and vulnerabilities of the hypervisor itself. In particular, attacks due to vulnerabilities in the hypervisor itself have a high impact, since they allow an attacker to control all the VMs. We therefore start by first describing the categories of attacks possible in a virtualized environment as presented in the literature. First, attacks can be categorized by considering their possible *targets*, that is, at what level an attacker can exploit a vulnerability. The taxonomy that we present here spans from the higher to the lowest levels:

—*Application-level* (guest VM's user-space): these are attacks against user applications, such as through injection of malicious code inside an application to divert its control flow and execute the attacker's code [Aleph One 1996; Roemer et al. 2012];
—*Kernel-level* (guest VM's kernel space): these attacks target the OS, such as kernel rootkits [Sparks and Butler 2005; Hoglund and Butler 2006; Levine et al. 2006], which allow an attacker to fully control the system;
—*Virtualization layer*: these attacks exploit the virtualization features in many ways, such as to attack VMs residing on the same host [Xu et al. 2011; Zhang et al. 2012];
—*Hypervisor*: these attacks try to exploit vulnerabilities at the hypervisor level (see also Table I) to gain control of it and of all the VMs on top of it; other ways require escaping from a VM to attack the hypervisor [Kortchinsky 2009];
—*lower levels*: in these attacks, an attacker tries to subvert the levels below the hypervisor, such as the hardware or the System Management Mode (SMM), for example, to directly access the memory to modify/read the hypervisor virtual space [Embleton et al. 2008; Stewin and Bystrov 2013].

Attacks can also be categorized by taking into account the *source* of attacks [Jamkhedkar et al. 2013], that is, from where the attack originated. In particular, they can be initiated by the same VM (e.g., from kernel, other applications, or by the application not respecting its intended behavior), or from the same host (e.g., coresident VMs,

Table I. Vulnerabilities in CVE with High Severity for the Period January 2012 - June 2015 for Xen, VMWare Esxi, Hyper-V and KVM

| Product | CVE ID | Vulnerability Type(s) | Score | Access | Complexity | Conf. | Integ. | Avail. |
|---|---|---|---|---|---|---|---|---|
| Xen | CVE-2015-4104 | DoS | 7.8 | R | Low | None | None | C |
| Xen | CVE-2015-3456 | DoS Exec Code Overflow | 7.7 | L-N | Low | C | C | C |
| Xen | CVE-2015-3209 | Exec Code Overflow | 7.5 | R | Low | P | P | P |
| Xen | CVE-2015-2751 | DoS | 7.1 | R | Medium | None | None | C |
| Xen | CVE-2015-2151 | DoS Exec Code Mem. Corr. +Info | 7.2 | L | Low | C | C | C |
| Xen | CVE-2015-0361 | DoS | 7.8 | R | Low | None | None | C |
| Xen | CVE-2014-9030 | DoS | 7.1 | R | Medium | None | None | C |
| Xen | CVE-2014-7188 | DoS | 8.3 | L-N | Low | C | C | C |
| Xen | CVE-2014-3969 | +Priv | 7.4 | L-N | Medium | C | C | C |
| Xen | CVE-2014-1666 | DoS +Priv | 8.3 | L-N | Low | C | C | C |
| Xen | CVE-2013-6375 | DoS +Priv | 7.9 | L-N | Medium | C | C | C |
| Xen | CVE-2013-2211 | Other | 7.4 | L-N | Medium | C | C | C |
| Xen | CVE-2013-2072 | DoS Overflow +Priv Mem. Corr. | 7.4 | L-N | Medium | C | C | C |
| Xen | CVE-2013-1432 | DoS +Priv | 7.4 | L-N | Medium | C | C | C |
| Xen | CVE-2012-6030 | DoS | 7.2 | L | Low | C | C | C |
| Xen | CVE-2012-3515 | +Priv | 7.2 | L | Low | C | C | C |
| Xen | CVE-2012-0217 | Overflow +Priv | 7.2 | L | Low | C | C | C |
| Xen | CVE-2011-1763 | DoS +Priv | 7.7 | L-N | Low | C | C | C |
| Esxi | CVE-2013-5970 | DoS | 7.1 | R | Medium | None | None | C |
| Esxi | CVE-2013-3658 | Dir. Trav. | 9.4 | R | Low | None | C | C |
| Esxi | CVE-2013-3657 | DoS Exec Code Overflow | 7.5 | R | Low | P | P | P |
| Esxi | CVE-2013-3519 | +Priv | 7.9 | L-N | Medium | C | C | C |
| Esxi | CVE-2013-1659 | DoS Exec Code Mem. Corr. | 7.6 | R | High | C | C | C |
| Esxi | CVE-2013-1406 | +Priv | 7.2 | L | Low | C | C | C |
| Esxi | CVE-2013-1405 | DoS Exec Code Mem. Corr. | 10 | R | Low | C | C | C |
| Esxi | CVE-2012-3289 | DoS | 7.8 | R | Low | None | None | C |
| Esxi | CVE-2012-3288 | DoS Exec Code Mem. Corr. | 9.3 | R | Medium | C | C | C |
| Esxi | CVE-2012-2450 | DoS Exec Code | 9 | R | Low | C | C | C |
| Esxi | CVE-2012-2449 | DoS Exec Code Overflow | 9 | R | Low | C | C | C |
| Esxi | CVE-2012-2448 | DoS Exec Code Overflow | 7.5 | R | Low | P | P | P |
| Esxi | CVE-2012-1518 | +Priv | 8.3 | L-N | Low | C | C | C |
| Esxi | CVE-2012-1517 | DoS Exec Code Overflow | 9 | R | Low | C | C | C |
| Esxi | CVE-2012-1516 | DoS Exec Code Overflow | 9 | R | Low | C | C | C |
| Esxi | CVE-2012-1515 | +Priv | 8.3 | L-N | Low | C | C | C |
| Esxi | CVE-2012-1510 | Overflow +Priv | 7.2 | L | Low | C | C | C |
| Esxi | CVE-2012-1508 | DoS +Priv | 7.2 | L | Low | C | C | C |
| Hyper-V | CVE-2013-3898 | DoS Exec Code Mem. Corr. | 7.9 | L-N | Medium | C | C | C |
| KVM | CVE-2015-3456 | DoS Exec Code Overflow | 7.7 | L-N | Low | C | C | C |
| KVM | CVE-2011-2212 | DoS Overflow +Priv | 7.4 | L-N | Medium | C | C | C |

*Note*: C=Complete, P=Partial, L=Local, L-N=Local Network, R=Remote.

the hypervisor, lower levels), or external hosts. As far as internal or external attack sources are concerned, we consider the following as possible attackers: (i) the Cloud providers, which can be considered as trusted providers, honest-but-curious providers, malicious providers, or trusted with insider threats; (ii) tenants of other VMs; (iii) tenants of the same VM (from the point of view of the provider). Any other possible source of external attacks, such as a server in the same Cloud environment or a remote host, is considered as a generic external attacker. The sources of attacks can be further fine-tuned if we consider entities such as the manufacturer of the hardware used by the provider, the developers of the software run by the provider, and, in general, any third-party involved with the provider. Such issues are covered well in Bleikertz et al. [2013]. Finally, we need to consider the *goal* of the attack in virtualized environments,

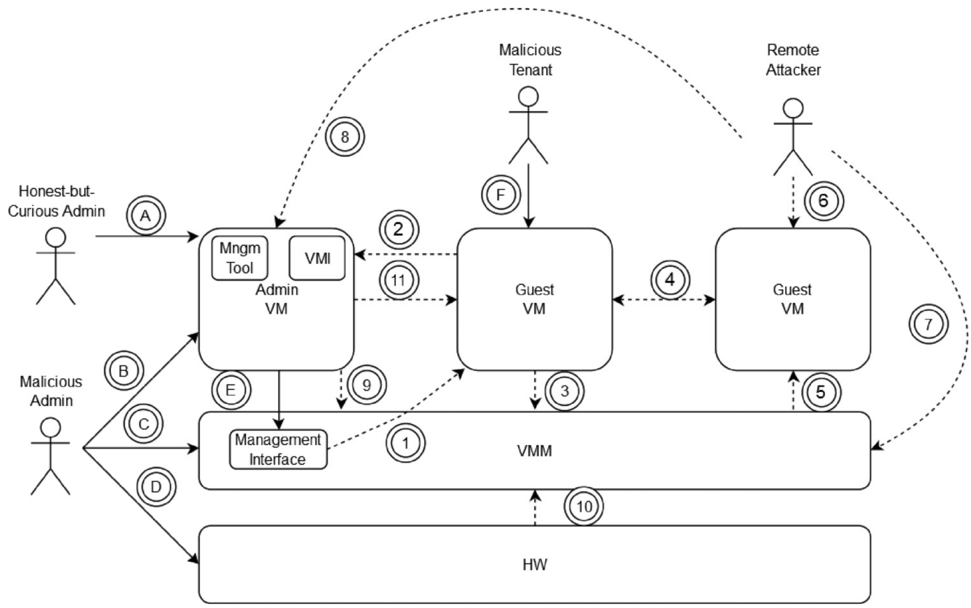


Fig. 2. Attack paths in a VM.

which can be the compromise of the integrity, confidentiality, or availability. In this work, we do not consider attacks against the availability of the services because these are usually regulated by service-level agreements (SLAs) and they are easier to be checked and accounted for.

### 3.1. Attack Paths in Virtualized Systems

Taking into account the previous taxonomy of attacks, we have enumerated in Figure 2 the possible attack paths in virtualized environments. The double-circled letters indicate the initial existing accesses to the assets according to the considered threat model. In particular, in Ⓐ, an honest-but-curious provider has (read-only) access to the Admin VM, whereas in Ⓑ, a malicious administrator has a read-and-write access to the Admin VM. The Admin VM may export a management tool, with virtual machine introspection (VMI) capabilities, that enables direct access to the VMM through its management interface (see path Ⓔ) to modify configuration files, start new VMs or halt existing ones, or read/modify the memory of the VMs. Having physical access to the premises, a malicious administrator can also directly access the VMM (see Ⓒ) and the layers below (see Ⓓ). Instead, a tenant usually has access only to a guest VM (see Ⓕ), which may entitle the tenant to either access the entire stack of the VMs (in the IaaS model) or only some applications (in the SaaS model). Here, we consider the case in which the (malicious) tenant has full access to the VM (as in the IaaS model).

The second set of paths, highlighted with circled numbers, show possible *attack hops*. In particular, in ①, we show the path that enables an entity using the management interface to (maliciously) access the state of a VM (e.g., memory, processor's registers, and so on). As an example, this path may enable an attacker to modify the status of the VM (integrity), read private files (confidentiality), or stop/start VMs (availability). With path ②, an entity owning a guest VM gains access to the Admin VM, either because of

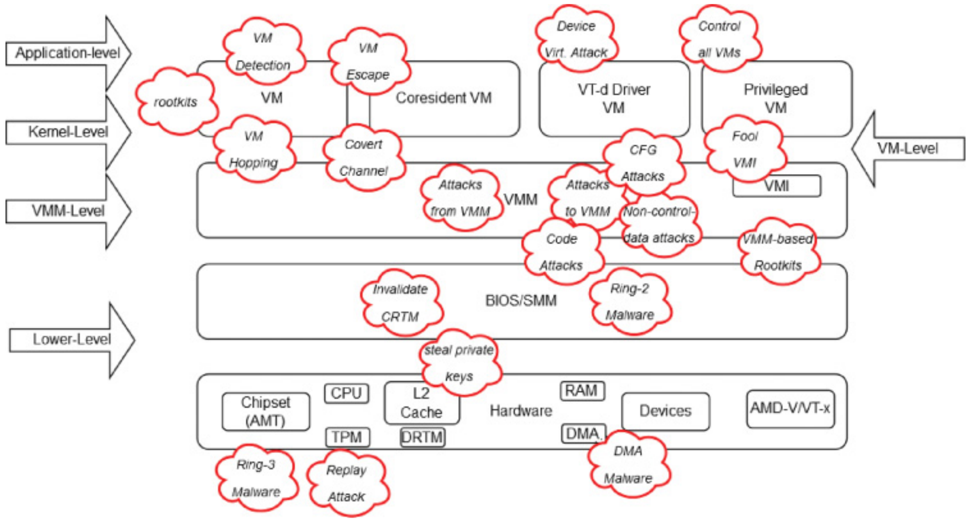


Fig. 3. Summary of some attacks at the various levels.

its vulnerabilities or open ports, which may be open only to the internal network or to the virtual private network of the VMM, or because of Cross-VM attacks (described in Section 3.2.1). With ③, a guest VM exploits a vulnerability of the VMM to gain access to the VMM or to attack it. In ④, a guest VM either gains access to another VM (not belonging to the user of the guest VM) or exploits Cross-VM attacks to steal private data from the co-located VM. The difference with ② is that, in ②, usually the attacker wants to elevate one's privileges, whereas in ④, the target is the VM itself. In ⑤, we show the path where a (compromised) hypervisor is exploited to attack, or access, a guest VM illegally. In ⑥, ⑦, and ⑧, a remote attacker exploits a vulnerability to illegally attack (or access), respectively, a tenant VM, the VMM, or the Admin VM. The Admin VM can also be used to exploit a vulnerability of the VMM to attack it, or gain access to it, as shown in path ⑨. Finally, path ⑩ shows the compromise of the VMM from the layers below.

Note that these paths can be combined, such as with the combination of ⑥ and ③, in which an attacker that wants to subvert an hypervisor first attacks a guest VM (⑥) and then attacks the VMM from the guest VM (③), or analogously with paths ⑧ and ①.

### 3.2. Existing Attacks

In the following, we briefly survey known attacks against a virtualized environment at different levels. In particular, we focus on the last three targets of the previous taxonomy: virtualization layer (*attacks to VMs*), hypervisor (*attacks to and from hypervisor*), and *attacks to lower levels*. Figure 3 illustrates some of the attacks discussed in the following by showing at which level they are deployed.

*3.2.1. Attacks to VMs.* The first layer that we consider is the virtualization one; in particular, we describe known attacks against VMs or attacks that are facilitated by the presence of the virtualization layer. We show that the presence of a VMM does not necessarily make a system more secure. In fact, even if virtualization enables strong separation among VMs on the same host, one of its interfaces, such as hypercalls

[Barham et al. 2003], may be vulnerable and exploited by an attacker. Furthermore, we need to consider whether an attacker can detect if the target runs in a virtualized environment and if that attacker can exploit this knowledge. We will then discuss existing attacks facilitated by the virtualization layer.

*VM Detection*. For an attacker, detecting that a target system runs inside a VM might open new avenues to subvert the system. In fact, by detecting the presence of the virtualization layer, the attacker can focus its targeted attacks on this layer rather than on the OS or applications so that the whole system can be more easily subverted, for example, by trying to exploit known vulnerabilities of the discovered VMM. Quist and Smith [2006] describe a method for determining the presence of VM emulation in a nonprivileged operating environment by using the local descriptor table as a signature for virtualization. Garfinkel et al. [2007] discuss the usage of timing benchmarks to detect the presence of VMMs, whereas Raffetseder et al. [2007] survey alternative strategies to detect system emulators. Ferrie [2006, 2007] offer some possible ways for VMM to hide their presence, such as intercepting nonsensitive instructions (e.g., `SIDT`), whereas Carpenter et al. [2007] present some mitigation techniques against VM detection on VMware by modifying the VM configuration files (VMX files). Franklin et al. [2008a, 2008b] show how it is possible for an attacker to detect the presence of a VMM under the OS by measuring the execution time of particular code sequences on the remote system, whose execution time differs from the perspective of an external verifier when a host runs inside a VM.

