Security Issues and Challenges for Virtualization Technologies

AUTHOR1, AUTHOR2, ... AUTHOR N, place
FEDERICO SIERRA-ARRIAGA, Intel Corporation and Universidad Autónoma de Guadalajara, México
RODRIGO BRANCO, Oregon State University, School of Electrical Engineering and Computer Science, USA
BEN LEE, Oregon State University, School of Electrical Engineering and Computer Science, USA

Virtualization-based technologies have become ubiquitous in computing. While they provide an easy to implement platform for scalable, high-availability services, they also introduce new security issues. Traditionally, discussions on security vulnerabilities in server platforms have been focused on standalone (i.e., non-virtualized) environments. For cloud and virtualized platforms, the discussion focuses on the shared usage of resources and the lack of control over the infrastructure. However, the impact virtualization technologies can have on exploit mitigation mechanisms of host machines is often neglected. Therefore, this survey discusses the following issues: First, the security issues and challenges that are introduced by the migration from standalone solutions to virtualized environments. A special attention is given to the Virtual Machine Monitor, since it is a core component in a virtualized solution. Second, the impact (sometimes negative) that these new technologies have on existing security strategies for hosts. Third, how virtualization technologies can be leveraged to provide new security mechanisms not previously available. Finally, how virtualization technologies can be used for malicious purposes.

CCS Concepts: • **Security and privacy → Virtualization and security**.

Additional Key Words and Phrases: Virtual Machine Monitor, cloud computing, security vulnerabilities, virtualization survey, hypervisor.

ACM Reference Format:
Author1, author2, ... author n, Federico Sierra-Arriaga, Rodrigo Branco, and Ben Lee. 2020. Security Issues and Challenges for Virtualization Technologies. *ACM Comput. Surv.* 1, 1, Article 1 (January 2020), 37 pages. https://doi.org/10.1145/3382190

1 INTRODUCTION

Virtualization technologies have their origins from the 1960's [40]. However, they only became available to the general public during the late 1990's and early 2000's when the first commercial virtualization solutions for the x86 architecture started gaining popularity and Internet speeds allowed for the launch of cloud computing platforms, such as Salesforce.com in 1999 and Amazon Web Services in 2002, and public virtualization services such as Amazon EC2 in 2006 [92]. The cloud

Authors' addresses: Author1, author2, ... author n, place, street, town, state, 5 positive integers, email1@domain1.com,..., emailn@domainn.com; Federico Sierra-Arriaga, Intel Corporation, Universidad Autónoma de Guadalajara, México; Rodrigo Branco, Oregon State University, School of Electrical Engineering and Computer Science, 1048 Kelley Engineering Center, Corvallis, OR, 97331, USA; Ben Lee, Oregon State University, School of Electrical Engineering and Computer Science, Corvallis, OR, 97331, USA, Emails:federico.sierra.arriaga@intel.com,rodrigo@kernelhacking.com,benl@engr.orst.edu.

0360-0300/2020/1-ART1 $15.00
https://doi.org/10.1145/3382190

computing market has experienced continuous growth without any indication of slowing down [22] and virtualization technologies are critical for cloud service models such as Infrastructure as a Service (IaaS).

The adoption of virtualization has created a new set of security considerations:

- In the case of cloud platforms, different customers may end up sharing a single physical host increasing the importance of *isolation*. That is, data confidentiality of a guest *Virtual Machine* (VM) needs to be protected against attacks initiated by another guest in the same physical machine. Likewise, a Denial of Service (DoS) attack initiated by a VM against the physical host must be properly mitigated since such an attack might impact all the VMs in the server.
- Most virtualization solutions have at least one important software component – the *Virtual Machine Monitor* (VMM), which can be affected by software bugs that lead to security vulnerabilities.
- New forms of malware that target virtualization have appeared and, since they operate with a higher level of privilege than the operating system (OS) [129] [167], they cannot be suppressed by traditional techniques like antivirus software.
- Some of the traditional security strategies for standalone systems are weakened or rendered useless when virtualization is introduced. Traditional security strategies may work under assumptions that aren't necessarily true for virtualized systems (i.e., a VM can share a physical host with another VM, which can be malicious).

Motivated by these issues, this paper surveys security issues in virtualization technologies, including new hardware capabilities to mitigate threats. Since the term *virtualization* is used in different contexts, we want to be specific on the use of the term as applied to this work: System-level virtualization that partitions and isolates a hardware system for running commodity operating systems. The objective of the survey is to provide the state-of-the-art on security issues related to virtualization by giving some real examples of security problems unveiled in the past, defining the technologies developed to overcome those problems, and how/where they are applied. The focus is on defining the problems (past mistakes) and challenges (on mitigating systemic issues, but not necessarily on mitigating simple case scenarios in which a simple patch would fix a code error).

Section 2 gives a quick review of the most common VMM architectures and discusses their differences with alternative approaches to resource partition and isolation. Section 3 uses a classification of the most common attacks to a VMM platform. Section 4 gives concrete examples of such attacks.

Since virtualization adds a new layer of complexity to existing platforms, Section 5 discusses how pre-existing security mechanisms are affected by the introduction of this technology. Section 6 discusses how virtualization can be used to improve security of existing systems in the context of the security triad (Confidentiality, Integrity and Availability) and Section 7 discusses how malicious actors can take advantage of virtualization technologies to design and implement new types of attacks. Before the survey concludes in Section 9, it also discusses (in Section 8) the future trends and gaps related in security research related to virtualization.

2 BACKGROUND

A VMM, also called *hypervisor*, is a dedicated component typically implemented in software that manages the host resources and makes them available to a set of VMs. A VM is a logical partition of the resources of the host system and has the capability of executing an OS in isolation from the other VMs (although resources like I/O devices and some memory areas may be shared) [147]. Figure 1 shows the two basic architectures on which a VMM can interact with the resources of the host. The following definitions are taken from Robert P. Goldberg's seminal work on VMM architecture [56]:

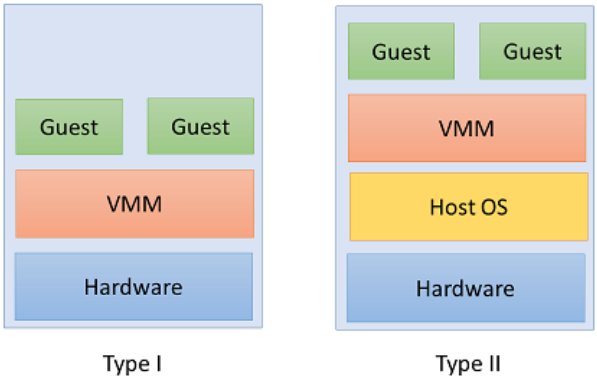

Fig. 1. VMM architectures. A Type I VMM runs directly on top of the physical hardware. A Type II VMM runs on top of an existing OS.

- Type I or bare metal – The VMM runs on a bare machine.
- Type II or hosted – The VMM runs on an *extended host*, under the host operating system.

Both types of VMMs handle the creation of a VM, but Type I VMMs are additionally directly in charge of allocating and scheduling the physical host resources. Examples of Type I VMMs are VMware ESXi [151] and The Xen Project [111].

A Type II VMM, on the other hand, runs as part of an extended host and the OS performs its usual allocating and scheduling of resources. Examples of Type II VMMs include Oracle VM VirtualBox [101], VMware Workstation [152] and KVM (which is packaged as a Linux kernel module and "uses Linux scheduler and memory management") [108] [16].

The following two subsections discuss the functionalities provided by VMM and other means of providing isolation as alternatives to using virtualization.

2.1 VMM Functionalities

To perform any security analysis on VMMs, the scope of their functions must be well defined and understood. Based on the NIST definition [18], the following VMM functions are identified:

- *Execution isolation of the VMs* – The VMM must be able to handle simultaneous execution of one or more guest VMs (or at least make their execution *appear* simultaneous). These VMs will be using the same set of resources (i.e., memory, CPU, I/O devices, etc.). Therefore, the VMM needs to provide each VM with an isolated execution environment to prevent one VM's behavior from affecting the others.
- *Devices emulation and access control* – The VMM is responsible for presenting the guest VMs with an abstraction for each device "decoupled from its physical implementation" [153]. Each VM must be able to use a device as if it had exclusive ownership without interfering with the other VMs. Some existing approaches allow a specific VM to have complete control over a device without a virtualization layer in-between, which is referred to as *hardware pass-through*, but this introduces new security considerations that will be discussed in Section 4.2.
- *Execution of privileged operations by the VMM for Guest VMs* – Some operations that are available to the OS (privileged mode) in a standalone architecture, e.g., the `LGDT` (*Load Global Descriptor Table register*) instruction, are not allowed to be performed by guest VMs in a

virtualized environment due to security concerns. These operations are instead executed by the VMM on behalf of guest VMs.
- *Management of VMs* – The VMM must provide a mechanism for management of guest VMs. This includes guest creation, assignment of host resources, monitoring, relocation, and maintenance. Most VMMs provide an interface to perform these tasks. Some advanced features such as snapshots and backups are also provided.
- *Administration of VMM platform and software* – The VMM controls the interaction between its software and the host, usually through a web interface or a virtual console.

Modern hardware provides extensions to facilitate the implementation of the different functionalities of a VMM. On Intel platforms, those extensions are called *Virtualization Technology for IA-32 and Intel 64 Processors* (VT-x), and on AMD, they are simply called *Virtualization (AMD-V)*. Solutions that rely on hardware extensions are often called *Hardware-assisted Virtual Machine* (HVM). Many platforms (and hypervisors) also support the concept of *nested virtualization*. In those systems, it is possible to have multiple layers of virtualization, which means that a hypervisor would run on top of another hypervisor, instead of directly on top of the hardware.

2.2 Alternative ways to provide isolation

VMMs are not the only technology that can provide isolation and partitioning of the resources of a host. Traditional OS-level sandboxes provide similar features, which are based on implementing or enhancing protections at the OS level instead of adding another layer to the platform. Although this survey focuses on VMMs, some implementations of this approach will be discussed in this subsection for completeness.

2.2.1 Chroot/Jails. In UNIX systems, the *chroot* operation changes the root directory of a process. This affects the directory resolution and prevents the process from writing to elements of the file system outside the directory tree passed as a parameter to the call. Although *chroot* provides a certain level of isolation, it was not designed as a security mechanism and it is not difficult to break [43]. *FreeBSD jails* adds to this mechanism by providing a separate partition and IP address for each jail. This mechanism is flexible enough to allow for partitions that behave like a complete system ("complete jails") or used only to run a service or a particular application ("service jails") [122]. Thus, FreeBSD jails are essentially security containers as explained next.

2.2.2 Containers. In a container-based solution, a single OS partitions the resources of a host and provides isolation among those partitions. This is shown in Figure 2. Implementations of this technology include Linux Containers (LXC) [63] and Microsoft Server Containers [126][106][1]. Note that containers isolate more than just the filesystem, as opposed to chroot/jails.

Container-based solutions have certain advantages over VMMs. Since they work at the OS-level, no additional layer is needed to manage the host resources. This results in less complexity and overhead than VMMs. However, sharing the OS kernel means that the provided isolation is not as strong as in virtualization-based environments. Therefore, solutions exist to limit the exposed/common areas per-container to minimize the impact of OS-level vulnerabilities [44].

2.2.3 Trusted Execution Environments. Trusted Execution Environments (TEEs) can be used in application domains where assets (such as hard-drive encryption keys and payment processor applications) need to be protected from a potentially malicious OS. The concept of an isolated processing area where sensitive information can be stored and trusted programs can be executed has been implemented independently by different vendors using software, hardware, or both. Figure 3

[1]Microsoft also offers Hyper-V containers, which are virtualization-based.

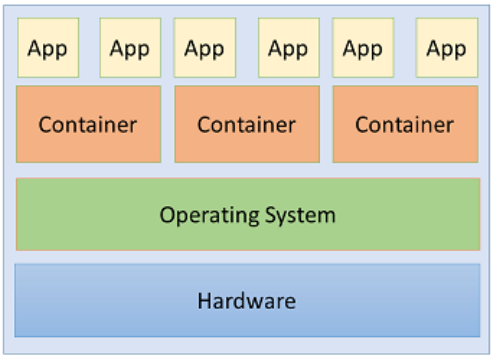

Fig. 2. Container-based isolation. An OS is shared among partitions and provides isolation between them.

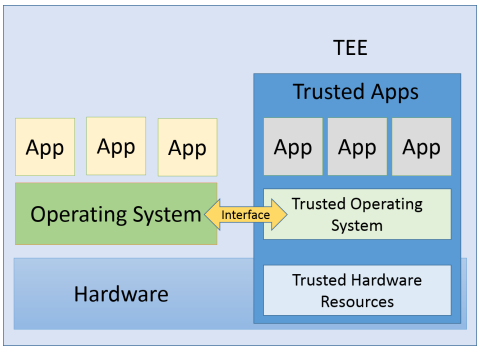

Fig. 3. Trusted Execution Environments. The resources of a system are not partitioned, but a processing area is isolated from the rest of the system. This area is used to store sensitive resources like data and code for execution. Access from outside to the sensitive assets can be controlled by a dedicated interface.

shows a simplified picture of a TEE, where the controlled interface provided by the OS allows a TEE to access resources outside of it's boundary. In order to facilitate adoption and interoperability, standards must be defined. GlobalPlatform is an association that is working towards standardization of TEEs [107]. Two examples of hardware architectures that support TEEs are ARM TrustZone [9], which implements the model described above, and Intel SGX (discussed below).