*Virtualization Extension-Based Attacks*. This class of attacks exploits existing virtualization technologies, such as AMD-V [Advanced Micro Devices 2005], Intel VT-x [Uhlig et al. 2005b] or Intel Virtualization Technology for Directed I/O (VT-d)[3] by trying to insert a malware (*VM-based rootkits*, or VMBRs) at a lower level than the hypervisor (this techniques is also called "hyperjacking"). This class of malware inserts a malicious hypervisor underneath the OS and leverages virtualization to make itself undetectable. Proof-of-concept VMBRs are *SubVirt* [King and Chen 2006], *Vitriol* [Dai Zovi 2006] and *Blue Pill* [Rafal Wojtczuk 2008]. Wojtczuk and Rutkowska [2011] show some software attacks to escape from a VM with direct access to physical devices to gain full control over the whole system. Further attacks against the VMM that exploit direct device assignment are discussed in Pék et al. [2014].

*VM Escape and VM Hopping*. In a *VM escape* attack, an attacker is able to break out from a compromised VM and take control of the underlying hypervisor. Once there, an attacker can invoke any function, such as for creating VMs, or managing I/O devices. For example, *Cloudburst* [Kortchinsky 2009] is a memory-corruption exploit that enables a guest VM to execute malicious code on the underlying host, then tunnels a connection to it. An orthogonal attack is *VM hopping* [Tsai et al. 2012], which allows an attacker to move out from one VM to compromise another VM on the same host.

*Cross-VM Attacks*. In contrast to the attacks just described, whose goal is to escape from the VM to control (i.e, getting access to) the hypervisor or another coresident VM, *Cross-VM attacks* aim at attacking coresident VMs. Examples of such attacks are limiting the availability of the VMs or retrieving sensitive information from them. These last attacks are usually implemented by leveraging various side channels, such as cache covert channels, network-based (such as network watermarking and fingerprinting) or memory-based (memory deduplication, memory bus covert channels). In fact, since VMs are sharing the same hardware resource, such as the CPU (e.g., caches),

[3]This technology provides hardware support for I/O Memory Management Units (MMUs), for direct memory access (DMA) and interrupt virtualization.

memory (e.g., buffers) and the network (e.g., virtual interfaces), it is possible to extract information from the cohosted VMs. As an example, Xu et al. [2011] show how to exploit L2 cache covert channels to steal small amounts of data from a coresident VM, and Zhang et al. [2012] show how a malicious VM can exploit the L1 cache to capture the target VM's private ElGamal decryption key. Wu et al. [2012] exploit the memory bus as a high-bandwidth covert channel medium and demonstrate realistic covert channel attacks on various virtualized x86 systems. *HomeAlone* [Zhang et al. 2011b] is a tool for detecting the existence of side channels on the same host to launch the attack by examining neighbor VMs' L2 cache memory activity. Irazoqui et al. [2014] demonstrate a Cross-VM Flush+Reload cache attack to recover the keys of an AES implementation of OpenSSL running inside the victim VM, where the VMs are located on separate cores. A similar attack for a commercial PaaS Cloud is proposed in Zhang et al. [2014], which has been exemplified with three real attacks. The proposed framework exploits the knowledge of the control-flow graph of an executable shared with the victim of the attack. The authors propose building a so-called *attack nondeterministic finite automaton* to determine which memory chunks should be monitored to trace and characterize the victim's execution for specific attacks. Last-level cache (LLC, basically L3 cache) attacks are discussed by Liu et al. [2015], who show how LLC presents a high-bandwidth channel that can be exploited to mount (cross-core) Cross-VM side-channel attacks. Since this kind of cache is shared across all cores of the CPU, the impact of such an attack can be much larger. Another L3 attack is discussed in Irazoqui et al. [2015], which does not rely on the deduplication features required by Cross-VM Flush+Reload, hence is applicable in a larger number of hypervisors. The authors demonstrate the viability of recovering AES keys when attacker and victim are located in different cores in less than 3 mins.

One of the issues in this class of attacks, as well as for VM Hopping attacks, is the identification of where a desired VM is located. Given this knowledge, an attacker then strives to place the malicious VM on the same host to finally launch the attack using these techniques. Ristenpart et al. [2009] show that it is possible to (i) map the internal Cloud infrastructure, (ii) identify where a particular target VM is likely to reside, and (iii) instantiate new VMs until one is placed coresident with the target. Another attack, called *resource-freeing attacks*, modifies the workload of a victim VM in a way that it frees up resources for the benefit of the attacker's VM [Varadarajan et al. 2012].

*3.2.2. Attacks to Hypervisor.* Several research solutions enhance the hypervisor to defend the system from attacks to VMs, such as the ones discussed in the previous section. The idea is that, by moving the enforcement and protection mechanism into a lower level, attackers cannot hide their actions. This is also true for attackers, who strive to reach the VMM level to control all the other VMs. Hence, it is a "race to the bottom" between the attackers and the defenders trying to attack, and protect, the lowest possible level. The basic assumption to insert a protection mechanism into hypervisors is that usually they have a code base that is much smaller than conventional OSes[4] and should have less bugs (vulnerabilities) and are more difficult to subvert. Unfortunately, despite the advances due to hardware virtualization and the leverage of various functionalities in host OS kernels, contemporary hypervisors such as Xen and VMware have a large and complex code base[5] thus have a potentially wide attack surface. Moreover, within the current code base, several components are rather

[4]As a rule of thumb, hypervisors have a code size in the order of 100K lines of code, whereas OSes have a code size in the order of 10M lines of code, hence with an approximate ratio of 1 over 100 lines.

[5]Xen has more than 200K source lines of code, while the KVM kernel module alone contains 33.6K source lines of code of TCB [Wu et al. 2013].

complex, such as memory virtualization and guest instruction emulation, and often offer venues to various exploitable vulnerabilities.

Between January 2012 and June 2015, the National Vulnerability Database (NVD) recorded 39 vulnerabilities for hypervisors with a severity higher than 7, that is, those classified “high”, which means that they are very critical from a security point of view (see Table I). From the table, we can see that there have been 18 vulnerabilities reported for Xen, 18 for VMWare Esxi, 1 for Hyper-V, and 2 for KVM (one shared with Xen, the CVE-2015-3456, the “VENOM” vulnerability). Some of these vulnerabilities allow an attacker to directly compromise the hypervisor, for example, by escaping from a VM. Once a hypervisor is compromised, the attacker can further take over all the VMs it hosts, which could lead to not only disrupting hosted services, but also leaking potentially confidential data contained within guest VMs. Perez-Botero et al. [2013] characterize hypervisor vulnerabilities in three dimensions: the trigger source, the attack vector, and the attack target. The authors have classified Xen and KVM vulnerabilities into 11 functionalities (*attack vectors*) that are provided by a hypervisor. The study shows that the most common trigger source is guest VM user space, accounting for 39% of Xen and 34.2% of KVM vulnerabilities. Some of the attacks (and solutions) mainly focus on code or control-flow integrity without considering the so-called *noncontrol data attacks* [Chen et al. 2005], that is, attacks that do not tamper with control data to divert the control flow of the program. However, Ding et al. [2013b] show how to construct attacks that target hypervisor noncontrol data to demonstrate which types of data in the hypervisor's code are critical to system security by showing that privilege, resource utilization, and security policy–related data are vulnerable to return-oriented programming or DMA attacks.

*Attacks from Hypervisors*. As previously discussed, once hypervisors have been attacked, they can be exploited to attack other VMs on the same host. Other venues of attack may stem from high privileged tools available to administrators. These tools, if attacked, may also provide means for malicious users to control the system. For example, the introspection capability (see Section 4.1) available to administrators of the VMM, if attacked or misused, might threaten the VMs' confidentiality and integrity, as shown by *DKSM* [Bahram et al. 2010], an attack against kernel structures that evade VM introspection by providing it with false information. In fact, an implicit assumption of VM introspection is that the guest OS uses the kernel data in a predetermined way. But, if the guest OS is compromised, any assumption concerning the kernel respecting data structures may become false. Furthermore, management consoles are at the core of administrating VMs, and are very attractive for attackers. If an adversary manages to get access to the management console of the hypervisor, the entire security of virtual environment is at risk. An example is *VM sprawl*, which is the excessive creation of VMs to waste resources and create more entry points for attackers [Chen and Noble 2001].

*3.2.3. Lower-Level Attacks.* If the threat model considers that physical access is not controlled, then high-impact attacks on the host become possible that make it easier for an attacker to subvert the security of the system. Often, in these cases, it might not always be possible to find an appropriate security solution. We discuss some attacks that exploit DMA, SMM, and BIOS and Trusted Platform Module (TPM).

*DMA Attacks*. *DMA malware* is a class of malware that exploits the DMA to launch stealthy attacks on a system by executing on dedicated hardware. An example is *DAGGER* [Stewin and Bystrov 2013], which is a keylogger that attacks Linux and Windows platforms and is executed on the Intel's Manageability Engine processor. The authors show that, currently, a host has no reliable means to protect itself against DMA malware and that even with IOMMU-enabled platforms, there exist several issues that are not easily tackled.

*SMM and BIOS Attacks*. The SMM is a separate CPU mode from the protected mode and real mode, which provides a transparent mechanism for implementing system-control functions, such as power management and system security. SMM is implemented by the Basic Input-Output System (BIOS) and is entered via the system management interrupt (SMI) when the SMM interrupt pin is asserted. The microprocessor automatically saves its entire state in a separate address space known as system management RAM (SMRAM) and enters the SMM to execute an SMI handler. The handler then executes the `rsm` instruction to exit the SMM. The SMRAM is inaccessible from other CPU modes (while not in SMM); hence, it can act as a trusted storage space. Embleton et al. [2008] propose a proof-of-concept *SMM rootkit* that implements a chipset-level keylogger and a network backdoor capable of directly interacting with the network card to send logged keystrokes to a remote machine. Another attack [Wojtczuk and Rutkowska 2009] modifies SMM memory via Intel CPU cache poisoning, in which the attacker can make the CPU execute the SMM code from cache instead of DRAM.

On the other hand, the BIOS is used to initialize the hardware devices, including the processor, main memory, chipsets, hard disk, and other necessary I/O devices. BIOS code is normally stored on a nonvolatile ROM chip on the motherboard. Since the BIOS is charged with the correct initialization of the SMM, it also needs to be protected from malicious tampering. If an attacker is able to attack the BIOS, that attacker can control the SMM or invalidate the chain-of-trust procedure of the TPM. As an example, Kauer [2007] modifies the BIOS by showing that, by rebooting the system, the Core Root of Trust for Measurement (CRTM) of the TPM can be changed without being noticed. Butterworth et al. [2013] also present a vulnerability that allows an attacker to take control of the BIOS update process and reflash it with an arbitrary image despite the presence of signed enforcement. This class of rootkits has been called "Ring -2 rootkits" (level -1 being hypervisor rootkits). Another, lower-level, class of rootkit is called "Ring -3 rootkit" [Tereshkin et al. 2009], which are essentially rootkits trying to subvert the chipset functionalities, such as the Intel Active Management Technology (AMT), to install backdoors (see Figure 3 for the various levels of rootkit deployment). Note that rootkits have been demostrated also for other low-level components, such as Unified Extensible Firmware Interface (UEFI, a low-level interface designed to replace BIOS) [Hudson and Rudolph 2015] and hard-disk firmware.[6]

*TPM Attacks*. This class of attacks tamper with the TPM to dump the content of its internal registers, and retrieve its private data (e.g., private keys), or to disable its functions Sparks [2007]. *Replay attacks* are also possible, as shown in Bruschi et al. [2005], that is, an attacker can capture a message exchanged between the TPM and some authorized user, then use the same message later in a malicious way. Sparks [2007] discuss time-of-check-to-time-of-use attacks against the TPM, a bus attack on the low pin count bus as well as side-channel attacks.

## 4. SOLUTIONS

The protection of the system running inside a VM can be enforced either from the OS itself of the guest VM or from a component that is independent of the system being monitored, that is, from an Admin VM or from hardware. These two distinct approaches are known in the literature as "in the box" and "out of the box" [Jiang et al. 2007]. The in-the-box method generally relies on the security hooks provided by the monitored OS, for example, a Linux kernel patched with Integrity Measurement Architecture

[6]http://www.malwaretech.com/2015/06/hard-disk-firmware-rootkit-surviving.html.

(IMA) [Sailer et al. 2004] or PRIMA [Jaeger et al. 2006], which is able to measure the executable files through the Linux Security Module (LSM) or Linux Integrity Module (LIM). In general, the disadvantages of these methods are that (i) since modifications must be made to the monitored OS, they are not always deployable on running systems and may also introduce further vulnerabilities; and (ii) such methods rely on the OS being trusted, therefore are vulnerable to kernel rootkits. To overcome these shortcomings, VMI [Garfinkel and Rosenblum 2003] has been proposed as an "out-of-the-box" approach that enables an Admin VM to access the memory and virtual CPUs of a monitored VM to check the state of the system, for example, some critical data structures, at the kernel or at the user level. To improve performance and stealthiness, VMI-based approaches have been proposed using additional hardware [Zhang et al. 2002; Petroni et al. 2004]. In devising protection mechanism with a limited attack surface, other solutions have removed the Admin VM from the TCB by enhancing the hypervisor directly [Litty and Lie 2006]. This requires that only the hypervisor is protected from attacks. To this end, several solutions have proposed the protection of the hypervisor from runtime attacks [Wang and Jiang 2010], or a micro-hypervisor architecture [Murray et al. 2008; Steinberg and Kauer 2010], or the introduction of a further nested layer of virtualization [Carbone et al. 2008; Zhang et al. 2011a]. All these approaches will be categorized and detailed next.

### 4.1. Using an Admin VM for Protection

In this category of solutions, an Admin VM can analyze the state of the processes and of the kernel hosted on the monitored VMs using an introspection interface exported by the hypervisor. In particular, the TCB includes the Admin VM, the hypervisor, and the lower levels; the attack surface includes the monitored VM, both at the kernel and user level.

*4.1.1. Virtual Machine Introspection-Based Solutions.* *Livewire* [Garfinkel and Rosenblum 2003] was the first prototype of an intrusion detection system (IDS) that monitors VMs through introspection from an Admin VM. One of the problems with VMI is the *semantic gap* between the activity of the VM (in terms of processes, files) and the low-level view of the introspection interface. One of the first libraries to implement VMI is *XenAccess* [Payne et al. 2007], which is used for monitoring OSes on Xen. *XENKimono* [Quynh and Takefuji 2007] detects violations of the kernel level through VMI using two distinct strategies: (i) integrity checking of illegal changes to kernel code and system call table, Interrupt Descriptor Table (IDT), and page-fault handler; and (ii) comparison from data as seen inside and from outside the VM to detect malicious modifications to critical kernel objects. *VMwatcher* [Jiang et al. 2007] is another out-of-the-box approach, which exploits a technique called *view casting* to reconstruct internal semantic views (e.g., files, processes, and kernel modules) of a VM from the outside. This tool can be used to apply: (i) comparison-based malware detection, which compares a VM's semantic view obtained from both inside and outside to detect any discrepancy; and (ii) out-of-the-box execution of off-the-shelf antimalware software.