Intel Software Guard Extensions (SGX). This is a TEE that allows developers to create applications with the ability to protect specific data and code portions from high privilege software in the system (OS or VMM). This is especially relevant in cloud environments where the owner of the data is not necessarily the owner of the platform hardware and software. Secrets protection is accomplished by creating dedicated regions of memory called *enclaves* (where the secrets will reside), and setting up trusted channels to provision these secrets into them. It also provides with a hardware based attestation mechanism so applications can confirm that the enclaves used are legitimate. In order to use this technology, the system must have a CPU with SGX support and it also must be supported in the BIOS. An SDK is distributed by Intel in order to build SGX-extended applications using common compilers and tools [24].

Some TEE architectures are designed to suplement virtualization technologies to enhance their security: Secure Encrypted Virtualization (SEV) is a technology developed by AMD [3]. It implements cryptographic protections for the data associated to the guest VMs. For each guest, a tag is generated and an AES-128 key is created based on the tag. The key is generated using the AMD secure processor (AMD-SP) integrated into the SoC and it is never exposed. This key is used to encrypt the guests data in main memory. This prevents external entities (including the host administrator) from accessing data in plaintext form. The guest is allowed to decide whether all of its memory is protected or only a subset [77].

TEEs are a viable alternative to VMMs to provide isolation for security purposes, but the technology can also be prone to security vulnerabilities: The implementation of a complete Trusted Operating System (as described in the model) can be a source of bugs [124]. And the complete security model collapses if the secure boot chain is compromised [114].

3 CLASSIFICATION OF VMM SECURITY PROBLEMS

This section discusses a classification scheme for virtualization vulnerabilities from the perspective of their impact on the overall platform security. The security problems discussed in this survey are mostly specific to virtualized technologies although some are also present in standalone solutions, e.g., side-channels. Pék *et al.* proposed this classification based on *source and target for the attacks* [104].

3.1 Classification based on source and target for the attack

This classification is based on the attack path (i.e., its origin and destination) and it is very helpful in understanding the boundaries introduced by virtualization and how these boundaries are susceptible to security attacks.

- Guest to VMM – An attacker in control of a guest VM can launch an attack against the VMM. This attack is the most serious because the VMM is in charge of providing isolation for the whole environment. This type of attacks will become more common as virtualization becomes pervasive, and thus protecting against it will be crucial.
- Guest to Host – The host is the underlying OS that runs a VMM in a Type II hypervisor (see Figure 1). Some hypervisors also have a special, highly privileged guest VM (e.g., Dom0 in the Xen hypervisor). This type of attack involves a guest VM escaping the hypervisor and executing code in the host OS or as a highly privileged guest. The consequences of this attack are comparable to Guest to VMM attacks.
- Host to VMM – An attack on the host (not necessarily by a privileged user) may be used to gain control of the VMM. An example of attack vector can be a vunerable application running in the host and being exploited.
- Guest to Guest – An attack (from a guest VM) breaks the isolation provided by the VMM and is able to influence the behavior or has access to the assets of another guest VM under the control of the same VMM.
- Guest to Self and Host to Self – This is a vulnerability that permits the elevation of privileges in the same context (either as guest VM or host OS). These attacks predate virtualization technologies and they have been discussed at large in works related to standalone systems. However, they must also be taken into account in the context of virtualization because of the specific cases where they exploit vulnerabilities in the VMM.

A commonly used term for vulnerabilities that break guest VM isolation is *guest escape vulnerabilities*, which can be considered as another classification option. This classification groups the Guest to VMM, Guest to Host, and Guest to Guest categories from the list above.

3.2 Threat Model

Threat modeling is the process of enumerating threats that a given system (software, hardware, or the entire computing environment) is expected to protect against. Threat models can be as big as involving an entire company data (including partner relations, customers and others) or as specific as looking at a single component. This survey discusses different vulnerabilities that exist on different threat models. For instance, a guest operating system attacking a host operating system is only a security problem if the virtualization is considered to be providing a security boundary for that overall platform.

With a threat model so diverse (given the multitude of potential uses of virtualization), the survey tries to enumerate different classes of issues (instead of listing all cases of a given class). The intention behind that choice is to provide an entry point for each of the classes to be considered (which makes finding additional instances of a given class much easier).

For each of the security issues analyzed, the survey also includes the primitive obtained by the attacker (a primitive is a capability that an attacker exploiting a given issue gets access to - and that she did not have before - and is what is leveraged to cause the impact). An example of a primitive is an attacker exploiting a side-channel vulnerability to read memory areas that were protected otherwise. The same primitive (memory read) could be obtained by a code error in which a privileged entity (like a hypervisor) receives an un-trusted (i.e., a value coming from the guest) pointer and does not properly check it before dumping the values in memory this pointer refers to. While the underlying security issue is completely different, the exposed primitive is essentially the same. With a primitive, the attacker has many creative ways to increase the impact on the system's security (i.e., with the same memory read an attacker could try to target the passwords in memory and with that elevate her privileges on the system).

In a threat model, assets (i.e., what needs to be protected) and security objectives (i.e., what security policies the system enforces to its assets) must be defined. For a general purpose computing platform, in which different use-cases exist, it gets much harder. That is why for each of the issues discussed, the survey also explicitly mentions a scenario in which the issue is a security problem/concern. Notice that in different scenario/use-cases, that same issue might not be a problem at all.

During the threat modeling, a *trusted computing base* (TCB) is also defined, which lists the set of all elements that are trusted (i.e., they are not considered attack sources) by the system under analysis in order to meet its security objectives. By definition, anything that is considered part of the TCB must also meet the security objectives, because if compromised, it would have the same effect as compromising the elements that trust it.

The composition of the different components (and threat models) is the challenge in securing a system (or environment). Threats (also known as *adversaries*) must be identified as they are the source of potential attacks. Adversaries could be anything that is un-trusted for that specific system and need to be mitigated in order for it to achieve its security objectives, i.e., other devices in the platform, software running with high privileges, etc.

As a technology that is commonly used to share platform resources, hypervisors are often expected to provide security boundaries between different guest operating systems. Supporting infra-structure to manage multiple guest systems (and automate the creation of virtual-machines and the management of resources) add to complexity of analyzing security for virtualized environments.

4 EXAMPLES OF VMM SECURITY ISSUES AND ATTACKS

This section provides examples of virtualization related vulnerabilities that have occurred in the past. The discussion is based on the source and target for the attack categorization presented in

Section 3.1. The examples for categories *Guest to Host* and *Guest to VMM* are presented together since there can be some overlap among them.

As explained before, instead of trying to be comprehensive (which is an unfeasible task), this work lists examples in each category to facilitate the understanding of the security boundaries that are provided by a hypervisor (and how the boundaries fail in the presence of bugs). Modern secure software development practices minimize the existence of such issues, and modern software mitigations might help prevent the exploitation of specific cases, but overall the examples discussed do not have a single general mitigation. To facilitate the understanding of each of the cases listed, the exposed primitive is also explained (so it is possible to compare the cases with others that were not covered).

4.1 Guest to Host and Guest to VMM

These two categories cover vulnerabilities that allow guest VMs to compromise the host machine thereby escaping the hypervisor policies. Usually the primitive the attacker is after provides in the end an arbitrary code execution (in which she is able to execute anything with the privileges of the attacked component). Notice that the relation between a read primitive, a write primitive and an arbitrary code execution is many times transitional (i.e., a write primitive might lead to the full control over the execution flow of a program). The intention on listing different examples is to show the extent of the complexity of a modern hypervisor implementation, and to list usual code issues that could culminate in a security violation. The mitigation for the issues listed in this subsection are all simple software patches (general mitigations like those to prevent the exploitation of buffer overflows are not discussed in this survey since they are not specific to virtualization, but would commonly reduce the impact from arbitrary code execution to a denial of service and not much else).

A hypervisor intercepts certain instructions from guest VMs in order to emulate their functionalities. In Parallels Desktop[2], some assembly instructions erroneously cause a VMM abort, which can be used to perform DoS attacks (thus giving an attacker the ability to disrupt services that otherwise she would not be able to disrupt). While DoS is not a primitive per-se, the mishandling of the event caused an undesired impact. These include the `INT 0xAA` instruction that generates the software interrupt 0xAA, the `IRET` instruction that returns from an interrupt, the `MOVNTI` instruction that moves a double word from register to memory, and reading from or writing to `SEGR 6` and `7` (segment registers). A malicious guest can use any of the previously mentioned instructions to cause a VMM abort and bring down all the guests in the system [102].

Project QEMU (Quick EMUlator) is an open source VMM and CPU emulator. It can be used as standalone or with other solutions such as Xen. When used in conjunction with the Xen hypervisor, only the device emulation functionality of QEMU (called the *Device Model*) is used and the CPU virtualization part is handled by Xen [109].

QEMU version 0.8.2 used in Xen provided a Cirrus VGA extension that was vulnerable to various heap-based buffer overflows [102]. This functionality provides a graphics update command to change a console display geometry (using the `cirrus_invalidate_region()` function to mark regions as dirty during video-to-video copy operations). Since the supporting code runs on the Device Model process with VMM privileges, a successful exploitation of this vulnerability leads to a guest VM escape (i.e., achieving arbitrary code execution from the Guest to VMM).

In 2004, an emulated Floppy Disk Controller support was added to QEMU [53]. This controller supports a series of commands to be received through a queue, which is initialized to a size of 512

[2]In the original article, software vendor was not named since they did not respond on time for publication. In [99], the product name is included as Parallels Desktop.

bytes. Since the commands can have different number and types of parameters, a variable `data_pos` is used to keep track of the next position in the queue to be written. Most of the commands correctly reset this variable to 0 after processing, but in two of them, the reset operation is delayed due to the usage of callbacks with specific timers. This allowed the guest to continue writing to the queue before the delay ends overflowing the FIFO queue [54]. Note that hypervisors still support floppy disk emulation and VMs configured with this feature enabled likely still exist (even if the hypervisor and guest VMs have been updated). Although this feature supports something quite old, the vulnerability was only discovered in 2015 and many environments are still potentially affected due to the legacy configuration of guest VMs. It is common that structure overflows like this lead to arbitrary code execution primitives.

A more notorious case was caused by an insufficient boundary check in the VMware SVGA II emulated video device code, which allowed a guest VM to take control of the `vmware-vmx` host process [82]. This vulnerability was also confirmed in VMware ESX Server 4.0.0, which is a bare metal VMM and is another example of an attacker obtaining arbitrary code execution.

Another example of an I/O related vulnerability is the heap-based buffer overflow in the `bx_ne2k_c::rx_frame()` function in `iodev/ne2k.cc` of the emulated NE2000 network interface card in Bochs 2.3, which allows a root user of the guest VM to write to memory locations belonging to the Bochs process [102]. This leads to a Guest to VMM or Guest to Host escape (once again, an example of arbitrary code execution).

Undocumented instructions can also introduce vulnerabilities when executed in virtualized environments: The `aam` instruction (*ASCII Adjust after Multiplication*) in QEMU 0.8.2, which is undocumented but is supported by the divisor hardware and is not privileged. Thus, this instruction can cause divide by zero errors, which can be used by a guest VM to perform possible DoS attacks against the host [102] (disrupting services that it should not).

4.2 Guest to Guest

This category involves one guest VM attacking (or somehow impacting) other guest VMs. This may be due to vulnerabilities in the hypervisor, services provided by the host machine, or services executed in other guests.

When a guest VM using a GPU in the pass-through mode is shutdown, there is a possibility that another guest VM is started and assigned to the same GPU afterwards. In certain situations, the GPU memory will not be completely erased and the new guest VM is able to read the information left by a previous guest. This vulnerability was confirmed on NVIDIA GPUs [91]. Given the pass-through of devices is supported for many different classes, this issue might exist in other devices as well. The primitive exposed (reading data from other VMs) is interesting and the impact highly vary depending on the data available to the attacker. The mitigation for this requires the devices to fully erase their memories (or the hypervisor software ensuring that).