Other VMI-based solutions try to detect hidden processes running in the monitored VM, which are usually an indicator of a rootkit being installed in the monitored VM. *Lycosid* [Jones et al. 2008] exploits cross-view validation to detect maliciously hidden OS processes by comparing the lengths of the process lists obtained, respectively, from the Admin VM and from the monitored VM. The *VIX tools* [Hay and Nance 2008] support a forensic analysis of a guest VM from an Admin VM by including a suite of tools that mimic the behavior of common Unix command line utilities. Christodorescu et al. [2009] adopt virtualization to monitor and protect the systems running in the Cloud from a centralized Admin VM. The proposed solution does not assume any *a* *priori* semantic knowledge of the guest OS or any trust assumptions into the state of the VM. Another out-of-the-box approach is discussed in Xing et al. [2014], which measures the integrity of critical files through system call interception and without any modification of the guest VMs.

Several papers have proposed techniques to reduce the semantic gap. For example, Srinivasan et al. [2011] present a technique called *process out-grafting*, which relocates a suspect process from inside a VM to run side-by-side with the out-of-the-box VM. *SYRINGE* [Carbone et al. 2012] protects the monitoring application by moving it into a separate VM where it can invoke guest functions using function-call injection. Another solution is *Virtuoso* [Dolan-Gavitt et al. 2011], which is an approach to automatically creating introspection tools. The solution analyzes dynamic traces of small, in-guest programs that retrieve the desired introspection information, then produces similar programs that retrieve the same information from outside the guest VM. Similarly, *VMST* [Fu and Lin 2012] automatically generates VMI tools by identifying the introspection-related kernel data and redirecting these accesses to the in-guest OS memory. To reduce the huge overhead of the previous solutions, Saberi et al. [2014] exploit *online memoization* to cache the trained metadata in an online fashion to execute the inspection command (such as `ps`, `netstat`, and so on).

### 4.2. Using the Hypervisor for Protection

The second method to protect applications and OS running in a VM is to exploit the presence of the hypervisor, for example, by enhancing it with mechanisms to protect the VMs that run on top of it. These solutions further reduce the TCB by removing the Admin VM from it. Some of these solutions perform code authorization at kernel level. For example, *Manitou* [Litty and Lie 2006] ensures that a VM executes only authorized code by enabling the executable bit of the virtual page containing the code only if its hash belongs to a list of authorized hashes. Other solutions periodically check the integrity of the OS kernel from the VMM [Xu et al. 2007]. As an example, *HIMA* [Azab et al. 2009] is a hypervisor-based solution to measure the integrity of VM by monitoring critical guest events and memory protection. It maintains the measurements of the code segments of all kernel components and user programs running inside the guest VMs. *HookSafe* [Wang et al. 2009] is a hypervisor-based system that relocates kernel hooks to a dedicated page, then exploits hook indirection to regulate accesses to them through hardware-based, page-level protection. *KvmSec* [Lombardi and Di Pietro 2009] is an extension to Linux KVM with the ability to check the integrity of the guest VMs. Hofmann et al. [2011] present *OSck*, a hypervisor-based system that protects the system call table through hardware page protection and a hypervisor call that ensures that, once the table is initialized, the guest OS may not modify it by setting as read-only the hardware page protections on pages containing kernel text. Finally, *AccessMiner* [Fattori et al. 2015] is a hypervisor-based detector that models the activities of benign applications to the OS and uses the extracted models to detect the presence of malicious applications.

Other VMM-based solutions exploit memory techniques. As an example, *NICKLE* exploits *memory shadowing*, in which the VMM maintains a shadow physical memory of a running VM and performs kernel code authentication so that the shadow memory stores only authenticated kernel code. *Overshadow* [Chen et al. 2008] exploits *multi-shadowing*, which leverages the extra level of indirection offered by memory virtualization in a VMM to offer a VM context-dependent mapping. This mechanism presents an application with a clear-text view of its pages, and the OS with an encrypted view to provide confidentiality and integrity. However, *Iago attacks* (in which the semantics of system calls is maliciously changed) have been shown to be possible in these scenarios [Checkoway and Shacham 2013].

Finally, other solutions of this category are aimed at protecting the VM from *untrusted components*. For example, *CHAOS* [Chen et al. 2007] uses a VMM to mediate privileged operations, in which the goal is to protect a trusted process from exposing its private data and prevent tampering from a compromised OS kernel and other processes. Yang and Shin [2008] present an approach for using hypervisors to protect application data privacy even when the OS cannot be trusted. The hypervisor encrypts and decrypts each memory page requested depending on the application's access permission to the page. *HUKO* [Xiong et al. 2011] is a hypervisor-based integrity protection system designed to protect commodity OS kernels from untrusted extensions by confining their behavior through mandatory access control policies and hardware-assisted paging. *InkTag* [Hofmann et al. 2013] is a hypervisor-based system that protects trusted applications from an untrusted OS, allowing them to securely use the OS services. It does so by introducing *paraverification*, in which an untrusted OS is required to verify its own behavior by communicating its intent to the hypervisor. Wen et al. [2013] propose a solution to protect VMs from VMMs in multiprocessor Cloud environments by exploiting hardware mechanisms to enforce access control over the shared resources (e.g., memory spaces). *STEALTHMEM* [Kim et al. 2012] is a system-level protection mechanism against cache-based side-channel attacks in the Cloud. The system manages a set of locked cache lines per core, which are never evicted from the cache, and multiplexes them so that each VM can load its own sensitive data into the locked cache lines. *HyperShot* [Srivastava et al. 2012] removes the Admin VM from the TCB and implements a protocol for Cloud environments that allows tenants to request and verify the runtime status of VM snapshots even in the presence of malicious administrators.

### 4.3. Protecting the Hypervisor

In the solutions described so far, the threat model assumes that the hypervisor is trusted. However, this assumption is sometimes not true, as demonstrated by the attacks discussed in Section 3.2.2. Hence, several approaches focus on protecting the integrity of the hypervisor itself. We categorize the existing research solutions by proposing the following classes:

—*Formal verification:* formally prove the correctness of the hypervisor (threat model: the hypervisor is considered trusted);
—*Hypervisor hardening:* these solutions aim to protect the integrity of the hypervisor, for example, by protecting from static attacks, control-flow attacks, and noncontrol-data attacks (threat model: the hypervisor is untrusted but *sanitizable* [Lacoste 2013]);
—*Minimize hypervisor TCB:* these approaches split functionalities among different components, and remove some of these components from the TCB, using an approach similar to microkernel approaches (threat model: the hypervisor is untrusted but is hardened);
—*Nested virtualization approaches:* these solutions add another layer below the hypervisor (threat model: the hypervisor is untrusted, but the new layer is in the TCB);
—*Hardware-assisted solutions:* exploiting hardware features, such as SMM, to protect the hypervisor (threat model: the hypervisor is untrusted);
—*Introducing additional hardware:* the hypervisor is protected using hardware (threat model: the hypervisor is untrusted);
—*Using root-of-trust:* these solutions allow the trusted loading of the hypervisor (threat model: the hypervisor is untrusted).

Each of these techniques will be detailed next.

*4.3.1. Formal Verification.* One approach to remove the hypervisor from the threat model is to formally verify that it is secure. For example, some approaches can prove the absence of known classes of vulnerabilities, such as buffer overflows and null pointer dereferences, on small micro-kernels (e.g., seL4 [Klein et al. 2009]). However, these approaches also impose strict requirements on the micro-kernel design and implementation and, even if attractive, they require significant efforts to the design of hypervisors. We also note that the size of micro-kernel that can be formally verified is around 10K LOC, whereas commodity hypervisors greatly exceed this size (in the order of 100K LOC), thus this approach is applicable in very few cases in practice.

*4.3.2. Hypervisor Hardening.* If the absence of known bugs cannot be ascertained, other techniques can be used to make it difficult to exploit them. *HyperSafe* [Wang and Jiang 2010] implements two techniques to protect a Type I VMM: (i) nonbypassable memory lockdown, in which, once a memory page is locked down, it guarantees that the page needs to be unlocked first before being modified; (ii) restricted pointer indexing, which leverages the memory lockdown to extend the protection coverage from hypervisor code to control data. *HyperVerify* [Ding et al. 2013a] is an architecture to monitor hypervisor noncontrol data using an introspection VM. Other approaches advocate a *self-protection paradigm* [Wailly et al. 2012] by enforcing hypervisor protection using security loops that control different VMM layers. To this end, hooks mediate and control interactions between a device driver and the VMM environment.

*4.3.3. Reducing the Hypervisor TCB.* Other solutions revisit hypervisor design by proposing new architectures in an attempt to minimize its TCB [Murray et al. 2008; Steinberg and Kauer 2010]. *NOVA* [Steinberg and Kauer 2010] is a microkernel-based VMM that decouples the VMM into a modular system by introducing a capability-based access control for the several components inside a VMM. Another approach that has been advocated is to isolate buggy or untrusted device drivers of the hypervisor [Sharif et al. 2009]. *HyperLock* [Wang et al. 2012] similarly creates a separate address space in the host OS kernel so that the execution of the hypervisor as a loadable module can be isolated. However, it still runs in privileged mode and requires additional techniques to avoid possible misuse of privileged code. *NoHype* [Keller et al. 2010; Szefer et al. 2011] is a system that implements a microkernel-like hypervisor to provide only preallocated memory and processor core to a guest OS and by sharing only the network interface and disk. To reduce the hypervisor TCB, Gebhardt et al. [2010] exploit hardware protection mechanisms to isolate hypervisor security critical functions, in particular by moving the hypervisor in ring 0 of VMX root mode, whereas the nonsecurity TCB are executed in one of the other VMX root modes. *DeHype* [Wu et al. 2013] is a system that deprivileges hypervisor execution to user mode by decoupling the hypervisor code from the host OS, hence by reducing the attack surface. *MyCloud* [Li et al. 2013] is a reduced VMM system that removes the Admin VM from processor root mode and keeps only crucial security components in the TCB.

*4.3.4. Inserting an Additional (Software) Layer Below the Hypervisor.* Other solutions insert another layer below the hypervisor to check the integrity of the hypervisor. *GuardHype* [Carbone et al. 2008] is a hypervisor with a focus on VMBR prevention that allows the execution of legitimate third-party hypervisors but disallows VMBRs. GuardHype mediates the access of third-party hypervisors to the hardware virtualization extensions, effectively acting as a hypervisor for hypervisors. *CloudVisor* [Zhang et al. 2011a] exploits *nested virtualization* [Ben-Yehuda et al. 2010] to introduce a tiny security monitor underneath a commodity hypervisor. CloudVisor is responsible for protecting the confidentiality and integrity of the resources owned by VMs, whereas the hypervisor is still responsible for the (de)allocation of the resources to the VM. In other approaches, such as Strackx and Piessens [2012], a new (minimal) hypervisor is introduced to protect applications and kernels. The difference from the previous approaches is that this is a thin layer that sits below the OS and applications (it can be seen as a reduced hypervisor) and not a hypervisor below an existing hypervisor.

*4.3.5. Hardware-Assisted Solutions to Protect the Hypervisor.* Several approaches (e.g., [Azab et al. 2010]) have shown that inserting an additional software layer to protect the integrity of hypervisors may not be sufficient. In fact, the additional layer may introduce new vulnerabilities and could start another race with malicious attackers in obtaining the highest privilege in the system. Hence, some solutions have proposed to exploit hardware-supported schemes to monitor the integrity of hypervisors. For example, some works exploit the SMM mode by adding a small (locked) code to protect the integrity of the hypervisor [Rutkowska and Wojtczuk 2008; Wang et al. 2010; Azab et al. 2010]. One of the problems with these approaches is that they are not easy to update and/or deploy on commodity systems. *Bastion* [Champagne and Lee 2010] is a hardware-software architecture for protecting critical software modules in an untrusted software stack. It includes a processor and a thin hypervisor, which are both enhanced to provide secure execution and storage of these modules. *HyperCheck* [Wang et al. 2010; Zhang et al. 2014] is a hardware-assisted tampering detection framework designed to protect the integrity of hypervisors by leveraging the SMM and a PCI device to securely generate and transmit the full state of the protected machine to an external server. *HyperSentry* [Azab et al. 2010] is a framework for integrity measurement of a hypervisor that introduces a software component that is isolated from the hypervisor. HyperSentry uses an out-of-band channel to trigger stealthy measurements, and adopts the SMM to protect its base code and critical data. *SICE* [Azab et al. 2011] shows that building strong isolation mechanisms on top of an existing SMM requires a very small TCB of about 300 LOC. *DataSafe* [Chen et al. 2012] is an architecture that uses policies to protect data from attacks from untrusted third-party applications. To this end, DataSafe provides dynamic instantiations of secure data compartments and continuously tracks and propagates hardware tags to identify sensitive data by enforcing unbypassable output control. *HyperWall* [Szefer and Lee 2012] exploits hardware to provide VMs protection from hypervisor by allowing it to freely manage the memory, processor cores, and other resources of a platform, and by protecting the memory of the guest VMs from accesses by the hypervisor or by a DMA. The protections are enabled through modifications to the microprocessor and MMUs. *HyperCoffer* [Xia et al. 2013] is a hardware-software framework to protect confidentiality and integrity of a tenant's VMs that only trusts the processor chip. HyperCoffer requires some changes to the (untrusted) hypervisor by introducing a mechanism called *VM-Shim*, which runs in-between a guest VM and the hypervisor.

*4.3.6. Protection Using Additional Hardware Units.* This set of solutions does not rely on the hypervisor at all, as an example, by exploiting a *secure coprocessor* [Dyer et al. 2001; Zhang et al. 2002]. In Arbaugh et al. [1997], system integrity is verified at boot time; the root of trust is a small bootloader that computes a hash of the content loaded into memory, and compares this to a signed hash stored in secure ROM. A device is allowed to boot only if the two hashes match. AEGIS [Suh et al. 2003] is a processor to build computing systems secure against physical and software attacks. In the threat model, all the components external to the processor, such as memory, are considered to be untrusted. AEGIS provides a tamper-evident, authenticated environment in which any physical or software tampering by an adversary is guaranteed to be detected. *COPILOT* [Petroni et al. 2004] is a coprocessor-based kernel integrity monitor for commodity systems designed to detect malicious modifications to the host kernel. Dinaburg et al. [2008] proposes *Ether*, which is a malware analyzer that exploits hardware virtualization to reside completely outside the target OS environment. Finally, to protect security-critical processes and data from Cloud administrators, Seol et al. [2015] propose a trusted VMM to encrypt confidential data of guest VMs and a hardware security module called Trusted Cloud Module (TCM), implemented as an external PCI device, to store the cryptographic keys.

*4.3.7. Protection Using Root-of-Trust.* Some solutions ascertain the integrity of the hypervisor through hardware-based technologies, including TPM [Bajikar 2002], Intel [Greene 2012] and AMD SVM [Uhlig et al. 2005b]. These are capable of effectively establishing static, or dynamic, root-of-trust by guaranteeing that the hypervisor has loaded in a trustworthy manner, that is, that its code has not been tampered with and the correct version has been loaded. However, one of the main challenges with these approaches is how to maintain the same level of integrity throughout the lifetime of the loaded hypervisor. Since the absence of software vulnerabilities in the loaded components (e.g., the hypervisor) cannot be proven, we have to consider threats that exploit such vulnerabilities after it has been loaded. The goal of the *static root-of-trust* (also known as Static Root of Trust Measurement, SRTM) is to guarantee that the system is started in a known good state by booting it from some immutable piece of code. The assumption underlying this approach is that the code used for loading the system is trusted, that is, it cannot be modified. This code initiates the measurement process, in which each component measures the next one in a chain of trust. In particular, the hardware firmware code first calculates the hash of the BIOS, and extends a TPM's platform configuration register (PCR) with the value of this hash. Then, the BIOS continues this chain measurement with the PCI EEPROMs and the master boot record, before handling execution to them. Finally, the bootloader measures the OS loader before executing the OS or the hypervisor.