A commonly discussed class of problems in virtualized environment is related to *side-channels*. An example is an extension of the Bernstein's attack [13], which relies on the dependency of the Advanced Encryption Standard (AES) algorithm on fetching values from tables in memory. An attacker can take advantage of being in the same physical system as the target. During the first phase, the attacker encrypts known plain-texts using a known key and profiles the performance for each key bit in a copy of the target server, called a *profiling server* (this is plausible since cloud implementations usually create equal instances of popular configurations when deploying new guests) and stores this information in a table. Then, the attacker sends the known plain-texts to the target server while profiling the timing for the encryption this time using an unknown key. This information is also registered in a table. Then, the correlation between both profiles are analyzed resulting in a table with the most probable key candidates. This reduces the key space to only a few

entries, which can then be brute-forced to obtain the secret AES key [8]. While this attack is based on the underlying hardware characteristics, in many cases it is possible to mitigate these risks using software-based techniques. Some techniques will be discussed in Section 5.4, which discusses side-channel attacks. The primitive obtained by a side-channels is the ability to read otherwise protected data (and the context, configuration and usage patterns of the system) highly influence on the impact. Usual side-channel mitigation recommendations apply also to virtualized environment (i.e., protecting from un-trusted elements running in another hyper-threading, scheduling trusted entities, avoiding unintentional speculation).

While the previous case is an example of an attack on confidentiality (i.e., disclosure of encryption keys), it has been shown that a Guest to Guest attack can also be performed from a malicious VM to bypass security measures of a co-resident target. Using a read primitive chained together with another vulnerability can lead to mitigation bypasses and is highly discussed in overall software security research [138].

Kim *et al.* presented a hardware problem in DRAM memory modules [79] where commodity DRAM modules (especially, modules that were manufactured after 2012) were vulnerable to disturbance errors. They showed that disturbance errors can be induced into a DRAM using a simple user mode program to repeatedly perform read operations on specific rows inside a memory bank. This resulted in adjacent rows being modified (i.e., bit flips - which is a form of write primitive). Attacks exploiting this problem are known as *rowhammer attacks*, and while not specific to virtualization, also impact virtualized environments. The mitigation for the *rowhammer* attacks came in the form of firmware patches to change the update frequency of DRAMs.

In [141], this vulnerability is used by a malicious guest to modify the state of a loaded executable file in a target guest. The attack, a variation of the *Flip Feng Shui* work presented in [119], takes advantage of *memory deduplication*, which is a mechanism that scans memory and merges pages that share the same content. An executable file is loaded into the attacker as well as the target. Since the file will contain pages that share the same content, they are merged by the deduplication mechanism. Once they are merged, a *rowhammer* attack from the malicious guest will modify the loaded file in the target. Talbi demonstrated that specific bits in authentication modules (Linux-PAM was used as example) could be flipped to bypass the authentication process. While this is an example of a *rowhammer* attack, it is listed in this survey to demonstrate the creativity of attackers on leveraging/chaining functionalities to increase the impact of their primitives (even if initially apparently highly limited). Deduplication mechanisms have been abused in side-channel attacks [46] as well, and the mitigation has been simply disabling the feature for platforms that uses a shared infra-structure between un-trusted entities and see the virtualization as a security boundary.

A different approach on exploiting *rowhammer* for a cross-VM attack (demonstrating the creativeness of attackers to abuse primitives) is presented in *One Bit Flips, One Cloud Flops* [161].

4.3 Guest to Self

This is perhaps the least intuitive category in which an unprivileged process running in a guest VM is able to elevate its privileges because of a vulnerability in the hypervisor (or due to changes made to the guest VM to support paravirtualization).

One such an example occurred in QEMU 0.8.2 that allowed local users to halt the guest VM by executing the undocumented instruction `icebp` (*ICE Breakpoint*) [21] [102]. This can be used to perform DoS attacks against the guest VM itself. A very similar issue impacting instruction emulation affected Microsoft Hyper-V in many Windows versions (Microsoft Windows Server 2008 SP2 and R2 SP1, Windows 7 SP1, Windows 8.1, Windows Server 2012 Gold and R2, Windows RT 8.1, Windows 10 Gold, 1511, 1607, 1703, and Windows Server 2016) but had as a consequence, the elevation of the privileges (which in essence gives arbitrary code execution capabilities) of a

program inside the guest itself [38]. While there are no systemic mitigations for this kind of issues, usual code security practices help detect/prevent a considerable part.

4.4 Host to Self

This category is similar to the Guest to Self case, but instead of an unprivileged guest process elevating its privileges, a process running in the host is able to elevate privileges by exploiting a vulnerability present in the hypervisor. While similar problems can exist in non-virtualized platforms, the difference is that the vulnerabilities in this case are present in the hypervisor code.

An example of such a case occurred in Oracle VM VirtualBox. The data structure `struct VM` mapped as read/write in *vboxsdl.exe* and in *VMMR0.sys* contains pointers to code in kernel mode. An unprivileged process in the host can overwrite these pointers (a write primitive) and achieve kernel code execution. This specific instance of the vulnerability applies only to Windows hosts [159]. The attack vector is the shared folders between the host and the guest. As with the Guest to Self category, there are no general mitigations against similar issues.

5 IMPACT OF VIRTUALIZATION TECHNOLOGIES ON HOST SECURITY

Virtualization is not inherently secure or insecure (although early VMMs such as VM/370 included security in their design goals [40]). During his Black Hat USA 2012 presentation, Kostya Kortchinsky said: "VMware isn't a security layer. It's just another layer to find bugs in." [82].

Adding virtualization to a platform can make the task of securing it more challenging. Virtualization of the I/O functions and device redirection (for example) add to the existing complexity of the system and increase its attack surface. Virtualization also breaks previous assumptions held in traditional systems: we can no longer assume that our system has exclusive ownership of a physical host and side channels are introduced. New attack points are created (some instructions can be intercepted by the VMM). The following subsections discuss these new challenges.

5.1 New requirements introduced by I/O Virtualization

In a traditional computer system, devices are configured in a privileged mode (e.g., ring0/kernel-mode) and are trusted by all applications. However, virtualization changes this model because devices might be configured by an entity that is not trusted by the whole system including the guest VMs themselves. To address this trust issue, devices must somehow be virtualized as well. The virtualization of I/O operations also improves utilization and eases management, but it introduces a new layer of complexity due to the isolation required between the VMM/host and the guest VMs. Early implementations of I/O virtualization added a performance penalty, which can be larger than a factor of two in systems with a heavy workload [125]. Moreover, since these early mechanisms depended exclusively on software, they also potentially introduced a new source of vulnerabilities [117]. Especially problematic was PCI devices performing Direct Memory Access (DMA), which could be used to access arbitrary locations in memory including memory of current processes. An entire class of attacks, known as *DMA attacks*, exploits this kind of vulnerability [158].

CPU manufacturers address this issue by providing hardware support for I/O Virtualization. For example, Intel *Virtualization Technology for Directed I/O* (VT-d) and AMD *I/O Virtualization Technology* (Vi) incorporate an Input/Output Memory Management Unit (IOMMU), which allows the I/O bus to be connected to the main memory of the system and performs the translation of I/O addresses to addresses in main memory.

Figure 4 (right) shows how a DMA re-mapping feature can be used to assign specific memory domains to a device preventing it from generating requests out of its assigned domains. This is achieved by having the IOMMU intercept DMA requests and reading the physical address associated

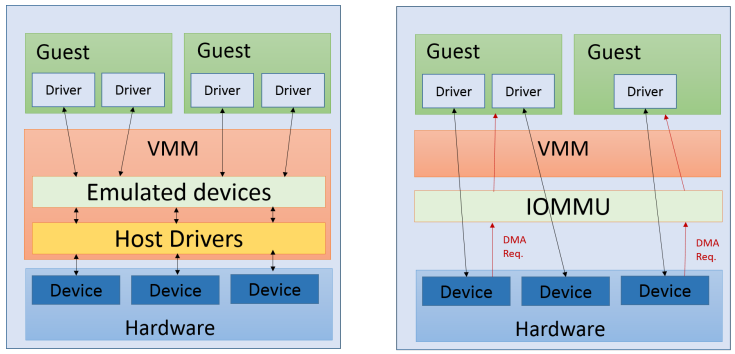

Fig. 4. The IOMMU DMA remapping engine (right) allows for separation of memory domains between devices. DMA requests by the devices are handled by the engine and redirected only to the assigned domain.

to them. It then translates the domain physical address to a host physical address. This translation prevents a device from accessing host addresses that it has not been assigned.

Intel VT-d has been the subject of discussion for potential vulnerabilities. Sang *et al.* discussed the possibility of corrupting the mapping structures that assign I/O devices to memory regions [133]. However, since these structures reside in memory areas that are non-accessible to user applications, this approach would require the attacker to be able to execute code in the host with high privileges. Some other possible attack vectors have also been identified. For example, the configuration registers in the DMA re-mapping unit could be modified to make it point to different configuration tables. Sang *et al.* suggest mitigating this by using mechanisms that require higher levels of privilege than the kernel [134]. Newer versions of the Intel platforms have the capability to prevent these registers from being modified after initial configuration. There is also an attack that causes a malicious I/O device to generate a malformed request using the source-id from a different device, which would allow an attacker to access the other device's memory region. This attack can be mitigated by PCIe Access Control Services (ACS), which perform validations to detect Transaction Layer Packets (TLPs) with invalid source-id [103][134].

Even with the protections mentioned previously, new attacks are being discovered. Pék *et al.* developed a tool called *PTFuzz* to fuzz the Message Signaled Interrupt (MSI) generation process [104]. In some instances, an MSI can be configured to trigger a legacy Non-Maskable Interrupt (NMI), which is normally generated as a result of a hardware error that must be immediately handled by the CPU. This interrupt cannot be re-mapped by the IOMMU and therefore cause a software fault. This leads to serious potential vulnerabilities such as a DoS or code execution in the host.

Although technologies such as Intel VT-d and AMD-Vi play an important part in protecting memory from attacks in virtualized environments, there are also other measures that are used to protect data from corruption. A couple of examples are mentioned below:

5.1.1 DMA Protected Range (DPR). Intel *Trusted Execution Technology* (TXT) can be used to verify the integrity of the code elements that will be executed on a system at launch time [72]. Intel TXT defines a *DMA Protected Range* (DPR), which is a protected area (from DMA accesses) used to store structures and information relevant to the TXT launch process. The size of this region is determined by the BIOS [66][71][72].

5.1.2 Protected Memory Regions. While DPR is specific to Intel TXT, *Protected Memory Regions* (PMRs) are defined in the Intel VT-d specification and can be used to initialize VT-d specific structures. They are also protected from DMA accesses. Optionally, one of these regions can be used to store Intel TXT structures [70][72].

5.2 Increase of the attack surface: Bugs in the VMM code

Adding a complexity layer to an existing system will increase its attack surface. VMMs are software products and as such they are prone to security bugs. Although it is desirable for any software to have a code size as small as possible, the functions that have to be performed by a production-ready VMM are complex: The interface to the virtualization hardware (including I/O virtualization, discussed in the previous section), VM Exits and Hypercalls (to return control to the VMM), VM management software (for creation and administration of guest VMs) and VMM add-ons (which extend the functionality of the VMM) are all areas that can be used by a malicious agent as attack vectors [105].

To reduce the possibility of exploitable bugs, VMM vendors must be looking for ways to keep the size of the attack surface in check, specially in present times, where more security problems in VMMs are being reported [55]. As some basic aspects of the virtualization support are implemented in hardware (with the advent of technologies already discussed), the VMM code can be relieved of some of this complexity.

5.3 Increase of the attack surface: IOMMU support bugs in the OS code

A hardware protection such as the IOMMU cannot operate on its own: It needs support from the Firmware and the Operating System. If the protections provided are not correctly enabled or configured, new vulnerabilities will emerge. Project *Thunderclap* [90] created an open source FPGA platform capable of producing a software model for a wide range of DMA-enabled peripherals, allowing it to test from simple DMA requests to complex interactions with the OS. The platform was able to detect issues such as the ability of devices to inspect memory intended for a different device, VPN cleartext data exposure and even kernel code injection when modeling a NIC device. Common Operating Systems such as Windows, MacOS, Linux and FreeBSD were all affected by vulnerabilities.

5.4 Vulnerability to side-channels

Side-channel attacks are based on the general concept of obtaining specific data (e.g., cryptographic keys) by monitoring activity in a target system to look for information disclosed unintentionally. This information is related to the data the attacker wants to obtain. There is a vast amount of work in the literature on this subject. Zhou and Feng proposed a classification scheme for these attacks according to the following criteria [166]:

- Control of the computation process: Whether an attacker can influence the computation process in the target.
- Ways of accessing the [cryptographic hardware] module: This classification is not in the context of virtualization and will not be discussed here.
- Method used in the analysis process: The attacks discussed below are classified in this category. Some of these methods, such as electromagnetic, acoustic, power analysis, and visible light, require at least some degree of physical proximity to the target. On the other hand, timing and cache-based attacks can be performed remotely.

At first, it may seem easy to dismiss the threats posed by side channels in virtualization-based environments since a prerequisite for many of these threats is *co-residency* of an attacker with its

target (i.e., both attacker and target residing on the same physical host). Since an attacker does not have control over the location of a guest VM instance, this would seem to be no more than a theoretical risk. However, Ristenpart *et al.* and Varadarajan *et al.*demonstrated using popular cloud providers that it is possible to achieve co-residency with a specific guest VM [123][146].