With Dynamic Root of Trust (DRTM), which is also referred as *secure late launch* [Kauer 2007; McCune et al. 2008], attestation functions can be invoked long after the normal boot sequence has completed. This allows the system to transfer control to known code after hardware-specific activities, such as device drivers, have initialized. The first implementation of DRTM on AMD processors was the Open Secure LOader (*OSLO*) [Kauer 2007]. Using this mechanism, the BIOS and bootloaders can also be removed from the trust chain. Similarly, Flicker [McCune et al. 2008] is an infrastructure for executing security-sensitive code in isolation and for providing attestation of code execution to a remote party. The TCB is only 250 lines of code and is meant to execute small sensitive pieces of code. SecVisor [Seshadri et al. 2007] is a tiny hypervisor that ensures code integrity for OS kernels by checking that only previously approved code can execute in kernel mode. SecVisor exploits late-launch technologies to boot the system safely. The disadvantages of these approaches is that one processor may interrupt all the other processors on the same machine, and it also requires developers to port their applications. Trusted Execution Environment [Dai et al. 2010] is an architecture that can run multiple instances of DRTM in parallel in a virtualized environment. *TrustVisor* [McCune et al. 2010] is a special-purpose hypervisor that provides code integrity as well as data integrity and secrecy for selected portions of an application, and is initialized via DRTM. SMART [Eldefrawy et al. 2012] is a primitive based on hardware-software for establishing a dynamic root-of-trust in a remote embedded device. Jayaram Masti et al. [2013] propose an architecture that enables the creation and management of multiple, concurrent, secure execution environments on multicore systems. The architecture relies on new processor extensions and a hardware-based virtualized TPM to support multiple, concurrent, dynamic root-of-trust requests from different VMs. Owusu et al. [2013] propose a new a set of CPU instruction set extensions for externally verifiable initiation and execution of an isolated execution environment.

Recently, new processor extensions have been devised to provide a form of integrity protection in Cloud environments. In particular, Software Guard Extensions (SGX) [McKeen et al. 2013; Anati et al. 2013] for Intel processors allow an application to instantiate a protected container, referred to as an *enclave*, which is a protected area in the application's address space having its own entry table, heap, stack, and code. The enclave provides both confidentiality and integrity of the application even in cases in which an attacker has gained the most privileged execution level. To this end, accesses to the enclave memory area from any software not resident in the enclave are prevented. Using SGX, the application to be protected can be distributed in clear. When the protected portion is loaded into an enclave, its code and data are measured and protected against external software access.[7] This new set of extensions in a Cloud environment makes it possible to attest a single portion of a host memory without interrupting the execution of other software on the machine. As an example, *Guardat* [Vahldiek et al. 2014] is an architecture that enforces data access policies at the storage layer. The enforcement can be implemented in a trustlet within a VMM (or OS) isolated using a container with SGX. *Haven* [Baumann et al. 2014] is a system designed to run unmodified legacy applications in the Cloud by exploiting the protection of SGX to provide integrity and confidentiality of a tenant's applications. In the considered threat model, the adversary may have full control of the hardware, excluding the processor but including memory and I/O, besides the entire software stack. The system is structured in such a way that Cloud providers are also protected from misbehaving tenants' applications. *VC3* [Schuster et al. 2015] is a system based on SGX to run distributed MapReduce computations in the Cloud; it keeps the code and data secret as well as ensuring the correctness and completeness of the result. The user code is loaded into an SGX enclave, exploiting a key exchange protocol to get the keys needed to decrypt the provided map and reduce functions. The TCB contains Hadoop, while the OS, and the VMM, are outside the TCB. Hence, in the considered threat model, an attacker can attack the OS and hypervisor, as well as record, replay, and modify network packets.

## 5. FRAMEWORK TO CATEGORIZE, DEFINE, AND EVALUATE SECURITY SOLUTIONS

To categorize the solutions discussed so far, we propose a framework that takes into account their *threat model*, the *security properties* of the solution (goal and TCB), and how the *solution is implemented* (methodology and features).We believe that such a common framework is needed to evaluate different solutions under the same taxonomy. Furthermore, such a framework can be used when proposing a new security solution to clearly define the assumptions and the goals. To this end, in the following, we propose a categorization of the security solutions based on the *threat models* (security and trust assumptions), *security properties* (goals and TCB), and the *implementation strategy* (methodology and features).

### 5.1. Threat Models of the Solutions

We have seen that different threat models may exist in a virtualized system, each of which includes a different set of security and trust assumptions and possible threats against the assets. A *threat model* thus comprises a set of *threats* (and related *attacks*), *security assumptions*, and *trust assumptions*. The *security assumptions* form the basis from which security control is enforced. For example, if in one model one considers physical access not possible, attacks trying to subvert the hypervisor from the lower level are not taken into account by the solution, but may still exist. In another example,

[7]For more details, see the Software Guard Extensions Programming Reference: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

if the threat model considers the hypervisor to be with no (exploitable) bugs, whereas the OS is vulnerable, then a proposed protection solution can be deployed directly inside the hypervisor itself to consider possible attacks against the kernel. The *trust assumptions* define the level of trust in the system's components and its principals before security mechanisms are deployed. Trust assumptions in the threat models might be based on trust in the Cloud providers, in SLAs, on an assessment, on proofs, or on a black-box approach. Trust can also be defined as an *accepted dependence* [Avizienis et al. 2004], in which dependence of a component A on a component B represents the extents to which the security (dependability) of A depends on that of B. In our case, we are interested in defining the *trust of actors* (willingness/capacity to operate in a compliant way) that have direct access to these components (i.e., a component B depends on the trust of the actor A). The notion of *dependability*, that is, the ability to deliver service that can justifiably be trusted, can be related to the security assumptions regarding the components, in particular, in terms of their *trustworthiness* (i.e., assurance that the component will perform as expected) and *control* (which actor is supposed to be in control of the component). In our model, the threats are generic (e.g., provider and an external attackers) rather than specific, as in Avizienis et al. [2004] (administrators, users, providers, infrastructures, intruders, and so on), and we consider trust assumptions regarding the Cloud providers only. The same applies to the classes of attacks (or faults) in which, when defining their threat model, we are usually referring to the security of the components with respect to generic attacks. This is due to the fact that we are trying to depict the worst-case scenarios.

We start by categorizing the existing security and trust assumptions at the different levels.

*5.1.1. Security and Trust Assumptions.* Concerning this set of assumptions, the first thing to consider is how well protected is the physical access to the server hosting the tenants' VMs. Obviously, any device that can be easily accessed and tampered with cannot be easily protected (even if some form of hardware mechanism and obfuscation can make this task more difficult [Chakraborty and Bhunia 2009, 2010; Desai et al. 2013]). Then, we need to consider the assumptions at the upper levels.

*Security Assumptions at Physical Level*. Concerning physical assumptions, those that may impact the integrity and confidentiality of the tenant's system are categorized as follows:

—*Is physical access possible?* When access is restricted to trusted staff, that is, including the case in which physical security systems in the Cloud premises are in place (such as cameras, security personel, and so on) [Wang and Jiang 2010; Azab et al. 2010; Szefer et al. 2011], versus those in which attackers may have physical access to the system [Owusu et al. 2013].
—*Is physical tampering possible?* When hardware is protected by physical tampering [Suh et al. 2003]; this includes cases in which, even if physical access in not restricted, some antitampering mechanisms exist. If this is the cases, and depending on the quality and protection level of such mechanisms, physical attacks might be removed from the threat model.
—*Are BIOS/SMM or DMA malicious?* If we consider these threats as possible, such as DMA attacks, then solutions that are based on the protection mechanisms offered by the CPU (e.g., paging) are no longer valid, and they propose alternative mechanisms [McCune et al. 2008]. Other solutions explicitly remove SMM threats from the threat model [Li et al. 2013; Wang and Jiang 2010].
—*Is trusted boot present?* When trusted boot is considered to be present or not, one can assume the loading of the hypervisor to be trustworthy and focus on protecting other functionalities or the system [Butt et al. 2012; Strackx and Piessens 2012].

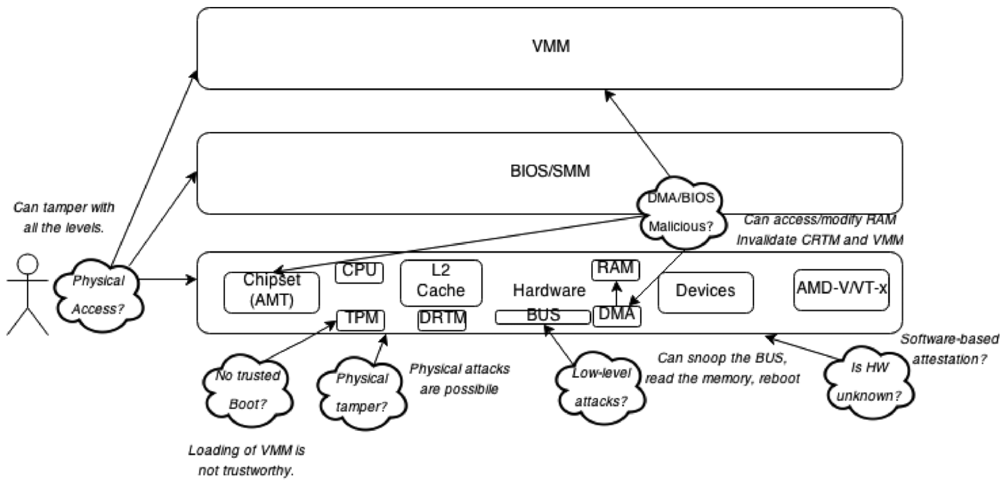


Fig. 4. Security assumptions at physical level.

—*Is the hardware known?* When the hardware is known (CPU, memory, and so on), that is, we assume to know the model of the CPU, its frequency, and the amount or RAM. For software-based attestation solutions [Seshadri et al. 2005; Perito and Tsudik 2010], this is usually a mandatory requirement since, being based on timing requirements, it is mandatory to know the model of the hardware components exactly.

Note that some attacks usually are not considered at all, such as monitoring the high-speed bus that connects the CPU and memory, because they are very difficult to implement and the protection mechanisms to resist such attacks are difficult to deploy. Figure 4 shows some of the possible attacks, discussed in Section 3, related to each of these assumptions in the case that they are not satisfied.

*Security Assumptions at the Virtualization Level*. Next, we need to categorize the assumptions concerning the VMs and the hypervisor:

—*Is the OS vulnerable?* When the OS is benign but may contain vulnerabilities [Criswell et al. 2014]: this includes cases in which the user trusts the initial setup of the OS inside the VM, but attacks are still possible due to the presence of bugs.
—*Is the OS untrusted?* When the OS is considered to be untrusted and can behave in any malicious way [McCune et al. 2008, 2010; Szefer et al. 2011; Jayaram Masti et al. 2013], that is, we assume that it can be remotely (locally) attacked. In this case, the owner of the VM does not trust the VM OS. A class of attacks due to improper semantics of the OS are the Iago attacks [Checkoway and Shacham 2013], in which the results of the system call are manipulated in such a way that the requesting application is tampered with. In this case, the OS is completely untrusted.
—*Is introspection enabled?* If this condition is met [Garfinkel and Rosenblum 2003; Payne et al. 2007; Jiang et al. 2007], an attacker gaining access to a privileged VM is able to read sensitive data of any guest VM. Furthermore, it might be able to modify the semantics of programs running inside VMs or configuration files.
—*Is the Admin VM protected?* When the *Admin VM* is considered to be protected [Ding et al. 2013a] or may be vulnerable to attacks or is considered to be malicious [Butt et al. 2012; Srivastava et al. 2012]; in these cases, the attacks might access the Admin VM from a guest VM or remotely; from this privileged VM, they may perform

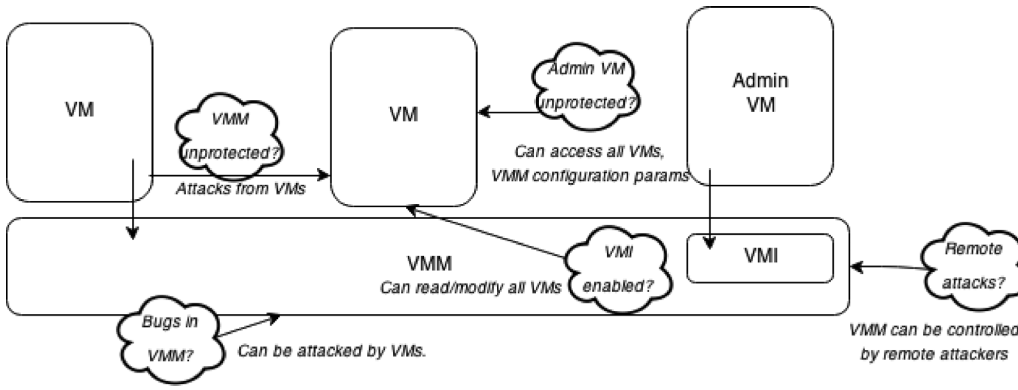


Fig. 5. Security assumptions at the virtualization level.

a denial-of-service attack, such as stopping a tenant VM. An attacker may also read or write to any portion of memory of the guest VM in an invisible manner.

—*Is the configuration interface protected?* When a (remote) attacker can modify, read, or restart any other VMs using the hypervisor configuration interface [Santos et al. 2012; Zhang et al. 2011a]. This might happen because the configuration interface (e.g., the Web interface) is not well protected (e.g., open port to the outside and easy password) or the hypervisor itself is not updated and vulnerabilities may be exploited. Otherwise, it is assumed to be trusted [Szefer et al. 2011].

—*Are there any vulnerabilities in the hypervisor?* When a hypervisor is considered trusted [Szefer et al. 2011] or it may contain vulnerabilities [Zhang et al. 2011a]: for example, solutions can consider that the hypervisor is trustworthy (and possibly loaded with SRTM technologies [Wang and Jiang 2010]), but it may contain vulnerabilities that can be exploited by an attacker [Azab et al. 2010].

—*Is the hypervisor protected?* When protection is in place against attacks on the hypervisor [Xiong et al. 2011] by guest VMs or remote attacks.

Figure 5 shows some of the possible attacks related to these assumptions in the worst case, that is, if these assumptions are considered as false in the threat model.

*Trust Assumptions at Cloud Level*. In this context, the trust level is related to the fact that tenants lose part of their control over the hardware and software components. This loss of control means that they have to trust the provider to some extent. The trust assumptions on the Cloud provider are categorized as follows:

—*Is the provider trusted?* When providers are trusted so that no attacks are considered possible on the virtualized system, especially low-level attacks [Szefer et al. 2011; Li et al. 2013].

—*It the provider honest-but-curious?* When system administrators of guest VMs are trusted but may have the opportunities to access the user's confidential data [Xiong et al. 2012]. In this case, it is assumed that they cannot alter programs or configuration files of VMs or the hypervisor.

—*Do we consider insider threats or malicious administrator?* Even if the providers can be trusted, it might be the case that either some untrusted staff or trusted staf unwillingly perform operations that might impact the integrity or confidentiality of VMs [Srivastava et al. 2012], such as deleting files or stopping a VM.

—What is the Cloud service model? When the tenant is using a IaaS/PaaS/SaaS model [Mell and Grance 2011]: this sets the tenant's level of access to the resources, that is,

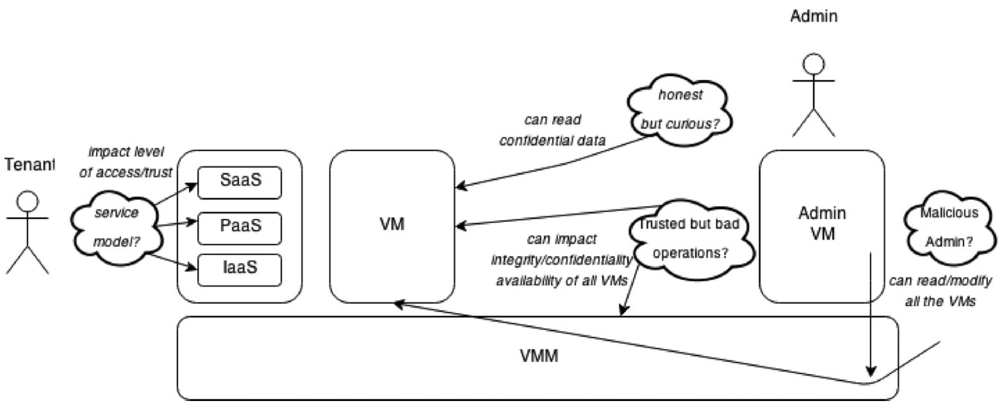


Fig. 6. Security assumptions at the Cloud level.

Table II. Assumptions in the Threat Model

| Assumptions | Meaning |
|---|---|
| **Hardware-Level** | |
| Physical attacks | When physical access is possible. |
| BIOS/DMA malicious | When low-level attacks are possible. |
| SRTM/DRTM | When trusted boot is present. |
| **Virtualization-Level** | |
| OS vulnerable | When the OS can have bugs. |
| OS untrusted | When the OS can be controlled by attackers. |
| Introspection untrusted | When the VMI can be used by attackers to read/modify VMs. |
| Admin VM untrusted | When the Admin VM can be used to access VMs and VMM. |
| Hypervisor vulnerable | When the hypervisor can be attacked. |
| Hypervisor untrusted | When the hypervisor can be controlled by attackers. |
| **Cloud-Level** | |
| Trusted but misoperations | When the providers are trusted. |
| Honest-but-curious | When the provider can read data. |
| Malicious admins | When the provider can read and modify data. |

if the user is in control of the entire stack of the VM, only of the OS, the middleware, or of the applications.

Regarding the third point, some papers on protecting tenants' VMs in the Cloud [Butt et al. 2012; Szefer et al. 2011; Li et al. 2013; Santos et al. 2012] differentiate between Cloud service providers and Cloud system administrators. Cloud providers, such as Amazon EC2 and Microsoft Azure, have a vested interest in protecting their reputations. On the other hand, Cloud system administrators are individuals entrusted with system tasks and maintaining the Cloud infrastructure, and that have access to Admin VM and the privileges that it entails. In these papers, it is assumed that Cloud system administrators are either malicious or that they could make mistakes. By extension, the assumption is that the Admin VM is untrusted, that is, system administrators can perform any operation with high-privileges on the hypervisor and VMs, but they cannot physically access them. Also, in this case, we have depicted some of the possible attacks if each assumption is not satisfied (Figure 6).

*Combining Security and Trust Assumptions*. In Table II, we summarize the set of possible security and trust assumptions in the threat models. Note that each threat model includes only those assumptions that the designer of the solution believes to be considered. Hence, each protection solution is tailored for that specific threat model. For this reason, before exploiting a protection solution, it is important to understand what are the exact threats that one needs to cover. In fact, it might be the case that no solution exists if one considers the worst-case scenario, that is, with a very restricted threat model. Note also that, once the threat model has been defined, the classes of attacks on tenants' resources vary noticeably.

### 5.2. Security Properties of the Solutions: Goals and TCB

In our categorization of the security solutions for virtualized environments, we also categorize the solutions based on their *security properties*, in terms of *goals* (covered attacks) and *TCB*. In particular, we consider these goals:

—*Integrity of the applications*: Protection from attacks aiming to modify the applications running in the VM, usually by exploiting standard techniques [Abadi et al. 2005] but by taking advantage of virtualization technology, for example, by placing them at a lower level;
—*Integrity of the VM* (OS and applications): Protection from attacks aiming to modify the VM; another distinction here is between *static protection* (i.e., the correct loading of the VM [Sailer et al. 2004; Jaeger et al. 2006]) versus *dynamic protection* [Quynh and Takefuji 2007] (e.g., runtime checking of the semantics of the OS [Criswell et al. 2014] or of the applications [Baiardi et al. 2009]);
—*Integrity of the VMs by untrusted components* (VM isolation): Protecting the integrity of the VMs by isolating them from an untrusted OS [Chen et al. 2008] or other components, such as drivers;
—*Integrity of the hypervisor*: Protection from attacks against the hypervisor. We further differentiate among solutions that provide *static integrity protection* (Hypeguard [Rutkowska and Wojtczuk 2008], Hypersentry [Azab et al. 2010], HyperCheck [Wang et al. 2010]), or *control-flow protection* (Hypersafe [Wang and Jiang 2010]), or *noncontrol-data protection* (HyperVerify [Ding et al. 2013a]);
—*Confidentiality of the data or program in the VM*: Protection from attacks whose goal is to illegally access (confidential) data of the tenants [Godfrey and Zulkernine 2014; Priebe et al. 2014; Kim et al. 2015];
—*Protection against specific threats*: Solutions suited for particular problems, such as kernel rootkits [Garfinkel and Rosenblum 2003], VMM detection [Carpenter et al. 2007], VM escape, VM hopping, Cross-VM, or for existing known vulnerabilities (CVE).

Other goals that we do not consider here, but that can be easily integrated in the framework, are: (i) the integrity of the computation—these attacks are aimed at modifying the results of a computation (there is a strong interest in this field in current papers, such as with verifiable computation [Parno et al. 2013; Vu et al. 2013]); and (ii) program or VM availability.

As far as regards the TCB, we have seen (in particular, in Section 4) that all the solutions rely on an (implicit) TCB, which is left outside the attack surface. Usually, the TCB is strongly related to security and trust assumptions, for example, if one assumes that the hypervisor cannot be attacked, then logically it is in the TCB. However, we have shown that some solutions also introduce additional software or hardware components that are also included in the TCB. For these reasons, we also explicitly include the TCB in the categorization. In particular, when analyzing the solutions discussed in Section 4, we have seen that there are many possible TCBs, among which the most common include:

—*The Admin VM and the hypervisor (and the layers below):* These are solutions based on an additional VM to check the integrity of the VM;
—*Only the hypervisor (and the layers below):* These are solutions based on monitoring the VMs from the hypervisor;
—*A reduced TCB hypervisor (and the layers below):* These are solutions that remove functionalities from the TCB of the hypervisor;
—*A nested microhypervisor (and the layers below):* These are solutions based on adding an additional software layer below the hypervisor;
—*Only the hardware:* These are solutions based on exploiting hardware-based mechanisms (such as the SMM mode or additional hardware).

In other cases, a custom TCB is proposed.

### 5.3. Implementation Strategy of the Solutions: Methodologies and Features

The next axis to categorize the solutions is their *implementation strategy*, in particular, in terms of *methodologies* and proposed *features*. We recall the categorization of the solutions for a virtualized environment that we proposed in the previous section:

—using an Admin VM for protection:
 —for example, VMI.
—using the hypervisor for protection;
—protecting the hypervisor:
 —formal verification;
 —hypervisor hardening;
 —minimize hypervisor TCB;
 —nested virtualization approaches;
 —hardware-assisted solutions;
 —introducing additional hardware;
 —using root-of-trust.

However, other methodologies are possible (they also depend on the state-of-the-art of the technology). We also consider the *features* of the solutions, such as: (i) the *transparency* of the solutions, that is, if the solution requires none or little changes to virtualization stack (OS, VMM) or if it requires modification of the computing platform (and at which level, i.e., libraries, or kernel level); (ii) if it can be used in any OS or hypervisor or only on specific ones (*portability*); and (iii) the *performance overhead* of the solution. Finally, those attacks that are not covered under the considered threat model define the *limitations* of the proposed solutions.

In the end, by combining the threat models, goals, TCB, methodology and features of the protection solutions, we can define of a common methodology to categorize and evaluate them. We have summarized all these features in Table III. This table is by no means exhaustive, that is, there can be other goals, assumptions, and methodologies that one can consider and add to it. By using the same approach, one can clearly understand the attacks that are considered by the protection solutions and only under what assumptions they are effective. We think that, by using the same taxonomy for defining, among others, the threat model, security properties and implementation strategy of a protection solution, it is possible to qualitatively and quantitatively evaluate a proposed solution more accurately. To show an instantiation of this categorization, in Table IV, we have summarized the threat models (security and trust assumptions), security properties (goals and TCB), and implementation strategy methodology and features) of some of the papers previously discussed in Section 4 (ordered chronologically).

Table III. Framework to Categorize, Define, and Evaluate Security Solutions

| Threat Model: Security Assumptions |
|---|
| Physical access |
| BIOS/DMA malicious |
| SRTM/DRTM available |
| OS vulnerable |
| OS untrusted |
| Introspection untrusted |
| Admin VM untrusted |
| Hypervisor vulnerable |
| Hypervisor untrusted |
| **Threat Model: Trust Assumptions** |
| Trusted |
| Honest-but-curious |
| Malicious admins |
| **Solution: Security Properties—Goals (Attack Covered)** |
| Integrity of the VM (static, dynamic) |
| VM isolation |
| Integrity of the hypervisor (static, control-flow attacks, noncontrol-data) |
| VM confidentiality |
| Specific threats (VMM Detection, VM Escape, VM Hopping, Cross VM, known CVE) |
| **Solution: Security Properties—TCB** |
| Admin VM and hypervisor |
| Hypervisor |
| A reduced TCB hypervisor |
| A nested microhypervisor |
| Only the hardware |
| **Solution: Implementation—Methodology** |
| Using an Admin VM for Protection (e.g., VMI) |
| Using the hypervisor for protection |
| Protecting hypervisor (e.g., formal verification, hardening, minimize TCB, ...) |
| **Solution: Implementation—Features** |
| Transparency |
| Portability |
| Performance |
| Limitations (e.g., attacks not covered) |

## 6. DISCUSSION

We have seen that virtualization technology has been used to prevent existing attacks, for example, to the OS and applications, but in a "clever way"; to prevent attacks facilitated by virtualization, such as Cross-VM; and to prevent attacks against the virtualization layer itself, for example, the hypervisor. Most of the proposed solutions address generic threats, such as integrity or confidentiality, instead of specific ones (such as VM Escape or a specific CVE). In general, the solutions that we have detailed are mainly targeted against integrity attacks in all their variants, whereas only a limited number address confidentiality issues (e.g., Cross-VM), and almost none consider availability issues.

In this survey, we have provided a thorough review of threats and attacks against a system running in a virtualized environment and the research solutions aimed at addressing a set of threats. We have seen that, to devise a protection mechanism in a virtualized environment, one needs to consider the possible threat models, that is, attacks, usage scenarios, trust assumptions at each architectural layer, for example, hardware, hypervisor, VM—the security of the system being linked to a set of threats and trust assumptions. One of the issues with this approach is that solutions consider only a specific threat model and are highly sensitive to variation to that model, for

Table IV. Comparison of Threat Models and Goals of Some Papers

| Paper | Security Assumptions | Trust Assumptions | Security Properties: Goal | Security Properties: TCB | Implementation: Methodology | Implementation: Features |
|---|---|---|---|---|---|---|
| McCune et al. [2008] | BIOS, OS, DMA may be malicious. Exploit AMD SVM. | N.A. | Execute sensitive code in isolation. Attestation of code execution. | 250 lines of code | DRTM | Applications need to be ported Pause all the processes |
| Wang and Jiang [2010] | No physical Attacks No BIOS/SMM attacks DMA attacks occur Hypervisor vulnerable Trusted boot to load hypervisor | N.A. | Integrity of the hypervisor (dynamic and noncontrol- data) | Loaded hypervisor | Hypervisor hardening | Modification required to hypervisors 5% overhead |
| Azab et al. [2010] | Out-of-band channel to enable SMI remotely Physical security in place Trusted boot to launch hypervisor SMRAM is protected | N.A. | Hypervisor integrity and attestation (instantiated in different typologies: code integrity, CFG integrity) | SMM Code | SMM-based Solution | Do not address periodic attacks in-between checks |
| McCune et al. [2010] | Untrusted OS and applications Untrusted DMA devices No physical attacks Requires/exploits DRTM | N.A. | Reduced TCB hypervisor Code and data integrity for selected portion of applications | 6K lines of Code | DRTM | 7% overhead Does not support multiprocessor Applications need to be ported to include PAL |
| Champagne and Lee [2010] | Software and hardware attacks | N.A. | Protection of SW modules in untrusted environments | Microprocessor chip | Hardware-assisted solution | Requires new microprocessor Requires ad-hoc hypervisor |
| Szefer et al. [2011] | Physical security controls in place Untrusted OS Trusted hypervisor | Trusted provider Trusted management interface | Attacks to VMM by guest VMs Isolation between VMs | New reduced VMM | Reducing TCB | Requires slightly modified OS available to tenants 1% overhead |
| Zhang et al. [2011a] | Trusted boot to launch Cloudvisor VMI is possible Management interface vulnerable Hypervisor untrusted No physical attacks | Honest provider misoperations can happen | Confidentiality and integrity of VMs from other VMs and VMM | 5K LOC of TCB | Nested virtualization | 1 LOC of modification to Xen Needs porting |
| Xiong et al. [2011] | Untrusted OS and extensions Trusted hypervisor | N.A. | OS integrity (static, data integrity and control-flow) | Hypervisor | Protection from hypervisor | Modification to Xen (3.3K LOC) Modification to Linux (450 LOC) |
| Strackx and Piessens [2012] | Attacker can execute arbitrary code at kernel level Trusted boot to load VMM No physical access | N.A. | Protect applications from OS and applications | TCB 7K | Nested virtualization | New hypervisors (1K) 3%–14% overhead |
| Butt et al. [2012] | Admin VM untrusted Trusted boot to check system-level TCB Physical security in place | CSP trusted Malicious sysadmins | Tenants VM security and privacy | Hypervisor and domain builder | Reducing hypervisor TCB | Modification to VMM |
| Santos et al. [2012] | Remote attackers can do anything No physical attacks Remote interface vulnerable Trusted boot to load hypervisor | Malicious administrator Misconfigura-tion | Data confidentiality (reveal data only to nodes satisfying a policy) | Hypervisor and trusted node | Trusted third party | Assume a trusted monitor node to certify nodes |

(Continued)