The following discusses vulnerabilities that arise from different side-channels available in virtualized environments. The discussion starts with a well-known class of issues that affect cryptographic operations and then moves to more recent discoveries.

5.4.1 Extraction of cryptographic keys using cache access timing. Zhang *et al.* made a distinction between *fine-grained* and *coarse-grained* attacks [164]. Fine-grained attacks allow the retrieval of specific information, such as cryptographic keys from a target system. On the other hand, coarse-grained attacks cannot reveal that level of detail, but they can give information such as which OS is running on the target. Their work demonstrated that it is possible to mount a fine-grained attack even in a virtualized environment running on a SMP system, where the attacker and target systems are assigned to different cores. In their attack, they managed to obtain information about cache accesses with enough frequency and removed noise to retrieve a decryption key from an application using the popular library *libgcrypt*. It is important to note that coarse-grained attacks are also relevant in a virtualized environment because reliable exploitation oftentimes depends on knowing the OS running on the host and this may allow exploitation of other vulnerabilities, such as buffer overflows that lead to guest VM escapes.

5.4.2 Controlled channel attack. This is a recently introduced type of attack, which assumes an adversarial OS (although this can be performed also by a VMM given its level of privilege) trying to break isolation from a trusted environment to extract confidential information (e.g., Digital Rights Management (DRM) protected content) from a legacy application executing within a protected area. Since the application is legacy, it is also assumed that the attacker has access to its code (in source or binary form). This code is analyzed to identify control flows that are input dependent. Since the attacker has control over resource management in the platform, it can manipulate memory mappings in the system to cause page faults when the protected application tries to access specific pages. The OS can keep track of the accessed pages during page fault handling, and extract the information that was being processed by the application by analyzing the access patterns [162].

5.4.3 Information leakage using memory de-duplication timing. Memory Deduplication (or *kernel same-page merging*) is a strategy that allows virtualized platforms to make a more efficient use of the memory by scanning for identical pages in the physical memory and share them among the guest VMs. However, this also introduces a channel for information disclosure. The "copy-on-write" mechanism separates the page again when a guest VM attempts to overwrite it. This separation process causes a difference in access time, which can be measured allowing an attacker to know whether a specific page was deduplicated prior to access. This allows an attacker to discover the presence of specific applications or files in the targeted guest, thereby breaking the isolation requirement of virtualization [140].

Barresi *et al.* discussed another problem where the memory de-duplication mechanism can also be abused to correctly guess the random base address for important libraries (e.g., *ntdll.dll* on Windows) in a target system essentially making ASLR ineffective [11]. This is achieved by setting up an attack server to generate several candidate copies of the first page of a *.dll* library. In Windows Portable Executable format, the `ImageBase` field contains the base address of the *.dll* in a process address space. Each candidate copy can contain a different guess for this address. If one of these candidates contains a match, the VMM de-duplication process will merge this page with the copy in the target. As a result, a write operation to this page by the attack server will experience a delay

compared to a write operation to a non-merged page. By comparing the write access times of the candidates, it is possible to find the merged page, and thus the base address of the library. The discovered base address can be used to bypass defense mechanisms, such as ASLR.

While unrelated to side-channels, the de-duplication mechanism was also demonstrated to be useful to improve the reliability of *rowhammer* attacks [46].

5.4.4 Breaking ASLR using Branch Target Buffer collisions. Evtyushkin *et al.* presented a technique to create a side-channel based on branch prediction [47]. The *Branch Target Buffer* (BTB) is a cache used to enhance performance by storing the target addresses of conditional branch and jump instructions. Since BTB is shared among applications, the performance of a given application can be affected by another application due to collision. This performance difference can be measured by the CPU timing mechanisms (using instructions `RDTSC` and `RDTSCP`) inside a specially crafted malicious process. They also showed that this side channel can be used to undermine ASLR for user processes and also for kernel code (KASLR). Later on, Wilhelm published a Proof of Concept (PoC), where this side channel was used by a guest VM to find the address of a Hypervisor Kernel Module [157].

5.4.5 Speculative Execution Attacks. Speculative Execution is a technique designed to improve the performance of a CPU by allowing microarchitectural execution of instructions even in the presence of incomplete data (data pending to be read from memory). The processor executes the instruction sequence without commiting the results or performing any security checks on them. If the results are found to be incorrect, they are not committed [65]. It used to be believed that this form of operation did not have any security implications because the processor is reverted to its original architectural state and the final results are not influenced, however, the uncommited microarchitectural changes are not reverted. This can leave traces that can be probed with architectural mechanisms. In January of 2018, a technically feasible attack was published that made use of these techniques to induce a victim process to leak information from its own address space to an attacker: *Spectre*[3] is a blanket name for a series of attacks where a malicious process sets up a victim process to trick it into executing speculative instructions (e.g., removing data from the cache), then triggers the execution of these instructions by explicitly requesting action from the victim. After the execution, the attacker retrieves the leaked information. The published attack uses *Flush+Reload* and *Evict+Reload* techniques as a covert channel to cross the boundary between the microarchitectural and architectural state. A variant of it was used by a guest VM to retrieve memory from a KVM host [81].

Another related attack: *L1TF* (also named *Foreshadow*) was published in August of 2018. It also uses speculative execution as attack mechanism. Two versions have been identified: The original published version attacks Intel SGX enclaves [144], and the second (subsequently discovered by Intel) can be used by a guest VM to read memory from coresident VMs or the VMM itself [156].

At this point, the list of new attacks based on the initial findings related to speculative behavior is fairly large [37] [33] [34] [35] [36] [26] [28] [27] [25] [29] [32] [30] [31].

An obvious measure against side channel attacks among VMs is to prevent co-residency. However, this beats the purpose of efficiently using the host resources since one of the main reasons for using virtualization in cloud environments is to consolidate a number of machines. In the following subsections, some mitigation approaches that can be implemented against cache-based timing attacks are discussed. The reason to discuss the mitigations against side-channels are because

[3] A related attack: *Meltdown*, was disclosed at the same time, but it has been determined that it cannot be used to cross the boundaries set by virtualized systems. So it will not be discussed here.

they have specific implementation challenges/characteristics in virtualized environment (while mitigations for overall software issues do not).

5.4.6 Cache isolation. In an attempt to provide cache isolation without specialized hardware support, the StealthMem project aims to protect VMs against cache-based side channel attacks by assigning a set of cache lines to each core and making sure that the pages of a given VM only use the assigned cache lines (by using the associativity principle of the corresponding physical addresses). This allows the VM to keep a memory area as if it was private since the assigned cache lines are safe from inspection by other VMs. This isolated memory area is called *Stealth memory*. Applications can use APIs *sm_alloc()* and *sm_free()* to allocate and deallocate stealth memory, and then store sensitive information in these areas [78].

While the software approach does provide isolation, any application (or VM) that makes excessive use of a shared resource such as the *Last Level Cache* (LLC) can affect the performance of others, which in essence can cause a DoS or relevant slow-down. This is referred to as the *Noisy Neighbor Problem* and the VMM by itself does not have a way to find out which application is the culprit. Therefore, there are hardware-supported features, such as *Intel Cache Monitoring Technology* (CMT) [62], that monitor LLC utilization by applications that coexist inside a platform. A related technology called *Intel Cache Allocation Technology* (CAT) [98] exposes a *Class of Service* (CLOS) interface to software, allowing it to define cache utilization tags that can be used to group threads, applications, virtual machines or containers under a common usage model. The CLOS is flexible enough via *Capacity Bit-Masks* (CBMs), which are bitmasks defined per CLOS to control overlap and allocation of different cache areas, to specify which applications have higher priority and the level of cache overlapping with other applications. *Intel Cache Allocation Technology* (CAT) was effectively leveraged by [48] to implement side-channel mitigation.

5.4.7 Performance Counters and Side-Channel Detection. The *Performance Monitor Unit (PMU)* is a mechanism created by Intel to help software diagnostic the system behavior. In [142], the authors demonstrate that side-channel attacks can be detected by using the technology. When the speculative side-channels became public, the company Endgame [45] also proposed a similar approach.

5.4.8 Scheduler-based defenses. In order to track the activity of a target system, the frequency of re-scheduling between the attacker and the target must be sufficiently high so the attacker can inspect the channel enough times to retrieve information. Varadarajan *et al.* introduced the concept of *soft isolation* to explore how this frequency can be reduced without having a major impact on the system performance [145]. In the Xen hypervisor, the Minimum Run Time (MRT) for each virtual CPU can be manipulated with the `ratelimit` parameter. Using this mechanism, attacks like the Cross-VM attack explained in [164] can be stopped since their observations of the target will fail to render enough useful data. It was also shown that this mechanism can be combined with cleansing the state of the CPU for additional protection in cases where control of the physical CPU is relinquished by the guest VM. Another important example of scheduler-based defense is *Nomad* [93].

5.4.9 Obfuscation of timing information from the guest. Project *Düppel* is designed to protect guest VMs from time-shared cache attacks [165]. It is intended as a measure against the *Prime+Probe* attack technique in which a malicious guest executing in a given physical CPU fills the shared cache sets with its own data (called the *Prime* phase), then relinquishes the CPU. When the target acquires the CPU and performs its own activity, some data belonging to the attacker will be evicted from the cache. After the target relinquishes the CPU, the malicious guest can obtain information about the target activity by trying to fill again the cache sets and measuring the time this process takes (called

the *Probe* phase). Düppel provides a mitigation by cleansing the caches between the attacker's *Probe* phases introducing noise to the information that the attacker can obtain. This approach does not require modifications to the hypervisor (as is the case for Scheduler-based defenses), but change is required to the guest OS kernel.

5.4.10 Limiting access to CPU timing resources. A possible mitigation for attacks such as the one described in 5.4.4 is to limit guest access to high-precision CPU timers, which are essential for computing timing differences. However, this does not completely eliminate the problem since a good enough timing can be obtained by having threads cooperate to measure the time of an attacker process [58].

5.4.11 Mitigations against speculative execution attacks. The *Spectre* attacks described in 5.4.5 are of such importance that they are motivating an effort from the CPU industry to deliver appropriate mitigations and make their new designs more resistant to them [83][4]. Some mitigations are software based and involve inserting specific instructions in selected parts of the code to limit speculative execution, while others involve preventing a malicious process to control the indirect branch predictions of a potential victim. These last mitigations require microcode updates as well as software support [65] [5].

6 LEVERAGING VIRTUALIZATION TECHNOLOGIES FOR SECURITY PURPOSES

In this section we discuss how virtualization can be used to help securing a platform. The solutions presented are associated to the components of the classic information security triad[4] : Subsection 6.1 gives an example of VMs designed to protect *Confidentiality* of data at rest and in transit (inside the platform). Subsection 6.2 discusses how the *Integrity* of certain components of the system can be protected (examples are given for filesystem and OS kernel). Section 6.3 discusses how virtualization-based fault tolerant architectures can be implemented in support of *Availability*. In the remaining subsections, some additional applications and techniques that are useful for security will be discussed.

6.1 Confidentiality

A general-purpose computing system processes a multitude of data. Different organizations (and users) have different requirements in regards to who is allowed to access information generated by their systems. Confidentiality is the security objective of protecting data from unauthorized access, where the authorized entity is defined by the information owner or an authorized administrator. This subsection explains the potential impact on confidentiality of using virtualization.

6.1.1 Privacy-preserving virtual machines. Mechanisms such as *VM snapshots*, which creates a complete copy of the guest state, can put confidential data (e.g., passwords in plain text form, encryption keys, credit card numbers, etc.) stored inside the memory of a guest VM at risk. An attacker can have access to this data by simply copying the snapshot. Li *et al.* addressed this problem using a second VM, called *Privacy-Preserving Virtual Machine* (PPVM), spawned from the original [86]. PPVM allows the confidential data to be excluded from the snapshot process or processed independently to include security features like encryption. This solution has the additional advantages of being transparent to the confidential applications and having low overhead. Another example of privacy-preserving use of virtualization is project *Overshadow*: It takes advantage of the extra level of memory indirection offered by VMMs to separate applications data and offer

[4]The CIA security triad (Confidentiality, Integrity and Availability) is a well known concept in information security. While the three properties considered separately have been understood since before the computer era, the exact origin of their use as part of a security model has not been identified.

an encrypted view of it to a guest OS, while offering a plaintext view to the application itself, protecting its contents from a compromised OS [19].

6.1.2 Protection from data flow analysis. Applications that are not designed with the user's privacy in mind usually leave behind traces of their activity. Dunn *et al.* have shown how even "secured" systems cannot erase these traces completely due to the complexity of their software stacks [42]. In their work, they provided the example of a Linux system where memory that is allocated and used as a cache by the X server remains allocated even after an X client application terminates, which can be inspected to reveal the screen output of applications that are no longer being executed. Similar exposures were found in audio applications, memory caches, and network buffers.