Table IV. Continued

| Paper | Security Assumptions | Trust Assumptions | Security Properties: Goal | Security Properties: TCB | Implementation: Methodology | Implementation: Features |
|---|---|---|---|---|---|---|
| Chen et al. [2012] | Attacker exploit applicationss or OS to leak sensitive data OS untrusted Secure boot is present No hardware attacks No analog hole | N.A. | Data confidentiality (only authorized applications can access data protect against data dissemination) | Only hardware | Hardware-assisted | Requires new hardware |
| Srivastava et al. [2012] | Malicious Admin VM Trusted boot to load hypervisor No hardware attacks | Trusted provider Malicious administrators | Provide attestation of integrity of VMs to tenants | Hypervisor | Protection from hypervisor | Adds 4K LOC to hypervisor Requires trusted third party |
| Hofmann et al. [2013] | Untrusted OS Trusted hypervisor | N.A. | Protect VM applications from untrusted OS | Hypervisor Trusted library | Protection from hypervisor | 3500 hypervisor LOC (extension to KVM) Add (trusted) user-library 5x–55x slowdown |
| Li et al. [2013] | No physical attacks Trusted boot Protection to SMM in place (custom BIOS) | Malicious sysadmins Provider trusted | TCB reduction Provide privacy to VMs VM-to-VMM attacks VM-to-VM attacks Insider attacks | TCB 5.8K LOC | Reducing TCB | New hypervisor 2%–9% slowdown |
| Wu et al. [2013] | No physical attacks Hypervisor vulnerable Host OS trusted | N.A. | Reducing hypervisor TCB | TCB 2.3K LOC | Reducing hypervisor TCB | Added 10 new system calls 6% overhead |
| Owusu et al. [2013] | Attacker can have physical access Can tamper with BUSes OS, App, Firmware untrusted | Trusted CSP Malicious administrators | Isolated execution environment | Only CPU | DRTM | Requires a new CPU |
| Jayaram Masti et al. [2013] | Hypervisor is trusted OS can be malicious | N.A. | Using TXT in multicore architectures (concurrent execution of secure environments, e.g., on the Cloud) | Hypervisor | DRTM | Requires changes to x86 platform |
| Xia et al. [2013] | Untrusted hypervisor Physical attacks are possible Requires secure processor VM not vulnerable | Malicious administrator | Protection of VMs from untrusted hypervisor | VM-Shim (1,100 LOC) | Hardware-based | 380 LOC of changes to hypervisor requires hardware support |
| Ding et al. [2013a] | Untrusted hypervisor Assume hypervisor code integrity is ensured Untrusted VMs can attack hypervisor Trusted admin VM DMA enabled | N.A. | Monitor hypervisor noncontrol data | Admin VM and loaded hypervisor | Hardening | VMI can be fooled Attacks in-between periodic checks |
| [Baumann et al. 2014] | Untrusted hardware (except processor) | Untrusted Cloud provider Untrusted admins | Integrity and confidentiality | LibOS (209MB) and Shield Module (180KB) | Hardware-based | Around 31%–54% overhead |

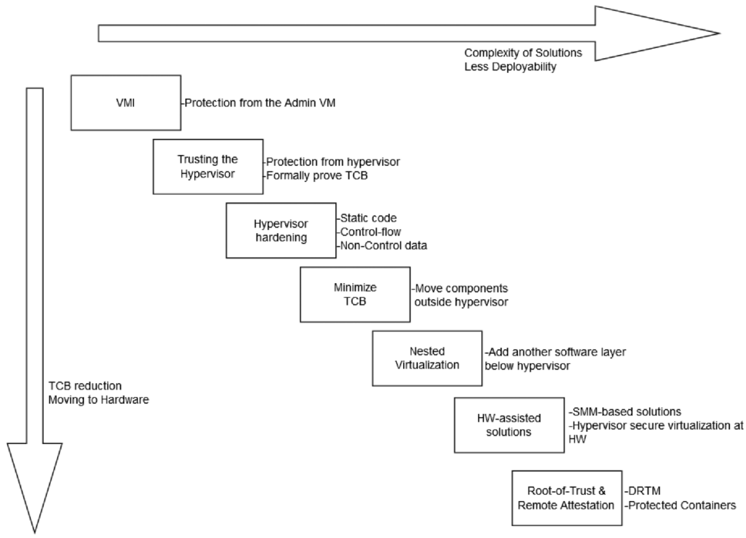


Fig. 7. Trends of virtualization-based solutions.

example, if an assumption is removed or a new threat is considered, the solution is no longer valid. Such changes occur naturally because new issues and attacks arise as new technologies emerge, as we have described for virtualization technology. We have seen that the hypervisor is a worthy target since, if an attacker is able to compromise the hypervisor of a physical host of the provider, that attacker is potentially able to access and modify any tenants' data or application, possibly without being noticed [Kortchinsky 2009; Elhage 2011]. Hence, many studies exploit virtualization to provide application integrity but assume that the hypervisor is trusted [Keller et al. 2010; Hofmann et al. 2011; Azab et al. 2009; Xiong et al. 2011]. This is the case for solutions based on virtual machine introspection [Garfinkel and Rosenblum 2003; Payne et al. 2007; Jiang et al. 2007], which check the memory of a VM for compromise and rely on the data gathered from an introspection interface exported by the hypervisor itself. However, in these scenarios, should a hypervisor be compromised, either fake data might be returned by the introspection interface or, worse, other VMs residing on the same host might by attacked from the compromised hypervisor. The argument that a hypervisor is secure is based on the observation of its restricted codebase, and narrow interface. However, as the hypervisor size is continuously increasing, and taking into account existing attacks against the hypervisor as well (e.g., VM escape), the assumption no longer holds and new threat models no longer consider it trusted [Szefer et al. 2011; Azab et al. 2010]. As we have depicted in Figure 7, we can see that proposed security solutions are increasingly moving closer to the hardware. Furthermore, we can clearly see a trend toward TCB reduction: this has happened as new attacks made it possible, and likely, to attack first the tenant VMs, then the Admin VM, then the hypervisor, and so on. In particular, regarding the hypervisor, we have seen that it has been initially assumed to be part of the TCB; then, solutions have directly enhanced it to protect it from external attacks; newer solutions have striven to reduce its TCB; finally, some recent solutions consider the hypervisor not part of the TCB anymore.

An interesting aspect that we want to underline is that, in the existing studies, threats are considered either possible or not possible, that is, threat models consider only worst-case scenarios, for example, the hypervisor is either fully compromised or trusted, without considering probabilities. Another interesting characteristic is whether solutions can be combined. We have seen that some can be merged, such as those that provide hypervisor integrity with those that guarantee the VM integrity, to produce a larger protection interface. Furthermore, we can also envision the combination of solutions that provide control-flow integrity with those that provide dataflow and noncontrol data integrity. Analogously, we can consider the combination of solutions tackling different goals, such as system integrity and confidentiality. Obviously, some of these combinations may increase the size of the TCB, whereas other combinations are not directly possible because they need to be integrated (e.g., in the hypervisor), which might require some effort to make them cooperate.

## 7. CONCLUSION

In this survey, we have provided a taxonomy of attacks in virtualized systems by considering the target at the different levels, the source and goals of the attackers, and also have shown the possible attack paths. We have then proposed a framework to categorize more clearly the threat models, which can be used to define precisely security and trust assumptions when proposing a new protection solution. This framework also includes the security properties of the solutions, such as their goal, for which we have proposed a taxonomy that considers the kind of attacks that they aim to prevent. We have also categorized the solutions using their implementation strategy, such as with a set of methodologies to protect the virtualized environment. We believe that the use of a common and standardized framework could ease the categorization, description, and evaluation of the security solutions for virtualized environments. In fact, we have witnessed, across papers, several different ways to express the threat model, and some assumptions are not always evident. Furthermore, by clearly defining the security properties of the solutions, we can compare different solutions with the same security properties using their implementation strategies. It also becomes easier to evaluate if different solutions can be combined to solve a complex problem.

## REFERENCES

Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. 2005. Control-flow integrity. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS'05)*. ACM, New York, NY, 340–353. DOI:http://dx.doi.org/10.1145/1102120.1102165

Advanced Micro Devices. 2005. AMD AMD64 virtualization codenamed "pacifica" technology. *Technology: Secure Virtual Machine Architecture Reference Manual* (2005), 1–124.

Aleph One. 1996. Smashing the stack for fun and profit. *Phrack Magazine* 7, 49, 14–16.

Ittai Anati, Shay Gueron, S. Johnson, and V. Scarlata. 2013. Innovative technology for CPU based attestation and sealing. *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP'13)*.

W. A. Arbaugh, D. J. Farber, and J. M. Smith. 1997. A secure and reliable bootstrap architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy (SP'97)*. IEEE Computer Society, Washington, DC, 65–. http://dl.acm.org/citation.cfm?id=882493.884371.

A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on* 1, 1, 11–33. DOI:http://dx.doi.org/10.1109/TDSC.2004.2

A. M. Azab, Peng Ning, E. C. Sezer, and Xiaolan Zhang. 2009. HIMA: A hypervisor-based integrity measurement agent. In *Computer Security Applications Conference, 2009 (ACSAC'09). Annual*. 461 –470. DOI:http://dx.doi.org/10.1109/ACSAC.2009.50

Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang, and Nathan C. Skalsky. 2010. HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*. ACM, New York, NY, 38–49. DOI:http://dx.doi.org/10.1145/1866307.1866313

Ahmed M. Azab, Peng Ning, and Xiaolan Zhang. 2011. SICE: A hardware-level strongly isolated computing environment for x86 multi-core platforms. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM, New York, NY, 375–388. DOI:http://dx.doi.org/10.1145/2046707.2046752

Sina Bahram, Xuxian Jiang, Zhi Wang, Mike Grace, Jinku Li, Deepa Srinivasan, Junghwan Rhee, and Dongyan Xu. 2010. DKSM: Subverting virtual machine introspection for fun and profit. In *Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems (SRDS'10)*. IEEE Computer Society, Washington, DC, 82–91. DOI:http://dx.doi.org/10.1109/SRDS.2010.39

Fabrizio Baiardi, Diego Cilea, Daniele Sgandurra, and Francesco Ceccarelli. 2009. Measuring semantic integrity for remote attestation. In *Trusted Computing, Second International Conference, Trust 2009, Oxford, UK, April 6–8, 2009, Proceedings (Lecture Notes in Computer Science)*, Vol. 5471. Springer, Berlin, 81–100.

Sundeep Bajikar. 2002. Trusted platform module (TPM) based security on notebook PCS-white paper. *White Paper, Mobile Platforms Group–Intel Corporation* 20.

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.* 37, 5, 164–177. DOI:http://dx.doi.org/10.1145/1165389.945462

Andrew Baumann, Marcus Peinado, and Galen Hunt. 2014. Shielding applications from an untrusted cloud with haven. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, 267–283. http://dl.acm.org/citation.cfm?id=2685048.2685070.

Muli Ben-Yehuda, Michael D. Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, and Ben-Ami Yassour. 2010. The turtles project: Design and implementation of nested virtualization. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, 1–6. http://dl.acm.org/citation.cfm?id=1924943.1924973.

S. Bleikertz, T. Mastelic, S. Pape, W. Pieters, and T. Dimkov. 2013. Defining the cloud battlefield—supporting security assessments by cloud customers. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*. 78–87. DOI:http://dx.doi.org/10.1109/IC2E.2013.31

Sara Bouchenak, Gregory Chockler, Hana Chockler, Gabriela Gheorghe, Nuno Santos, and Alexander Shraer. 2013. Verifying cloud services: Present and future. *SIGOPS Oper. Syst. Rev.* 47, 2, 6–19. DOI:http://dx.doi.org/10.1145/2506164.2506167

D. Bruschi, L. Cavallaro, A. Lanzi, and M. Monga. 2005. Replay attack in TCG specification and solution. In *Computer Security Applications Conference, 21st Annual*. 11, pp.–137. DOI:http://dx.doi.org/10.1109/CSAC.2005.47

Shakeel Butt, H. Andrés Lagar-Cavilla, Abhinav Srivastava, and Vinod Ganapathy. 2012. Self-service cloud computing. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12)*. ACM, New York, NY, 253–264. DOI:http://dx.doi.org/10.1145/2382196.2382226

John Butterworth, Corey Kallenberg, Xeno Kovah, and Amy Herzog. 2013. BIOS chronomancy: Fixing the core root of trust for measurement. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*. ACM, New York, NY, USA, 25–36. DOI:http://dx.doi.org/10.1145/2508859.2516714

Martim Carbone, Matthew Conover, Bruce Montague, and Wenke Lee. 2012. Secure and robust monitoring of virtual machines through guest-assisted introspection. In *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses (RAID'12)*. Springer, Berlin, 22–41. DOI:http://dx.doi.org/10.1007/978-3-642-33338-5_2

Martim Carbone, Diego Zamboni, and Wenke Lee. 2008. Taming virtualization. *IEEE Security and Privacy* 6, 1, 65–67. DOI:http://dx.doi.org/10.1109/MSP.2008.24

Matthew Carpenter, Tom Liston, and Ed Skoudis. 2007. Hiding virtualization from attackers and malware. *IEEE Security and Privacy* 5, 3, 62–65. DOI:http://dx.doi.org/10.1109/MSP.2007.63

R. S. Chakraborty and S. Bhunia. 2009. HARPOON: An obfuscation-based SoC design methodology for hardware protection. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 28, 10, 1493–1502. DOI:http://dx.doi.org/10.1109/TCAD.2009.2028166

Rajat Subhra Chakraborty and Swarup Bhunia. 2010. RTL hardware IP protection using key-based control and data flow obfuscation. In *Proceedings of the 2010 23rd International Conference on VLSI Design (VLSID'10)*. IEEE Computer Society, Washington, DC, 405–410. DOI:http://dx.doi.org/10.1109/VLSI.Design.2010.54

D. Champagne and R. B. Lee. 2010. Scalable architectural support for trusted software. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. 1–12. DOI:http://dx.doi.org/10.1109/HPCA.2010.5416657

Stephen Checkoway and Hovav Shacham. 2013. Iago attacks: Why the system call API is a bad untrusted RPC interface. *SIGPLAN Not.* 48, 4, 253–264. DOI:http://dx.doi.org/10.1145/2499368.2451145

Haibo Chen, Fengzhe Zhang, Cheng Chen, Rong Chen, Binyu Zang, Pen-chung Yew, and Wenbo Mao. 2007. Tamper-Resistant Execution in an Untrusted Operating System Using A Virtual Machine Monitor. Parallel Processing Institute Technical Report, Number: FDUPPITR-2007-0801, Fudan University, August, 2007.

Peter M. Chen and Brian D. Noble. 2001. When virtual is better than real. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HOTOS'01)*. IEEE Computer Society, Washington, DC, 133–. http://dl.acm.org/citation.cfm?id=874075.876409.

Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. 2005. Non-control-data attacks are realistic threats. In *Proceedings of the 14th Conference on USENIX Security Symposium, Volume 14 (SSYM'05)*. USENIX Association, Berkeley, CA, 12–12. http://dl.acm.org/citation.cfm?id=1251398.1251410.

Xiaoxin Chen, Tal Garfinkel, E. Christopher Lewis, Pratap Subrahmanyam, Carl A. Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan R. K. Ports. 2008. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. *SIGPLAN Not.* 43, 3, 2–13. DOI:http://dx.doi.org/10.1145/1353536.1346284

Yu-Yuan Chen, Pramod A. Jamkhedkar, and Ruby B. Lee. 2012. A software-hardware architecture for self-protecting data. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12)*. ACM, New York, NY, 14–27. DOI:http://dx.doi.org/10.1145/2382196.2382201

Mihai Christodorescu, Reiner Sailer, Douglas Lee Schales, Daniele Sgandurra, and Diego Zamboni. 2009. Cloud security is not (just) virtualization security: A short paper. In *CCSW'09: Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. ACM, New York, NY, 97–102. DOI:http://dx.doi.org/10.1145/1655008.1266852

Robert J. Creasy. 1981. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development* 25, 5, 483–490.