The project *Lacuna* goes beyond the traditional approaches of secure file deleting and clearing deallocated memory [42], which is based on the following two key concepts:

(1) A *private session* – A user can create a session within which usual activities such as browsing the web, reading email, etc. can be performed. This session is confined to a VM.
(2) One or more *ephemeral channels* – Although a VM prevents data leakage from inter-process communications, data generated from the interaction between the VM and external devices are still at risk of exposure. Ephemeral channels provide a mechanism to protect data as it flows through the host OS, and allow only the endpoints (i.e., the device and the guest VM) to see it in its unencrypted form. Because of the encryption, data allocated within the host OS is unreadable. At session termination, the encryption key is erased from the system to make sensitive data unrecoverable.

The data privacy protection that can be offered by virtualization can benefit enormously by working in tandem with encryption technologies. Solutions like AMD SEV (discussed in subsection 2.2.3) Can provide additional protection for data residing in main memory.

6.2 Integrity

Any data generated or stored in a general-purpose computing system should be modified only by authorized users. Data modification by unauthorized users (or due to bugs) affect a security objective known as *integrity*. The VMM is capable of intercepting and monitoring many parts of a guest VM (and under different events) that would be very difficult or impossible to monitor in a standalone system. Most research in this area is focused on protecting a guest VM from being exploited. The integrity of components in a guest VM, such as the file system or the OS kernel, can be constantly monitored by the VMM. The following subsections describe some examples of this approach.

6.2.1 File system integrity. FSGuard was designed to monitor the integrity of a file system and implemented on a Xen platform [155]. The system to be monitored is located under an unprivileged guest (DomU), which has no access to the hardware, and Xen is modified to intercept system calls. When a system call is intercepted, its information is stored in the processor's registers. The registers are then examined to ignore system calls that are not related to read/write operations on files. Other important information, such as a file path, is also accessible in this way. The user can configure in advance the files that need to be protected (via the UI). If the user is allowed to perform the operation based on the analysis of the system call and the defined protection policies, FSGuard allows the normal flow of the call to continue. If the operation is prohibited, the flow is interrupted and the operation is rolled back.

6.2.2 Kernel Integrity. The OS kernel has unrestricted access to all the resources in the system. This makes the kernel code a common target for attacks. Seshadri *et al.* identified the following three basic kinds of kernel attacks [139]:

- Acquiring root privileges and loading an arbitrary kernel module.
- Compromising the running kernel by exploiting a vulnerability.
- Utilizing DMA-capable peripherals to corrupt the kernel memory.

The following enumerates the recent efforts to tackle this problem using virtualization technologies:

SecVisor SecVisor is implemented as a very small hypervisor that preserves the integrity of kernel code by virtualizing the MMU and the IOMMU [139], where the former allows for intercepting and checking of any change attempts to the addresses mapped to the protected system kernel while the latter allows for protecting the kernel memory from write operations by DMA-capable devices. SecVisor relies on the built-in CPU protections provided by Intel VT-x (see Section 2.1) and uses the notion of approved code, i.e., only codes that have been approved by SecVisor can be executed. Kernel mode entry and exit are monitored so that the appropriate execution permissions are set. Thus, it checks that kernel mode entries are made only to addresses belonging to approved code, and kernel mode exits will set the CPU privilege back to user mode. The dynamic changes in the permissions is functionally equivalent to enforcing the $W \oplus X$ property in the kernel code [87]. One downside of this solution is that slight modifications to the kernel source code are required making it unsuitable for closed source products. Also, it is unable to handle memory pages with mixed code and data due to the enforcement of $W \oplus X$, i.e., data mixed with code cannot be modified.

No Instruction Creeping into Kernel Level Executed (NICKLE) This solution takes advantage of the isolation property of VMM. When a guest VM starts, the hypervisor allocates a memory space to maintain a shadow copy of the kernel code. This shadow copy is verified after decompression using a cryptographic hash. Instruction fetches are intercepted and if the current privilege level (CPL) indicates that the code being fetched is kernel code, both copies are compared. If they match, the code is executed. Otherwise, the code is assumed to have been modified without authorization and an appropriate action can be taken to mitigate the threat, which can be either aborting the execution or rewriting the instructions (essentially, recovering the original code) to allow the guest VM to handle the error. This guarantees that the kernel code cannot be modified even if a rootkit [5] manages to fully control the system memory permissions and modify them. This approach overcomes the limitation of SecVisor – the need for modifications to the source code to avoid pages with mixed code and data. Moreover, it is able to run transparently to the guest VM. Another advantage is the ability to handle pages where code and data are mixed [121].

hvmHarvard Systems based on x86 are an example of a von Neumann architecture, where code and data share a single address space. This leads to the possibility of modifying code at runtime if no other protection measures exist. On the other hand, a system based on a Harvard architecture will allocate separate address spaces for code and data (see Figure 5). This mitigates the risk of *code injection attacks*, which involves injecting a code stream to some program input and redirecting the flow of execution to make the system execute the injected code [120]. *hvmHarvard* utilizes virtualization to create separate memory areas for code and data. NICKLE also emulates a Harvard architecture using virtualization, but it traps all instruction fetches impacting performance. hvmHarvard gets around this by taking advantage of the fact that Translation Lookaside Buffers (TLBs) for instructions and data can be *desynchronized*, which means that the same virtual/logical address points to their

[5]Rootkit is a code that is injected in a compromised system to subvert normal operation usually by giving unauthorized access to an attacker as administrator or root.)

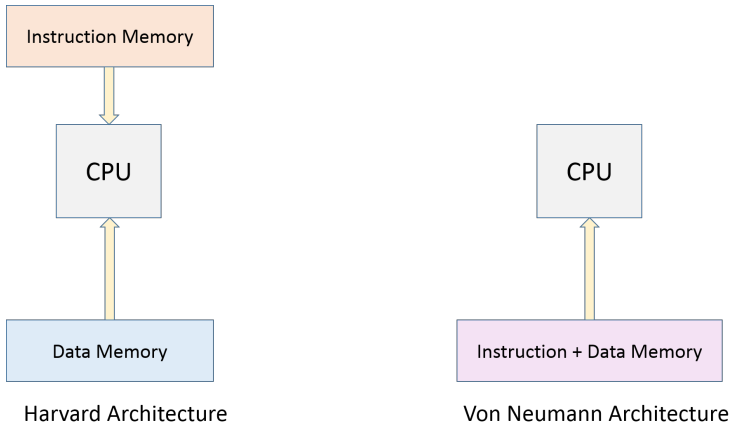

Fig. 5. Harvard and Von Neumann architectures. In a Harvard architecture, since code and data are stored in different address spaces, data manipulated by an attacker cannot be injected for execution by the CPU.

respective entries in the two TLBs, and the physical addresses in the two entries are then made to point to different frames effectively separating code and data memory areas [57].

Intel Kernel Guard Technology Intel Kernel Guard Technology (iKGT) takes advantage of Intel VT-x technology to protect critical platform assets such as kernel code pages and data structures. It includes a software component: *Xmon*, which is a thin VMX-root layer module that executes at the VMM level and deprivileges the operating system. Since this component has higher privilege than a guest VM, it can intercept calls to read or write the protected assets and enforce a predefined policy. The policy specifies the assets to protect and the actions to be taken when these assets are accessed. iKGT has been released as open source and its current version supports Linux guests [84].

6.3 Availability

Data must be available to the authorized users when needed. This security objective is called *availability*. This section discusses how virtualization can be used to support availability requirements by facilitating implementation of fault-tolerant solutions.

6.3.1 Hypervisor-based persistence. Data availability and integrity can be negatively impacted by power outages. Information stored in volatile memory is especially vulnerable and a number of strategies have been developed to mitigate the risk of data loss. One solution that has been proposed is to take advantage of Non-Volatile Dual In-line Memory Modules (NVDIMMs), which are DRAM modules with flash memories, in commodity servers by adding an extension to VMM. Since the software modification takes place at the virtualization layer, the changes are transparent to guest VMs. Some elements of the state of a VM, such as the state of the CPU and the states of virtual devices, need to be explicitly written to the NVRAM since they are not automatically backed up. Moreover, the underlying hardware must include a Power Outage Detector device that detects sudden voltage drops and generates an interrupt to save the CPU state to the NVRAM. This approach has been tested in commercially available hardware with promising results [135].

6.3.2 Fault-Tolerant Virtual Machines. Deployment of fault-tolerant systems using redundancy can be done conveniently using VMs. This feature is available for several virtualization platforms (like VMware vSphere, XenServer, Red Hat Enterprise Virtualization among others). Scales *et al.*

presented a complete implementation using commercial tools [136]. The problem of synchronizing information between the primary system and the backup is addressed by modelling it as a state machine: Instead of syncing the complete state of the primary server (including CPU, memory, and I/O devices), both systems are started in the same state and the inputs to the primary are duplicated for the backup using a *logging channel*. When a failure is detected in the primary system, the backup can be used to pick up from the point in time where the primary failed with no interruption of service.

6.3.3 VM snapshots. Some virtualization solutions today provide the ability to generate a copy of the state of a VM at a given time, and then resume execution from it [20] [149]. This copy is called a *VM snapshot* and can be used for several purposes (quick deployment of several identical VMs, as a migration mechanism, etc.) but it can also be used as a protection measure in case the original VM becomes unavailable due to a failure or a crash. This is the VMM equivalent of the traditional solution of *Checkpoint-Restart* strategy that exists in other domains in computing [23].

6.4 Network Security

Network Functions Virtualization (NFV) is a technology that takes advantage of virtualization to implement network services that traditionally are deployed in physical, vendor-specific hardware. Virtualized Network Functions can work along with *Software Defined Networks* (SDN) to provide flexibility and ease of management: The VNFs are easily deployed on commodity hardware using VMM management facilities. Examples of network functions that can be virtualized are security functions like Firewalls and Intrusion Detection Systems [160].

6.5 Malware Analysis

With malware attacks steadily on the rise [85], the arms race between security professionals and malware creators does not seem to be slowing down. Dynamic malware analysis is usually performed in a controlled environment to protect the system from malicious activity on the sample under study. Virtualization can work alone or in combination with other technologies to provide this controlled environment.

6.5.1 Virtual Machine Introspection. Since high privilege software such as the VMM has access to the resources of its managed guests, it can be used to inspect the state of a VM from the outside. This technique is called *Virtual Machine Introspection* (VMI). The concept was introduced in [52] applied to intrusion detection systems (IDS), but its usefulness has been extended to several other applications, including malware analysis.

Project *VMwatcher* makes use of VMI to obtain the current low level state of a VM (registers, memory contents and persistent storage). It performs this in a non-intrusive manner by observing and not interferring with its functioning. This ability to watch the activity inside the VM from the outside comes at the cost of losing the abstractions available to mechanisms operating in the inside (processes running in memory, files, network connections, etc.). This is known as the *semantic gap*. In order to narrow this gap, VMWatcher utilizes a technique called *Guest View Casting* to reconstruct these abstractions and make them available to the malware detection system, which operates outside the VM under analysis [73]

6.5.2 Combination of virtualization and emulation techniques. An *emulated* system (which, as opposed to virtualization, is just a software representation of hardware) provides a suitable platform for malware analysis. It can be tailored to facilitate the analysis. However, malware developers are improving their techniques to detect when they are being executed in an emulated environment and block the task of the security analyst. An emulated environment can mimic the behavior of

a real system, but it cannot be done in a transparent manner because of the amount of legacy hardware behavior that must be supported and the timing differences in emulation versus real execution. The detection of an emulated environment is easy in standalone systems [163] [76]. On the other hand, detecting virtualized platforms is harder (although there are techniques to do it as discussed in 7.2.1), but they do not offer the flexibility of emulated systems.

Yan *et al.* proposed a combined approach where a malware sample is first executed in a virtualized platform to record its activity [163]. This is called the *reference platform*. Afterwards, its execution is replayed in an *emulated platform*, which has been tailored using dynamic binary translation to provide instrumentation support. The emulated platform is then modified (i.e., adapted) based on the observed deviations of the malware behavior from the reference platform. The emulation platform will now execute the malware code as if it was running in the reference platform and suppress the possible anti-emulation techniques used by the malware code. This approach allows the analyst to leverage both the transparency offered by virtualized environments and the flexibility of emulated systems.

6.6 Trusted VMMs

Trusted Computing is a set of specifications proposed by the Trusted Computing Group [61]. The technologies described in these specifications aim to provide security to computing platforms through a cooperation between software and hardware specifically designed for this purpose.

Project *Terra* is an example of the use of a VMM as a trusted element of the platform. This *Trusted VMM* extends the usual functions of traditional hypervisors and provides capabilities for deploying VMs with specific security requirements that aim to provide the functionality of closed platforms. These VMs are called *closed box* systems. In addition, it can deploy *open box* systems: Standard VMs which allow the execution of commodity operating systems and applications.

The Trusted VMM also has the capacity to authenticate the software running in a closed box to external entities. This process is called *attestation* and it is an important element of trusted platforms. The Trusted VMM becomes an element of a certificate chain: The hardware certifies the firmware, the firmware certifies the bootloader and then the bootloader certifies the Trusted VMM. The Trusted VMM is capable of certifying the closed VMs [51].

6.7 Domain isolation

The capacity of virtualized environments to provide isolation can be used to protect security sensitive elements in the system by separating functions and assign dedicated domains (VMs) to them. This technique is known as *domain isolation*. Some examples of this technique follow.

6.7.1 Qubes. Project *Qubes* is a security-oriented OS based on Xen. It intends to enhance platform security by using a *compartmentalization* approach. Separate VMs (or domains) are created and assigned for security-sensitive functions in the system: The networking functions, storage, and administration are each confined to their separate lightweight VMs. Also, user applications run in their own domain. The isolation property prevents the compromise of the entire platform if one of the domains falls victim of an attack [132].

6.7.2 Driver domains. Driver Domains are an optional feature of the Xen hypervisor. The system administrator can setup unprivileged domains whose only function is to handle specific hardware devices. If one of the drivers fails, the integrity of the privileged domain (Dom0) is preserved. Also, if the driver is compromised, only its specific domain is left vulnerable [110].

6.8 Privilege Separation

Privilege separation is an approach where an application is divided into privileged and unprivileged parts in order to reduce the amount of code with high privilege executing at any given time [112].

6.8.1 Proxos. Proxos is an architecture based on VMM isolation, and makes a distinction between applications that are security-sensitive and applications that are not [88]. Security-sensitive applications are executed in their own private VM, while non-security-sensitive applications are executed in the commodity OS. This reduces the size of the TCB for the sensitive application to the bare functionality that it needs from the OS. This isolation prevents an attacker who has compromised an application running in the commodity OS to have access to the sensitive information of the application running in the private VM.

6.9 Virtualization Based Security

Virtualization features originally conceived for server environments are now being used to improve the security of desktop systems as well. Microsoft Windows 10 includes exploit mitigation features, which prevent certain vulnerabilities from being exploited. One such feature is based on virtualization technology called *Virtual Secure Mode* (VSM), which uses Microsoft Hyper-V to partition the resources of a system. One of the partitions, or *Virtual Trust Levels* (VTL per Microsoft nomenclature), is assigned to the OS (named VTL0 by Microsoft), while the other (VTL1) is used as a separate environment protected from the OS to store secret information like user credentials [75]. Microsoft implements this feature under the name *Credential Guard* [41]. Another security feature based on VSM is the capability to enforce code integrity controls outside of the protection domain of the OS. This feature is implemented by Microsoft under the name *VSM Protected Code Integrity* [41].

Any integrity-monitoring system must consider the techniques (and limitations in the defense) demonstrated in [118], as well as the capabilities discussed in [89] and [12], that demonstrate a lot of operations that can be securely arbitrated by leveraging virtualization.

7 MALICIOUS USE OF VIRTUALIZATION TECHNOLOGIES

A VMM is inherently more privileged than the guest VMs it supports. Thus, if the capabilities provided by virtualization are used for malicious purposes, traditional approaches to OS security (e.g., antivirus programs and malware detectors) that operate at the OS-level must be reevaluated. The following subsections discuss some research results in this area and their implications. Subsection 7.1 describes malware modules that take advantage of virtualization to achieve high level privileges in the target system. Their feasibility for real world attacks is also discussed. Since these malware modules are highly privileged, special techniques must be devised for their detection. Subsection 7.2 shows different approaches to this problem.

7.1 Malicious VMMs

King *et al.* introduced the concept of *Virtual Machine Based Rootkit* (VMBR) [80], which is based on the principle of achieving control of the lower layers (i.e., more privileged layers) of the system to implicitly gain control of the upper layers (i.e., less privileged layers, such as Ring3). Two successful attempts to implement this concept are described below.

7.1.1 SubVirt. SubVirt was implemented as a proof-of-concept using Microsoft Virtual PC and VMware Workstation, which are host-based VMMs [80]. SubVirt modifies the boot sequence of the host so that the VMBR is loaded first, and then the target (i.e., the guest VM) in a deprivileged state is loaded on top of it. Afterwards, the rootkit is executed without the target being aware of

its existence. In addition to the VMBR module, SubVirt installs a malicious VM by leveraging the existing host OS. This means that the target cannot detect the execution of the malicious VM, and so it can attack, extract information or disrupt the activity of the target system.

Although SubVirt demonstrated the potential of VMM as a malicious attacker, it was released only as a proof-of-concept due to some limitations mentioned below:

- Requires changes to the host OS kernel, and in some cases to the hypervisor code, which limits portability.
- Requires privileges in the host system to modify the boot sequence so that VMBR can be loaded before the target system. Moreover, the OS kernel in modern platforms does not necessarily have the ability to modify boot sectors due to *Secure Boot* [143][64], which verifies each phase of the initialization with a cryptographic key.

These limitations can be mitigated, but not completely removed, by taking advantage of hardware-assisted virtualization (see Section 2.1). The nested virtualization support on the hardware makes it possible for a code that executes on the host OS (or a hypervisor) to virtualize such running OS/hypervisor itself thereby transforming it to a guest. The next two examples accomplish just that.

7.1.2 The Blue Pill. Hardware-assisted virtualization introduced a fundamental change in the x86 protection model by adding one privilege level below Ring 0 (often called "Ring -1") to be used by the VMM. The *Blue Pill* is a proof-of-concept rootkit created by Joanna Rutkowska [129] [130] and presented at the Black Hat conference in 2006 [14]. It is one of the first VMBRs that successfully achieved this privilege level. The Blue Pill is able to start from a running system and transform the host OS into a guest under its control, and was originally implemented using AMD64 Secure Virtual Machine (SVM) extensions. These extensions include a new set of instructions to support execution in a virtualized environment [2]. The rootkit can be installed on a running system and takes advantage of the new instructions to prepare the necessary control structures for the virtualized instance of the target. For example, the `VMRUN` instruction can be used to transition the target OS to execute in guest mode.

7.1.3 Vitriol. Vitriol was presented at the same conference as the Blue Pill [167]. It was implemented to target Mac OS X as a Loadable Kernel Extension to detect and enable hardware virtualization support, and then to de-privilege the target system to run as a guest. It also installs a handler for VM Exit events (i.e., the *on_vm_exit()* function). This handler can be designed to perform malicious activities, such as escalating privileges for a specific process or monitoring I/O operations. As opposed to the Blue Pill, which is based on AMD SVM extensions, Vitriol is based on the Intel VT-x technology.

These examples pose a recurrent challenge, which is the need for VMBR detection as a countermeasure. This is an important issue and will be discussed in the next subsection.

7.2 The issue of VMBR detection

After Blue Pill and Vitriol were made public, it was clear that VMBRs were not just a theoretical risk. In addition, statements made by Rutkowska when introducing the Blue Pill[6] and amplified by the media[7] started a debate in the security community. On one side, Rutkowska expressed that, although the Blue Pill was only a proof-of-concept, it would be possible to develop 100%

[6] "Over the past few months I have been working on a technology code-named Blue Pill, which is just about that - creating 100% undetectable malware, which is not based on an obscure concept..." [129].

[7] "A security researcher with expertise in rootkits has built a working prototype of new technology that is capable of creating malware that remains "100 percent undetectable," even on Windows Vista x64 systems." [95].

undetectable VMBRs [128]. On the other side, more skeptical researchers believed that the effort needed to develop such a VMBR was too large compared to the effort needed to develop a reliable detection tool[8].

Early discussions about detection of VMBRs were brought up by some of the creators themselves. Rutkowska noted that a distinction should be made between detecting when a system is running under the control of a VMM, which includes any legitimate use of virtualization (malware authors can be interested in finding out to prevent any further analysis, or to fingerprint a target system), and detecting a malicious VMBR. The following subsections discuss these two issues separately.

7.2.1 Detecting virtualized environments. Transparency in virtual environments was not considered as a top priority in the early days of x86 virtualization (and it is not considered a security objective of the VT extensions by Intel). As security of VMMs started to catch the interest from researchers, hiding the fact that a system is running under a hypervisor grew in importance [50]. Some of the possible ways to detect VMMs are discussed below:

Explicit communication from the VMM. There are a number of ways that an x86-based VMM can communicate information to guest VMs. Some of these are provided by the architecture itself or defined in standards, such as Desktop Management Interface (DMI) or System Management BIOS (SMBIOS):

- The `CPUID` instruction can be used by the VMM to communicate to a guest VM. In addition, the `CPUID` range 0x40000000-0x400000FF (options that can be configured by software) is sometimes used to pass information about the hypervisor to guest VMs. Although this is a feature used by the hypervisor, it also provides an easy way to detect virtualization usage [67].
- Linux guests can use tools like `dmidecode` to obtain information exposed in the DMI table. This approach is vendor agnostic [39].
- VMware provides a specific purpose I/O port 0x5658 to perform various tasks. If this port is read (using the `IN` instruction) after assigning some values to EAX, EBX, ECX and EDX registers, then the EBX register will be assigned the "magic" value 0x564D5868 to indicate the presence of a VMM [148].

Differences in results from specific CPU instructions. Omella described a number of instructions that give different results when they are executed in a non-virtualized environment versus a virtualized environment [100]. Those instructions provide information on internal structures of the system under test and can even be executed from a user (Ring3) application. For example, the instructions `SIDT`, `SGDT` and `SLDT` store the contents of the Interrupt Descriptor Table Register, the Global Descriptor Table Register, and the Local Descriptor Table Register, respectively, in memory. It has been observed that the resulting values written to memory can be examined to distinguish a virtualized environment. This distinction is possible because non-virtualized OSs initialize those values much earlier in the boot sequence, and therefore they use lower addresses than the ones used by a hypervisor. Prior to publication of her work on The Blue Pill, Rutkowska had implemented a small tool (appropriately named "The Red Pill"), which uses the `SIDT` instruction for this purpose

[8]A high point of the debate was reached when Thomas Ptacek from Matasano Security, Nate Lawson from Root Labs and Peter Ferrie from Symantec issued a public challenge to Rutkowska: She would be provided with two laptops, infect one of them, and then allow the challengers to execute their detection software on both of them to try to identify the infected system. If no infection was detected, it would be considered a win for Rutkowska and she would be able to keep the equipment [113][97]. In the end, the challenge was not pursued further due to disagreements on the conditions (Rutkowska set as one of the conditions that her development team be paid a fee for the resources involved in refining the rootkit code in order to be ready for the challenge, which Ptacek and the rest of the challengers deemed unreasonable [96]). Still, the VMBR detection debate remains open and research in the area is still being carried out.

[127]. However, Quist demonstrated that this method led to a lot of false positives when used on multiprocessor systems due to each processor having its own Interrupt Descriptor Table [115]. Instead of using `SIDT`, Quist proposed using the `SLDT` instruction having observed in experiments that the Local Descriptor Table (LDT) does not vary between single processor and multiprocessor systems.

Timing differences. The use of virtualization introduces overhead in a system, thus operations such as CPU transition between execution of VMM, guests, interrupt handling, memory management, and so forth, require some extra processing compared to native (i.e., non virtualized) execution. Although technologies such as hardware-assisted virtualization are narrowing this performance gap, a measurable difference still exists. Intel x86 processors include a timestamp counter that can be used for benchmarking purposes. This counter starts at zero with each reset and increments every clock cycle. The `RDTSC` instruction (introduced with the Pentium processors) copies the contents of the timestamp counter into the EDX:EAX registers [68]. A detection mechanism could be devised to measure the number of clock cycles needed to execute an interceptable instruction on a native system, and then compare this value against the execution on a virtualized system. The virtualization overhead will be reflected by an increased number of cycles. There are other time information, such as Real Time Clock, Network Time Protocol, etc., to which similar principles could be applied [7]. However, many researchers are quick to point out that without a trusted baseline for comparison, applying these techniques to develop a reliable detection tool is difficult.

Translation Lookaside Buffers (TLB) Profiling. The TLB profiling technique was first discussed by Peter Ferrie from Symantec [49], which relies on mechanisms not under the direct control of the VMM. The technique described in Ferrie's work is as follows:

(1) A set of memory pages is loaded so that the TLB is filled with information on these pages.
(2) The time to access these pages is measured (now that they have corresponding entries in the TLB).
(3) Generate a hypervisor event using an interceptable instruction (Ferrie suggests `CPUID` since it is non-destructive to memory). This should cause some TLB entries to be flushed as a side effect.
(4) These pages are reloaded and the time is measured again. If there is a noticeable difference in the access time for some of the pages, this would likely mean that they were evicted from the TLB revealing the presence of a VMM.

Barbosa presented an update to the above research [10], and stated that this attack does not always work in AMD-based VMMs since they provide a way to control whether the TLB entries are flushed upon execution of a VMM interception. However, since there is only a limited number of entries, the processor still flushes TLB entries as needed as part of the replacement algorithm [10]. Therefore, although detection is possible depending on implementation-specific characteristics, it is not as general as first thought.