John Criswell, Nathan Dautenhahn, and Vikram Adve. 2014. KCoFI: Complete control-flow integrity for commodity operating system kernels. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP'14)*. IEEE Computer Society, Washington, DC, 292–307. DOI:http://dx.doi.org/10.1109/SP.2014.26

Weiqi Dai, Hai Jin, Deqing Zou, Shouhuai Xu, Weide Zheng, and Lei Shi. 2010. TEE: A virtual DRTM based execution environment for secure cloud-end computing. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS'10)*. ACM, New York, NY, 663–665. DOI:http://dx.doi.org/10.1145/1866307.1866390

D. A. Dai Zovi. 2006. Hardware virtualization rootkits. *BlackHat Briefings USA, August.*

Avinash R. Desai, Michael S. Hsiao, Chao Wang, Leyla Nazhandali, and Simin Hall. 2013. Interlocking obfuscation for anti-tamper hardware. In *Proceedings of the 8th Annual Cyber Security and Information Intelligence Research Workshop (CSIIRW'13)*. ACM, New York, NY, Article 8, 4 pages. DOI:http://dx.doi.org/10.1145/2459976.2459985

Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. 2008. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, New York, NY, 51–62. DOI:http://dx.doi.org/10.1145/1455770.1455779

Baozeng Ding, Yeping He, Yanjun Wu, and Yuqi Lin. 2013a. HyperVerify: A VM-assisted architecture for monitoring hypervisor non-control data. In *Software Security and Reliability-Companion (SERE-C'13), 2013 IEEE 7th International Conference on*. 26–34. DOI:http://dx.doi.org/10.1109/SERE-C.2013.20

Baozeng Ding, Yeping He, Yanjun Wu, and Jiageng Yu. 2013b. Systemic threats to hypervisor non-control data. *Information Security, IET* 7, 4, 349–354. DOI:http://dx.doi.org/10.1049/iet-ifs.2012.0252

B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and Wenke Lee. 2011. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *Security and Privacy (SP'11), 2011 IEEE Symposium on*. 297–312. DOI:http://dx.doi.org/10.1109/SP.2011.11

Joan G. Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendert van Doorn, Sean W. Smith, and Steve Weingart. 2001. Building the IBM 4758 secure coprocessor. *Computer* 34, 10, 57–66. DOI:http://dx.doi.org/10.1109/2.955100

Karim Eldefrawy, Aurélien Francillon, Daniele Perito, and Gene Tsudik. 2012. SMART: Secure and minimal architecture for (establishing a dynamic) root of trust. In *NDSS 2012, 19th Annual Network and Distributed System Security Symposium, February 5–8, San Diego, CA*. http://www.eurecom.fr/publication/3536.

Nelson Elhage. 2011. Virtunoid: Breaking out of KVM. In *Black Hat USA*, 2011.

Shawn Embleton, Sherri Sparks, and Cliff Zou. 2008. SMM rootkits: A new breed of OS independent malware. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm'08)*. ACM, New York, NY, Article 11, 12 pages. DOI:http://dx.doi.org/10.1145/1460877.1460892

Aristide Fattori, Andrea Lanzi, Davide Balzarotti, and Engin Kirda. 2015. Hypervisor-based malware protection with accessminer. *Computers & Security* 52, 0, 33–50. DOI:http://dx.doi.org/10.1016/j.cose.2015.03.007

P. Ferrie. 2006. *Attacks on Virtual Machine Emulators*. Technical Report. Symantec Security Response.

P. Ferrie. 2007. *Attacks on More Virtual Machine Emulators*. Technical Report. Symantec Security Response.

Jason Franklin, Mark Luk, Jonathan M. McCune, Arvind Seshadri, Adrian Perrig, and Leendert van Doorn. 2008a. Remote detection of virtual machine monitors with fuzzy benchmarking. *SIGOPS Oper. Syst. Rev.* 42, 3, 83–92. DOI:http://dx.doi.org/10.1145/1368506.1368518

Jason Franklin, Mark Luk, Jonathan M. McCune, Arvind Seshadri, Adrian Perrig, and Leendert van Doorn. 2008b. Towards sound detection of virtual machines. In *Botnet Detection. Advances in Information Security*, Vol. 36. Springer, Berlin, 89–116.

Yangchun Fu and Zhiqiang Lin. 2012. Space traveling across VM: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection. In *Security and Privacy (SP'12), 2012 IEEE Symposium on*. 586–600. DOI:http://dx.doi.org/10.1109/SP.2012.40

Tal Garfinkel, Keith Adams, Andrew Warfield, and Jason Franklin. 2007. Compatibility is not transparency: VMM detection myths and realities. In *Proceedings of the 11th Workshop on Hot Topics in Operating Systems (HotOS-XI'07)*.

Tal Garfinkel and Mendel Rosenblum. 2003. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the 2003 Network and Distributed System Symposium*.

Tal Garfinkel and Mendel Rosenblum. 2005. 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 (HOTOS'05)*. USENIX Association, Berkeley, CA, 20–20. http://dl.acm.org/citation.cfm?id=1251123.1251143.

Carl Gebhardt, Chris I. Dalton, and Allan Tomlinson. 2010. Separating hypervisor trusted computing base supported by hardware. In *Proceedings of the 5th ACM Workshop on Scalable Trusted Computing (STC'10)*. ACM, New York, NY, 79–84. DOI:http://dx.doi.org/10.1145/1867635.1867648

M. Godfrey and M. Zulkernine. 2014. Preventing cache-based side-channel attacks in a cloud environment. *Cloud Computing, IEEE Transactions on* 2, 4, 395–408. DOI:http://dx.doi.org/10.1109/TCC.2014.2358236

R. P. Goldberg. 1973. Architecture of virtual machines. In *Proceedings of the Workshop on Virtual Computer Systems*. ACM Press, New York, NY, 74–112. DOI:http://dx.doi.org/10.1145/800122.803950

R. P. Goldberg. 1974. Survey of virtual machine research. *IEEE Computer* 7, 6, 34–45.

James Greene. 2012. Intel Trusted Execution Technology—Hardware-based Technology for Enhancing Server Platform Security. (2012). Retrieved December 24, 2015 from http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf.

Brian Hay and Kara Nance. 2008. Forensics examination of volatile system data using virtual introspection. *SIGOPS Oper. Syst. Rev.* 42, 3, 74–82. DOI:http://dx.doi.org/10.1145/1368506.1368517

Owen S. Hofmann, Alan M. Dunn, Sangman Kim, Indrajit Roy, and Emmett Witchel. 2011. Ensuring operating system kernel integrity with OSck. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*. ACM, New York, NY, 279–290. DOI:http://dx.doi.org/10.1145/1950365.1950398

Owen S. Hofmann, Sangman Kim, Alan M. Dunn, Michael Z. Lee, and Emmett Witchel. 2013. InkTag: Secure applications on an untrusted operating system. *SIGPLAN Not.* 48, 4, 265–278. DOI:http://dx.doi.org/10.1145/2499368.2451146

Greg Hoglund and James Butler. 2006. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, Indianapolis, IN.

Trammell Hudson and Larry Rudolph. 2015. Thunderstrike: EFI firmware bootkits for Apple MacBooks. In *Proceedings of the 8th ACM International Systems and Storage Conference (SYSTOR'15)*. ACM, New York, NY, Article 15, 10 pages. DOI:http://dx.doi.org/10.1145/2757667.2757673

Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2015. S$A: A shared cache attack that works across cores and defies VM sandboxing and its application to AES. In *36th IEEE Symposium on Security and Privacy (S&P'15)*.

Gorka Irazoqui, MehmetSinan Inci, Thomas Eisenbarth, and Berk Sunar. 2014. Wait a minute! a fast, cross-VM attack on AES. In *Research in Attacks, Intrusions and Defenses*, Angelos Stavrou, Herbert Bos, and Georgios Portokalidis (Eds.). *Lecture Notes in Computer Science*, Vol. 8688. Springer, Berlin, 299–319. DOI:http://dx.doi.org/10.1007/978-3-319-11379-1_15

Trent Jaeger, Reiner Sailer, and Umesh Shankar. 2006. PRIMA: Policy-reduced integrity measurement architecture. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT'06)*. ACM, New York, NY, 19–28. DOI:http://dx.doi.org/10.1145/1133058.1133063

P. Jamkhedkar, J. Szefer, D. Perez-Botero, Tianwei Zhang, G. Triolo, and R. B. Lee. 2013. A framework for realizing security on demand in cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, Vol. 1. 371–378. DOI:http://dx.doi.org/10.1109/CloudCom.2013.55

Ramya Jayaram Masti, Claudio Marforio, and Srdjan Capkun. 2013. An architecture for concurrent execution of secure environments in clouds. In *Proceedings of the 2013 ACM Workshop on Cloud Computing Security Workshop (CCSW'13)*. ACM, New York, NY, 11–22. DOI:http://dx.doi.org/10.1145/2517488.2517489

Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. 2007. Stealthy malware detection through VMM-based "out-of-the-box" semantic view reconstruction. In *CCS'07: Proceedings of the 14th ACM Conference on Computer and Communications Security*. ACM, New York, NY, 128–138. DOI:http://dx.doi.org/10.1145/1315245.1315262

Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2008. VMM-based hidden process detection and identification using lycosid. In *VEE'08: Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. ACM, New York, NY, 91–100. DOI:http://dx.doi.org/10.1145/1346256.1346269

Bernhard Kauer. 2007. OSLO: Improving the security of trusted computing. In *Proceedings of the 16th USENIX Security Symposium on USENIX Security Symposium (SS'07)*. USENIX Association, Berkeley, CA, Article 16, 9 pages. http://dl.acm.org/citation.cfm?id=1362903.1362919

Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. 2010. NoHype: Virtualized cloud infrastructure without the virtualization. *SIGARCH Comput. Archit. News* 38, 3, 350–361. DOI:http://dx.doi.org/10.1145/1816038.1816010

Chung Hwan Kim, Sungjin Park, Junghwan Rhee, Jong-Jin Won, Taisook Han, and Dongyan Xu. 2015. CAFE: A virtualization-based approach to protecting sensitive cloud application logic confidentiality. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS'15)*. ACM, New York, NY, 651–656. DOI:http://dx.doi.org/10.1145/2714576.2714594

Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. 2012. STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud. In *Presented as Part of the 21st USENIX Security Symposium (USENIX Security 12)*. USENIX, Bellevue, WA, 189–204. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/kim.

S. T. King and P. M. Chen. 2006. SubVirt: Implementing malware with virtual machines. In *Security and Privacy, 2006 IEEE Symposium on*. 14 pp.–327. DOI:http://dx.doi.org/10.1109/SP.2006.38

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. 2009. seL4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*. ACM, New York, NY, 207–220. DOI:http://dx.doi.org/10.1145/1629575.1629596

Kostya Kortchinsky. 2009. Cloudburst: Hacking 3D (and breaking out of VMware). In *Black Hat Technical Security Conference*. USA, 2009.

Marc Lacoste. 2013. What does the future hold for hypervisor security? In *Trustworthy Clouds*, 2013. Available at https://www.zurich.ibm.com/~cca/tclouds13/abstracts/what_does_future.pdf.

J. F. Levine, J. B. Grizzard, and Henry L. Owen. 2006. Detecting and categorizing kernel-level rootkits to aid future detection. *Security Privacy, IEEE* 4, 1, 24–32. DOI:http://dx.doi.org/10.1109/MSP.2006.11

Min Li, Wanyu Zang, Kun Bai, Meng Yu, and Peng Liu. 2013. MyCloud: Supporting user-configured privacy protection in cloud computing. In *Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC'13)*. ACM, New York, NY, 59–68. DOI:http://dx.doi.org/10.1145/2523649.2523680

Lionel Litty and David Lie. 2006. Manitou: A layer-below approach to fighting malware. In *ASID'06: Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability*. ACM, New York, NY, 6–11. DOI:http://dx.doi.org/10.1145/1181309.1181311

Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *36th IEEE Symposium on Security and Privacy (S&P'15)*.

Flavio Lombardi and Roberto Di Pietro. 2009. KvmSec: A security extension for Linux kernel virtual machines. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC'09)*. ACM, New York, NY, 2029–2034. DOI:http://dx.doi.org/10.1145/1529282.1529733

J. M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, A. Datta, V. Gligor, and A. Perrig. 2010. TrustVisor: Efficient TCB reduction and attestation. In *Security and Privacy (S&P'10), 2010 IEEE Symposium on*. 143–158. DOI:http://dx.doi.org/10.1109/SP.2010.17

Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. 2008. Flicker: An execution infrastructure for TCB minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008 (Eurosys'08)*. ACM, New York, NY, 315–328. DOI:http://dx.doi.org/10.1145/1352592.1352625

Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. 2013. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP'13)*. ACM, New York, NY, Article 10, 1 page. DOI:http://dx.doi.org/10.1145/2487726.2488368

Peter M. Mell and Timothy Grance. 2011. *SP 800-145. The NIST Definition of Cloud Computing*. Technical Report. National Institute of Standards & Technology, Gaithersburg, MD, United States.

Derek Gordon Murray, Grzegorz Milos, and Steven Hand. 2008. Improving Xen security through disaggregation. In *Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'08)*. ACM, New York, NY, 151–160. DOI:http://dx.doi.org/10.1145/1346256.1346278

Emmanuel Owusu, Jorge Guajardo, Jonathan McCune, Jim Newsome, Adrian Perrig, and Amit Vasudevan. 2013. OASIS: On achieving a sanctuary for integrity and secrecy on untrusted platforms. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*. ACM, New York, NY, 13–24. DOI:http://dx.doi.org/10.1145/2508859.2516678

Bryan D. Payne, Martim Carbone, and Wenke Lee. 2007. Secure and flexible monitoring of virtual machines. *Computer Security Applications Conference, Annual* 0, 385–397. DOI:http://dx.doi.org/10.1109/ACSAC.2007.10

Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP'13)*. IEEE Computer Society, Washington, DC, 238–252. DOI:http://dx.doi.org/10.1109/SP.2013.47

Michael Pearce, Sherali Zeadally, and Ray Hunt. 2013. Virtualization: Issues, security threats, and solutions. *ACM Comput. Surv.* 45, 2, Article 17, 39 pages. DOI:http://dx.doi.org/10.1145/2431211.2431216

Gábor Pék, Levente Buttyán, and Boldizsár Bencsáth. 2013. A survey of security issues in hardware virtualization. *ACM Comput. Surv.* 45, 3, Article 40, 34 pages. DOI:http://dx.doi.org/10.1145/2480741.2480757

Gábor Pék, Andrea Lanzi, Abhinav Srivastava, Davide Balzarotti, Aurélien Francillon, and Christoph Neumann. 2014. On the feasibility of software attacks on commodity virtual machine monitors via direct device assignment. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS'14)*. ACM, New York, NY, 305–316. DOI:http://dx.doi.org/10.1145/2590296.2590299

Diego Perez-Botero, Jakub Szefer, and Ruby B. Lee. 2013. Characterizing hypervisor vulnerabilities in cloud computing servers. In *Proceedings of the 2013 International Workshop on Security in Cloud Computing (Cloud Computing'13)*. ACM, New York, NY, 3–10. DOI:http://dx.doi.org/10.1145/2484402.2484406

Daniele Perito and Gene Tsudik. 2010. Secure code update for embedded devices via proofs of secure erasure. In *Computer Security ESORICS 2010*, Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou (Eds.). *Lecture Notes in Computer Science*, Vol. 6345. Springer, Berlin, 643–662. DOI:http://dx.doi.org/10.1007/978-3-642-15497-3_39

Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh. 2004. Copilot—a coprocessor-based kernel runtime integrity monitor. In *Proceedings of the 13th Conference on USENIX Security Symposium, Volume 13 (SSYM'04)*. USENIX Association, Berkeley, CA, 13–13. http://dl.acm.org/citation.cfm?id=1251375.1251388.