TLB Profiling without timing. Garfinkel *et al.* described another TLB profiling technique that does not use any timing information [50]. Their technique is based on computing the size of available TLB entries before and after intervention of a VMM. The reported size of the TLB will be smaller due to some entries taken by the VMM [1].

Barbosa's Counter based technique. Barbosa described a technique where a multi-core system is set up with two threads [10]. The first thread increments a counter and executes an instruction that causes a VM Exit, e.g., reading the Extended Feature Enable Register (EFER), while the second thread simply increments a counter. Note that the threads are synchronized, meaning that they start

counting at the same time. The difference in counter values serves as a timing measure that can be use to detect the existence of a VMM. In a virtualized scenario, the first thread causes a VM exit that must be handled before the thread continues and increments its own counter. In a non-virtualized scenario, the first threads executes much faster since there is no hypervisor intervention, and therefore the count difference between the first and the second thread will be much smaller.

Presence of emulated devices. Physical systems (i.e., printers, scanners, video hardware, etc.) can be configured in many different ways. A commercial VMM is not able to support every possible guest configurations, and thus only a limited number of emulated peripherals is presented to the guests. These peripherals can be detected indicating that the system is being executed in a virtualized environment. For example, malware programs look for known emulated devices provided by major VMMs to detect that they are running in a virtualized environment. The presence of a hypervisor serves as a good indicator that malware programs are being analyzed, and thus a common reaction by these malware is to not perform malicious activities once it detects that it is running in a guest OS.

Memory dump analysis techniques. This approach is different from the previous ones in the sense that it is executed from the system that hosts the VMM rather than from a guest VM. Intel VT-x and AMD-V technologies provide support for hardware-assisted virtualization through programming interfaces. Both interfaces include data structures used to store state information for each virtual processor running under control of a VMM. These structures are called Virtual Machine Control Structure for Intel VT-x [69] and Virtual Machine Control Block for AMD-V [6]). *Actaeon* is a tool developed at EURECOM France, which can perform forensic analysis of memory dumps from a system and detect these structures indicating the presence of a VMM [60]. The tool can detect parallel and nested guests (see Section 2.1) and has been implemented as an open source Volatility plugin [59].

The practicality and accuracy of the techniques above varies among them. If the VMM communicates this information on purpose through a documented API, these mechanisms would be the first choice, but as we will discuss in the following subsection, a VMM can have motivation to hide this information from the guest VMs, so speculative techniques like TLB profiling and timing based techniques could be used. These techniques can produce inaccurate results and require more technical skills. Detection of emulated devices will give false positives if the system actually uses matching physical devices. Techniques based on memory dump analysis are of practical interest only if the user has access to a memory dump file. This would place the technique more in the domain of forensic investigations.

Leveraging speculative execution. The recent draw of attention towards speculative execution culminated on researchers identifying different behaviors of speculation under a hypervisor (versus a bare-metal system) [137].

7.2.2 VMBR techniques to avoid detection. As with malware in general, VMBRs have an obvious motivation to avoid detection: Once they are detected, the target can perform actions to remove them from the system. This subsection discusses techniques that VMBRs can implement to thwart detection attempts.

Use of thin hypervisors. A hypervisor that implements only the basic/mandatory requirements to virtualize guest VMs is called a *Thin Hypervisor*. Many of the features that can be used to detect commercial VMMs are not needed in a VMBR. For example, SubVirt is based on commercial VMMs (e.g., VMware) but Vitriol and Blue Pill are basically developed as thin hypervisors. Figure 6 compares thick versus thin hypervisors. VMBRs based on thin hypervisors do not have to explicitly

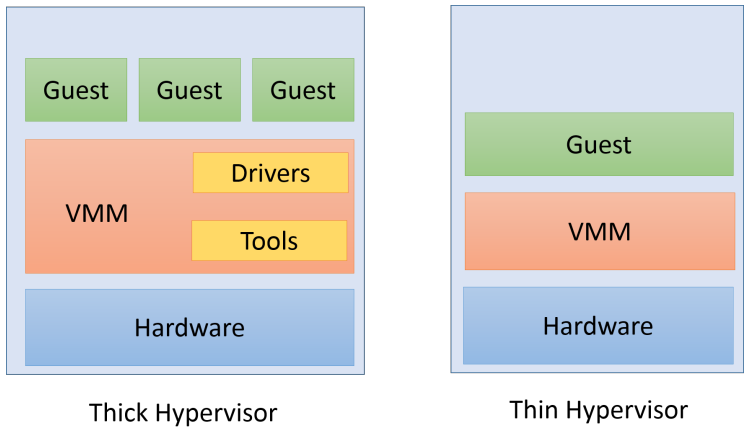

Fig. 6. Thick Hypervisors can include functions such as emulated devices, drivers and management tools. These additional functionalities make it easier to detect the existence of a hypervisor. Thin Hypervisors include only the basic support to virtualize an OS.

communicate to guest VMs that they are controlled by a VMM (e.g., using `CPUID`). Moreover, emulated devices that could be used for detection do not have to be presented to guest VMs. In the AMD-V technology, Extended Feature Enable Register (EFER), which is a Model-Specific Register (MSR), includes a bit for enabling SVM support. A guest VM can detect if virtualization is enabled by reading the register, but the instruction to read MSRs can be intercepted and manipulated by a VMBR [131]. Intel architecture also includes MSR EFER, but it does not include the virtualization enabling bit [69].

TimeStamp Counter (TSC) offset manipulation. As mentioned in Section 7.2.1, the `RDTSC` instruction can be used to detect timing differences on a virtualized system. However, this instruction can be intercepted by the VMM and the value that is reported to guest VMs can be manipulated. The Intel VT-x technology provides a simple way to intercept `RDTSC` instructions by setting a TSC_Offset field in the Virtual Machine Control Structure (VMCS) [69]. AMD also provides an equivalent mechanism [6]. Intel refers to this feature as Timestamp-counter offsetting (TSC offsetting for short), which allows a VMM software to specify a value (the TSC offset) to be added to the TSC when it is read by a guest VM. Although obtaining an appropriate offset is straightforward, some techniques such as instruction tracing [131] can be used to facilitate guest migration[9]. Migrating from a physical system to another do not impact guest operations that depend on the TSC value because the hypervisor is able to offset the time differences between the machines and hide it from the guest.

The "Blue Chicken" Technique. During Black Hat 2007, Rutkowska also presented a possible approach for VMBR to avoid timing-based detection called "Blue Chicken". This technique temporarily uninstalls the malicious hypervisor when an attempt to uncover its presence is detected. The VMBR then waits a specified amount of time and then reinstalls itself [131].

[9]A common case in datacenters is to have virtual machines move from a physical platform to another, due to performance or high-availability needs.

Although the techniques to detect malicious hypervisors are not perfect, it seems like the problem of creating a truly 100% undetectable VMBR is still open, as it seemed to be evidenced by the outcome of the Blue Pill challenge mentioned at the beginning of this section.

7.2.3 Techniques to differentiate between VMBR and legitimate uses of virtualization. The techniques mentioned in the previous subsections only detect the presence of a VMM. They are unable to distinguish a legitimate VMM from a malicious one. Therefore, it is important to consider detection of VMM versus VMBR separately. A false positive detection of VMBR can cause the system to take actions against a legitimate VMM, while a false negative will allow a VMBR to continue attacking the system.

A VMBR can manipulate elements such as `CPUID` output, BIOS reported serial number, and output from the `RDTSC` instruction. Therefore, detection must be based on elements that the VMM cannot control. The following techniques rely on low-level elements of the system that can only be controlled by a VMBR.

Chipset based detection. The *DeepWatch* project was presented as a proof-of-concept by Intel at Black Hat USA 2008 [17], which is based on an embedded microcontroller included as a part of the Memory Controller Hub (i.e., North Bridge) in the Intel Hub Architecture. This microcontroller executes Intel signed firmware (stored in SPI Flash) and has DMA capabilities, which can be programmed to directly access the system memory bypassing protection and privilege mechanisms. Once it gains access to the system memory, it is able to scan for signatures and potentially remove malicious code as well as verify the integrity of VMMs it finds.

SMM based detection. System Management Mode (SMM) was introduced with the Intel 80386SL processor in 2001 [94], and it is now available in all Intel x86 processors. SMM allows the CPU to execute special-purpose code for platform specific functions, such as power management or Original Equipment Manufacturer (OEM) code. This is transparent to the OS, which is suspended during SMM execution. A System Management Interrupt (SMI) needs to be generated to switch the processor to this execution mode (more details on SMM can be found in [15]). Afterwards, the current state of the CPU is saved and an SMI handler is invoked. The SMM code resides in a separate memory address space (SMRAM) that cannot be accessed in real or protected mode. It is also unavailable to the OS and VMMs. Although the example discussed in the following subsection is slightly dated (its implementation is based on earlier versions of the SMRAM protection mechanisms and it does not take into account the restrictions that can be imposed by an IOMMU), it is a good example of how SMM can be leveraged for security purposes.

The *HyperCheck* project proposed an SMM-based mechanism to monitor the integrity of an installed VMM [154]. A remote system will monitor a target system and communication between the monitor and the target will be achieved via Ethernet. The solution adds the following elements to the platform:

- A Network Interface Card (NIC) that will be used to transmit the collected information about the system to an outside monitoring system.
- An SMM code module that will read the system memory using DMA, and will drive the data transmit through the NIC.
- An SMM code module to read the CPU state (stored in SMRAM).
- An analysis module to be installed on the monitoring system.

When the target system is initialized, a copy of the system memory is sent to the monitor to serve as a reference. Subsequent snapshots will be compared to this reference copy and discrepancies are reported. In addition, the CPU registers are verified for integrity since a rootkit could modify these

registers, e.g., IDTR that points to the Interrupt Descriptor Table, and make it point to a malicious copy.

8 FUTURE: RESEARCH TRENDS AND GAPS

While it is very hard to predict the future, the amount of new side-channels vulnerabilities recently uncovered [37] [33] [34] [35] [36] [26] [28] [27] [25] [29] [32] [30] [31] is a good indication that the impacts on the virtualized environments are still to be fully understood (issues [31] and [28] are specifically against virtualization). There is a slight gap between demonstrations in the academia and the leverage by real attacks, but that gap is expected to shorten as more researchers jump into the topic of exploring the issues in different environments.

Additionally, competitions like Pwn2Own [150] are offering (money) prizes for hypervisor escape issues (and researchers have been fairly successful in finding and exploiting vulnerabilities with that end).

Microsoft openness in discussing security internals of Hyper-V [116] [74] (and how their own teams are bypassing mitigations) is also setting a trend in the industry, and hopefully the academics will leverage the learnings to once again move the needle on the state-of-the-art in mitigations in this area.

9 CONCLUSIONS

Virtualization technologies have brought immense benefits in terms of resource utilization and ease of management. However, these benefits have come at the cost of more complexity from the security standpoint. This is of increasing relevance now that cloud computing is easily available to the general public and more and more sensitive information is stored in cloud infrastructures. Even though virtualization intends to provide isolation, as long as the VMM and guest VMs reside in the same physical host, some sharing of resources among them will take place and solutions designed without taking into account their potential use in virtualized environments are at higher risk of introducing security vulnerabilities. At the same time, more work is needed on the the challenges that apply to both standalone and virtualized worlds (e.g., side channels, timing vulnerabilities, code bugs, etc.). As such, security should not be ignored when designing and implementing new approaches to virtualization. As more support for security technologies is implemented at the hardware level (encryption, key management, isolation of resources, etc.) virtualization technologies will benefit from this, and as a result, the final users.