Nick L. Petroni, Jr. and Michael Hicks. 2007. Automated detection of persistent kernel control-flow attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)*. ACM, New York, NY, 103–115. DOI:http://dx.doi.org/10.1145/1315245.1315260

Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, David Eyers, Brian Shand, Ruediger Kapitza, and Peter Pietzuch. 2014. CloudSafetyNet: Detecting data leakage between cloud tenants. In *ACM Cloud Computing Security Workshop (CCSW'14)*. ACM, ACM, Scottsdale, AZ.

Danny Quist and Val Smith. 2006. *Detecting the Presence of Virtual Machines Using the Local Data Table*. Technical Report. Offensive Computing.

Nguyen Anh Quynh and Yoshiyasu Takefuji. 2007. Towards a tamper-resistant kernel rootkit detector. In *SAC'07: Proceedings of the 2007 ACM Symposium on Applied Computing*. ACM, New York, NY, 276–283. DOI:http://dx.doi.org/10.1145/1244002.1244070

Thomas Raffetseder, Christopher Krügel, and Engin Kirda. 2007. Detecting system emulators. In *Information Security, 10th International Conference (ISC'07), Valparaíso, Chile, October 9–12, 2007, Proceedings*, *Lecture Notes in Computer Science*, Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta (Eds.), Vol. 4779. Springer, Berlin, 1–18.

M. A. Rappa. 2004. The utility business model and the future of computing services. *IBM Syst. J.* 43, 1, 32–42. DOI:http://dx.doi.org/10.1147/sj.431.0032

Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*. ACM, New York, NY, 199–212. DOI:http://dx.doi.org/10.1145/1653662.1653687

Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. 2012. Return-oriented programming: Systems, languages, and applications. *ACM Trans. Inf. Syst. Secur.* 15, 1, Article 2, 34 pages. DOI:http://dx.doi.org/10.1145/2133375.2133377

Mendel Rosenblum. 2004. The reincarnation of virtual machines. *Queue* 2, 5, 34–40. DOI:http://dx.doi.org/10.1145/ 1016998.1017000

Joanna Rutkowska and Rafał Wojtczuk. 2008. Preventing and detecting Xen hypervisor subversions. In *Blackhat Briefings USA*, 2008.

Mark D. Ryan. 2013. Cloud computing security: The scientific challenge, and a survey of solutions. *Journal of Systems and Software* 86, 9, 2263–2268. DOI:http://dx.doi.org/10.1016/j.jss.2012.12.025

Alireza Saberi, Yangchun Fu, and Zhiqiang Lin. 2014. HYBRID-BRIDGE: Efficiently bridging the semantic gap in virtual machine introspection via decoupled execution and training memoization. In *Proceedings Network and Distributed Systems Security Symposium (NDSS'14)*.

J. Sahoo, S. Mohapatra, and R. Lath. 2010. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT'10), 2010 Second International Conference on*. 222–226. DOI:http://dx.doi.org/10.1109/ICCNT.2010.49

Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. 2004. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium, Volume 13 (SSYM'04)*. USENIX Association, Berkeley, CA, 16–16. http://dl.acm.org/citation.cfm?id=1251375.1251391.

Nuno Santos, Rodrigo Rodrigues, Krishna P. Gummadi, and Stefan Saroiu. 2012. Policy-sealed data: A new abstraction for building trusted cloud services. In *Proceedings of the 21st USENIX Conference on Security Symposium (Security 12)*. USENIX Association, Berkeley, CA, 10–10. http://dl.acm.org/citation.cfm?id=2362793.2362803.

Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *36th IEEE Symposium on Security and Privacy (S&P'15)*.

J. Seol, S. Jin, D. Lee, J. Huh, and S. Maeng. 2015. A trusted IaaS environment with hardware security module. *Services Computing, IEEE Transactions on* PP, 99, 1–1. DOI:http://dx.doi.org/10.1109/TSC.2015.2392099

Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. 2007. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP'07)*. ACM, New York, NY, 335–350. DOI:http://dx.doi.org/10.1145/1294261.1294294

Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. 2005. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. *SIGOPS Oper. Syst. Rev.* 39, 5, 1–16. DOI:http://dx.doi.org/10.1145/1095809.1095812

Monirul I. Sharif, Wenke Lee, Weidong Cui, and Andrea Lanzi. 2009. Secure in-VM monitoring using hardware virtualization. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09)*. ACM, New York, NY, 477–487. DOI:http://dx.doi.org/10.1145/1653662.1653720

Evan R. Sparks. 2007. *A Security Assessment of Trusted Platform Modules*. Technical Report TR2007-597. Dartmouth College, Computer Science, Hanover, NH. http://www.cs.dartmouth.edu/reports/TR2007-597.pdf.

Sherri Sparks and Jamie Butler. 2005. Shadow walker: Raising the bar for rootkit detection. *Black Hat Japan*), 504–533.

Deepa Srinivasan, Zhi Wang, Xuxian Jiang, and Dongyan Xu. 2011. Process Out-grafting: An efficient "out-of-VM" approach for fine-grained process execution monitoring. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM, New York, NY, 363–374. DOI:http://dx.doi.org/10.1145/2046707.2046751

Abhinav Srivastava, Himanshu Raj, Jonathon Giffin, and Paul England. 2012. Trusted VM snapshots in untrusted cloud infrastructure. In *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses (RAID'12)*. Springer, Berlin, 1–21. DOI:http://dx.doi.org/10.1007/978-3-642-33338-5_1

Udo Steinberg and Bernhard Kauer. 2010. NOVA: A microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys'10)*. ACM, New York, NY, 209–222. DOI:http://dx.doi.org/10.1145/1755913.1755935

Patrick Stewin and Iurii Bystrov. 2013. Understanding DMA malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Ulrich Flegel, Evangelos Markatos, and William Robertson (Eds.). *Lecture Notes in Computer Science*, Vol. 7591. Springer, Berlin, 21–41. DOI:http://dx.doi.org/10.1007/978-3-642-37300-8_2

Raoul Strackx and Frank Piessens. 2012. Fides: Selectively hardening software application components against kernel-level or process-level malware. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12)*. ACM, New York, NY, 2–13. DOI:http://dx.doi.org/10.1145/2382196.2382200

G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. 2003. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS'03)*. ACM, New York, NY, 160–171. DOI:http://dx.doi.org/10.1145/782814.782838

Jakub Szefer, Eric Keller, Ruby B. Lee, and Jennifer Rexford. 2011. Eliminating the hypervisor attack surface for a more secure cloud. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS'11)*. ACM, New York, NY, 401–412. DOI:http://dx.doi.org/10.1145/2046707.2046754

Jakub Szefer and Ruby B. Lee. 2012. Architectural support for hypervisor-secure virtualization. *SIGPLAN Not.* 47, 4, 437–450. DOI:http://dx.doi.org/10.1145/2248487.2151022

Alexander Tereshkin, Rafal Wojtczuk, Joanna Rutkowska. 2008. Bluepilling the Xen Hypervisor. Retrieved December 24, 2015 from http://invisiblethingslab.com/resources/bh08/part3.pdf. (Aug 2008).

Alexander Tereshkin and Rafal Wojtczuk. 2009. Introducing ring -3 rootkits. *Black Hat USA*, July 2009.

Hsin-Yi Tsai, M. Siebenhaar, A. Miede, Yu-Lun Huang, and R. Steinmetz. 2012. Threat as a service?: Virtualization's impact on cloud security. *IT Professional* 14, 1, 32–37. DOI:http://dx.doi.org/10.1109/MITP.2011.117

R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith. 2005a. Intel virtualization technology. *Computer* 38, 5, 48–56. DOI:http://dx.doi.org/10.1109/MC.2005.163

Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kagi, Felix H. Leung, and Larry Smith. 2005b. Intel virtualization technology. *Computer* 38, 5 (2005), 48–56.

Anjo Vahldiek, Eslam Elnikety, Aastha Mehta, Deepak Garg, Peter Druschel, Ansley Post, Rodrigo Rodrigues, and Johannes Gehrke. 2014. *Guardat: A Foundation for Policy-Protected Data*. Technical Report 014-002, MPI-SWS. MPI-SWS. Retrieved December 24, 2015 from http://www.mpi-sws.org/cont/tr/2014-002.pdf.

Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M. Swift. 2012. Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense). In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12)*. ACM, New York, NY, 281–292. DOI:http://dx.doi.org/10.1145/2382196.2382228

V. Vu, S. Setty, A.J. Blumberg, and M. Walfish. 2013. A hybrid architecture for interactive verifiable computation. In *Security and Privacy (SP'13), 2013 IEEE Symposium on*. 223–237. DOI:http://dx.doi.org/10.1109/SP.2013.48

Aurelien Wailly, Marc Lacoste, and Hervé DEBAR. 2012. KungFuVisor: Enabling hypervisor self-defense. In *EuroDW'12 : The 6th EuroSys Doctoral Workshop*. Bern, Switzerland. https://hal.archives-ouvertes.fr/hal-00738069.

Jiang Wang, Angelos Stavrou, and Anup Ghosh. 2010. HyperCheck: A hardware-assisted integrity monitor. In *Recent Advances in Intrusion Detection*, Somesh Jha, Robin Sommer, and Christian Kreibich (Eds.). *Lecture Notes in Computer Science*, Vol. 6307. Springer, Berlin, 158–177. DOI:http://dx.doi.org/10.1007/978-3-642-15512-3_9

Zhi Wang and Xuxian Jiang. 2010. HyperSafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy (SP'10)*. IEEE Computer Society, Washington, DC, 380–395. DOI:http://dx.doi.org/10.1109/SP.2010.30

Zhi Wang, Xuxian Jiang, Weidong Cui, and Peng Ning. 2009. Countering kernel rootkits with lightweight hook protection. In *CCS'09: Proceedings of the 16th ACM Conference on Computer and Communications Security*. ACM, New York, NY, 545–554. DOI:http://dx.doi.org/10.1145/1653662.1653728

Zhi Wang, Chiachih Wu, Michael Grace, and Xuxian Jiang. 2012. Isolating commodity hosted hypervisors with hyperlock. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys'12)*. ACM, New York, NY, 127–140. DOI:http://dx.doi.org/10.1145/2168836.2168850

Yuanfeng Wen, JongHyuk Lee, Ziyi Liu, Qingji Zheng, Weidong Shi, Shouhuai Xu, and Taeweon Suh. 2013. Multi-processor architectural support for protecting virtual machine privacy in untrusted cloud environment. In *Proceedings of the ACM International Conference on Computing Frontiers (CF'13)*. ACM, New York, NY, Article 25, 10 pages. DOI:http://dx.doi.org/10.1145/2482767.2482799

Rafal Wojtczuk and Joanna Rutkowska. 2009. Attacking SMM Memory via Intel CPU Cache Poisoning. In *Invisible Things Lab*, 2009. Available at: http://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf.

Rafal Wojtczuk and Joanna Rutkowska. 2011. Following the white rabbit: Software attacks against Intel VT-d technology. ITL: http://invisiblethingslab.com/resources/2011/Software%20Attacks%20on%20Intel%20VT-d.pdf.

Chiachih Wu, Zhi Wang, and Xuxian Jiang. 2013. Taming hosted hypervisors with (mostly) deprivileged execution. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'13)*. San Diego, CA.

Zhenyu Wu, Zhang Xu, and Haining Wang. 2012. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. USENIX, Bellevue, WA, 159–173. Retrieved December 24, 2015 from https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/wu.

Yubin Xia, Yutao Liu, and Haibo Chen. 2013. Architecture support for guest-transparent VM protection from untrusted hypervisor and physical attacks. In *High Performance Computer Architecture (HPCA'13), 2013 IEEE 19th International Symposium on*. 246–257. DOI: http://dx.doi.org/10.1109/HPCA.2013.6522323

Zhifeng Xiao and Yang Xiao. 2013. Security and privacy in cloud computing. *Communications Surveys Tutorials, IEEE* 15, 2, 843–859. DOI: http://dx.doi.org/10.1109/SURV.2012.060912.00182

Bin Xing, Zhen Han, Xiaolin Chang, and Jiqiang Liu. 2014. OB-IMA: Out-of-the-box integrity measurement approach for guest virtual machines. *Concurrency and Computation: Practice and Experience,* n/a–n/a. DOI: http://dx.doi.org/10.1002/cpe.3273

Huijun Xiong, Xinwen Zhang, Wei Zhu, and Danfeng Yao. 2012. CloudSeal: End-to-end content protection in cloud-based storage and delivery services. In *Security and Privacy in Communication Networks*, Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis (Eds.). *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Vol. 96. Springer, Berlin, 491–500. DOI: http://dx.doi.org/10.1007/978-3-642-31909-9_30

Xi Xiong, Donghai Tian, and Peng Liu. 2011. Practical protection of kernel integrity for commodity OS from untrusted extensions. In *NDSS*.

Min Xu, Xuxian Jiang, Ravi Sandhu, and Xinwen Zhang. 2007. Towards a VMM-based usage control framework for OS kernel integrity protection. In *SACMAT'07: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*. ACM Press, New York, NY, 71–80. DOI: http://dx.doi.org/10.1145/1266840.1266852

Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. 2011. An exploration of L2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop (CCSW'11)*. ACM, New York, NY, 29–40. DOI: http://dx.doi.org/10.1145/2046660.2046670

Jisoo Yang and Kang G. Shin. 2008. Using hypervisor to provide data secrecy for user applications on a per-page basis. In *Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'08)*. ACM, New York, NY, 71–80. DOI: http://dx.doi.org/10.1145/1346256.1346267

Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. 2011a. CloudVisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*. ACM, New York, NY, 203–216. DOI: http://dx.doi.org/10.1145/2043556.2043576

Fengwei Zhang, Jiang Wang, Kun Sun, and Angelos Stavrou. 2014. HyperCheck: A hardware-assisted integrity monitor. *IEEE Transactions on Dependable and Secure Computing* 11, 4, 332–344. DOI: http://dx.doi.org/10.1109/TDSC.2013.53

Xiaolan Zhang, Leendert van Doorn, Trent Jaeger, Ronald Perez, and Reiner Sailer. 2002. Secure coprocessor-based intrusion detection. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop (EW 10)*. ACM, New York, NY, 239–242. DOI: http://dx.doi.org/10.1145/1133373.1133423

Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. 2011b. HomeAlone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy (SP'11)*. IEEE Computer Society, Washington, DC, 313–328. DOI: http://dx.doi.org/10.1109/SP.2011.31

Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS'12)*. ACM, New York, NY, 305–316. DOI: http://dx.doi.org/10.1145/2382196.2382230

Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2014. Cross-tenant side-channel attacks in PaaS clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*. ACM, New York, NY, 990–1003. DOI: http://dx.doi.org/10.1145/2660267.2660356

Received February 2015; revised October 2015; accepted November 2015