REFERENCES

[1] Keith Adams. 2007. BluePill detection in two easy steps. Retrieved May 10, 2016 from http://x86vmm.blogspot.mx/2007/07/bluepill-detection-in-two-easy-steps.html
[2] AMD. 2005. AMD64 Virtualization Codenamed "Pacifica" Technology Secure Virtual Machine Architecture Reference Manual. Retrieved May 10, 2016 from http://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf
[3] AMD. 2016. AMD Secure Encrypted Virtualization (SEV). Retrieved Nov 18, 2018 from https://developer.amd.com/sev/
[4] AMD. 2018. AMD Processor Security Updates. Retrieved Nov 21, 2018 from https://www.amd.com/en/corporate/security-updates
[5] AMD. 2018. *White Paper: SOFTWARE TECHNIQUES FORMANAGING SPECULATION ON AMD PROCESSORS.*
[6] AMD Inc. 2015. *AMD64 Architecture Programmer's Manual Volume 2: System Programming.* Number 24593-3.25.
[7] Zachary Amsden. 2010. Timekeeping Virtualization for X86-Based Architectures. Retrieved May 10, 2016 from https://www.kernel.org/doc/Documentation/virtual/kvm/timekeeping.txt
[8] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. 2014. Fine grain Cross-VM Attacks on Xen and VMware are possible! *IACR Cryptology ePrint Archive* (2014).
[9] ARM. 2017. TrustZone – Arm. https://www.arm.com/products/security-on-arm/trustzone

[10] Edgar Barbosa. 2007. Detection of hardware virtualization rootkits. Retrieved May 10, 2016 from https://www.coseinc.com/en/index.php?rt=download&act=publication&file=Detecting_virtualized_hardware_rootkits.ppt.pdf
[11] Antonio Barresi, Kaveh Razavi, Mathias Payer, and Thomas R. Gross. 2015. CAIN: Silently Breaking ASLR in the Cloud. In *Proceedings of the 9th USENIX Conference on Offensive Technologies* (Washington, D.C.) *(WOOT'15)*. USENIX Association, Berkeley, CA, USA, 13–13. http://dl.acm.org/citation.cfm?id=2831211.2831224
[12] Adam Belay, Andrea Bittau, Ali Mashtizadeh, David Terei, David Mazières, and Christos Kozyrakis. 2012. Dune: Safe User-level Access to Privileged CPU Features. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. USENIX, Hollywood, CA, 335–348. https://www.usenix.org/conference/osdi12/technical-sessions/presentation/belay
[13] Daniel J Bernstein. 2005. Cache-timing attacks on AES.
[14] blackhat.com. 2006. Black Hat USA 2006 Speakers. Retrieved May 10, 2016 from https://www.blackhat.com/html/bh-usa-06/bh-usa-06-speakers.html#Rutkowska
[15] d0nand0n BSDaemon, coideloko. 2008. System Management Mode Hacks. Retrieved Oct, 16, 2018 from http://phrack.org/issues/65/7.html#article
[16] Edouard Bugnion, Jason Nieh, and Dan Tsafrir. 2017. *Hardware and Software Support for Virtualization.* Morgan & Claypool Publishers.
[17] Yuriy Bulygin and David Samyde. 2008. Chipset based approach to detect virtualization malware. *BlackHat Briefings USA* (2008).
[18] Ramaswami Chandramouli. 2014. *DRAFT NIST Special Publication 800-125-A. Security Recommendations for Hypervisor deployment.* National Institute of Standards and Technology, Gaithersburg, MD.
[19] 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 (March 2008), 2–13. https://doi.org/10.1145/1353536.1346284
[20] CITRIX. 2018. VM snapshots. Retrieved Nov 18, 2018 from https://docs.citrix.com/en-us/xenserver/current-release/dr/snapshots.html
[21] Robert R. Collins. 1997. Intel's System Management Mode. Retrieved Sep 18, 2016 from http://www.drdobbs.com/embedded-systems/undocumented-corner/184410120
[22] Louis Columbus. 2015. Roundup Of Cloud Computing Forecasts And Market Estimates Q3 Update, 2015. Retrieved Sep 27, 2015 from http://www.forbes.com/sites/louiscolumbus/2015/09/27/roundup-of-cloud-computing-forecasts-and-market-estimates-q3-update-2015/
[23] Lee Copeland. 2002. Checkpoint and Restart. Retrieved Nov 18, 2018 from https://www.computerworld.com/article/2588055/disaster-recovery/checkpoint-and-restart.html
[24] Intel Corp. 2018. Intel Software Guard Extensionsl. Retrieved Oct 18, 2018 from https://software.intel.com/en-us/sgx
[25] Intel Corporation. 2018. Intel Bounds Check Bypass Store (CVE-2018-3693). Retrieved August 20, 2019 from https://01.org/security/advisories/intel-oss-10002
[26] Intel Corporation. 2018. Intel CPU Speculative Side-channel L1 Terminal Fault: OS/SMM (CVE-2018-3620). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html
[27] Intel Corporation. 2018. Intel CPU Speculative Side-channel L1 Terminal Fault: SGX (CVE-2018-3615). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html
[28] Intel Corporation. 2018. Intel CPU Speculative Side-channel L1 Terminal Fault: VMM (CVE-2018-3646). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html
[29] Intel Corporation. 2018. Intel Lazy FP State Restore (CVE-2018-3665). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00145.html
[30] Intel Corporation. 2018. Rogue System Register Read (RSRE) CVE-2018-3640). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00115.html
[31] Intel Corporation. 2018. Speculative Execution and Indirect Branch Prediction Side Channel (CVE-2017-5753, CVE-2017-5754, CVE-2017-5715). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00088.html
[32] Intel Corporation. 2018. Speculative Store Bypass (SSB) (CVE-2018-3639). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00115.html
[33] Intel Corporation. 2019. Intel Microarchitectural Data Sampling Uncacheable Memory (MDSUM) (CVE-2019-11091). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00233.html
[34] Intel Corporation. 2019. Intel Microarchitectural Fill Buffer Data Sampling (MFBDS) (CVE-2018-12130). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00233.html
[35] Intel Corporation. 2019. Intel Microarchitectural Load Port Data Sampling (MLPDS) (CVE-2018-12127). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00233.html

[36] Intel Corporation. 2019. Intel Microarchitectural Store Buffer Data Sampling (MSBDS) (CVE-2018-12126). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00233.html
[37] Intel Corporation. 2019. Intel Microprocessor Memory Mapping Advisory (CVE-2019-0162). Retrieved August 20, 2019 from https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00238.html
[38] Microsoft Corporation. 2017. Hypervisor Code Integrity Elevation of Privilege Vulnerability. Retrieved August 20, 2019 from https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-0193
[39] Alan Cox and Jean Delvare. 2018. dmidecode(8)-Linux man page. Retrieved Oct 18, 2018 from https://linux.die.net/man/8/dmidecode
[40] R.J. Creasy. 1981. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development* 25, 5 (Sept. 1981), 483–490.
[41] Ash de Zylva. 2016. Windows 10 Device Guard and Credential Guard Demystified. Retrieved Jan 22, 2017 from https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/
[42] Alan M. Dunn, Michael Z. Lee, Suman Jana, Sangman Kim, Mark Silberstein, Yuanzhong Xu, Vitaly Shmatikov, and Emmett Witchel. 2012. Eternal Sunshine of the Spotless Machine: Protecting Privacy with Ephemeral Channels. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, USA) *(OSDI'12)*. USENIX Association, Berkeley, CA, USA, 61–75. http://dl.acm.org/citation.cfm?id=2387880.2387887
[43] Drew Eckhardt. 1992. chroot(2) Linux Programmer's Manual. Retrieved May 10, 2016 from http://man7.org/linux/man-pages/man2/chroot.2.html
[44] Jake Edge. 2015. A seccomp overview. Retrieved Aug, 20, 2017 from https://lwn.net/Articles/656307/
[45] Endgame. 2018. Detecting Spectre And Meltdown Using Hardware Performance Counters. Retrieved August 20, 2019 from https://www.endgame.com/blog/technical-blog/detecting-spectre-and-meltdown-using-hardware-performance-counters
[46] Herbert Bos Erik Bosman, Kaveh Razavi and Cristiano Giuffrida. 2016. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. (2016).
[47] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2016. Jump over ASLR: Attacking branch predictors to bypass ASLR. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*. IEEE, 1–13.
[48] Yuval Yarom Fangfei Liu, Qian Ge et al. 2016. CATalyst: Defeating Last-Level Cache Side Channel Attacks in Cloud Computing. (2016).
[49] Peter Ferrie. 2007. Attacks on Virtual Machine Emulators. Retrieved May 10, 2016 from http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf
[50] Tal Garfinkel, Keith Adams, Andrew Warfield, and Jason Franklin. 2007. Compatibility is Not Transparency: VMM Detection Myths and Realities. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems* (San Diego, CA) *(HOTOS'07)*. USENIX Association, Berkeley, CA, USA, Article 6, 6 pages. http://dl.acm.org/citation.cfm?id=1361397.1361403
[51] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. 2003. Terra: A Virtual Machine-based Platform for Trusted Computing. *SIGOPS Oper. Syst. Rev.* 37, 5 (Oct. 2003), 193–206. https://doi.org/10.1145/1165389.945464
[52] Tal Garfinkel and Mendel Rosenblum. 2003. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *In Proc. Network and Distributed Systems Security Symposium*. 191–206.
[53] Jason Geffner. 2015. VENOM: Virtualized Environment Neglected Operations Manipulation. Retrieved May 10, 2016 from http://venom.crowdstrike.com/
[54] Jason Geffner. 2015. VENOM Vulnerability Details. Retrieved May 10, 2016 from http://blog.crowdstrike.com/venom-vulnerability-details/
[55] Antonios Gkortzis, Stamatia Rizou, and Diomidis Spinellis. 2016. An empirical analysis of vulnerabilities in virtualization technologies. In *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. IEEE, 533–538.
[56] Robert P. Goldberg. 1973. Architectural principles for virtual computer systems. (1973), 22–27.
[57] Michael Grace, Zhi Wang, Deepa Srinivasan, Jinku Li, Xuxian Jiang, Zhenkai Liang, and Siarhei Liakh. 2010. Transparent protection of commodity os kernels using hardware virtualization. In *Security and Privacy in Communication Networks*. Springer, 162–180.
[58] Ben Gras, Kaveh Razavi, Erik Bosman, Cristiano Giuffrida, and Herbert Bos. 2017. ASLR and Cache: Practical Cache Attacks on MMU from JavaScript. In *NDSS2017*.
[59] Mariano Graziano. 2013. Actaeon. Retrieved May 10, 2016 from http://www.s3.eurecom.fr/tools/actaeon/
[60] Mariano Graziano, Andrea Lanzi, and Davide Balzarotti. 2013. Hypervisor Memory Forensics. In *Symposium on Research in Attacks, Intrusion, and Defenses (RAID)* (Saint Lucia) *(RAID 13)*. Springer.
[61] Trusted Computing Group. 2018. Trusted Computing Group. Retrieved Oct 18, 2018 from https://trustedcomputinggroup.org/

[62] William Hayles. 2015. New Technology From Intel Will Help Solve The Cloud's Noisy Neighbor Problem. Retrieved Jan 8, 2017 from https://blog.outscale.com/us/new-technology-from-intel-will-help-solve-the-clouds-noisy-neighbor-problem/

[63] Matt Helsley. 2009. LXC: Linux container tools: Tour and set up the new container tools called Linux Containers. *IBM developerWorks* (2009).

[64] Intel. 2013. Intel Hardware-based Security Technologies for Intelligent Retail Devices. Retrieved Nov 18, 2018 from https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/security-technologies-4th-gen-core-retail-paper.pdf

[65] Intel. 2018. *White Paper: Intel Analysis of Speculative Execution Side Channels.* Number 326983-001.

[66] Intel Corporation. 2013. *Mobile 3rd Generation Intel® Core™ Processor Family, Mobile Intel® Pentium® Processor Family, and Mobile Intel® Celeron® Processor Family: Datasheet - Volume 2 of 2.* Number 326769-004.

[67] Intel Corporation. 2014. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture.* Number 253665-051US.

[68] Intel Corporation. 2014. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2: Instruction Set Reference.* Number 325383-051US.

[69] Intel Corporation. 2014. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3: System Programming Guide.* Number 325384-051US.

[70] Intel Corporation. 2014. *Intel® Virtualization Technology for Directed I/O: Architecture Specification.* Number D51397-007.

[71] Intel Corporation. 2015. *5th Generation Intel® Core™ Processor Family, Intel® Core™ M Processor Family, Mobile Intel® Pentium® Processor Family, and Mobile Intel® Celeron® Processor Family: Datasheet - Volume 2 of 2.* Number 330835-003.

[72] Intel Corporation. 2015. *Intel® Trusted Execution Technology: Software Development Guide.* Number 315168-012.

[73] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. 2010. Stealthy Malware Detection and Monitoring Through VMM-based &Ldquo;Out-of-the-box&Rdquo; Semantic View Reconstruction. *ACM Trans. Inf. Syst. Secur.* 13, 2, Article 12 (March 2010), 28 pages. https://doi.org/10.1145/1698750.1698752

[74] Nicolas Joly and Joe Bialek. 2019. A Dive in to Hyper-V Architecture & Vulnerabilities. Retrieved August 20, 2019 from http://i.blackhat.com/us-18/Wed-August-8/us-18-Joly-Bialek-A-Dive-in-to-Hyper-V-Architecture-and-Vulnerabilities.pdf

[75] Seth Juarez. 2015. Windows 10 Virtual Secure Mode with David Hepkin. Retrieved Jan 21, 2017 from https://channel9.msdn.com/Blogs/Seth-Juarez/Windows-10-Virtual-Secure-Mode-with-David-Hepkin

[76] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. 2009. Emulating Emulation-resistant Malware. In *Proceedings of the 1st ACM Workshop on Virtual Machine Security* (Chicago, Illinois, USA) *(VMSec '09)*. ACM, New York, NY, USA, 11–22. https://doi.org/10.1145/1655148.1655151

[77] David Kaplan, Jeremy Powell, and Tom Woller. 2016. *AMD Memory Encryption.*

[78] 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

[79] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 361–372. https://doi.org/10.1109/ISCA.2014.6853210

[80] Samuel T. King, Peter M. Chen, Yi-Min Wang, Chad Verbowski, Helen J. Wang, and Jacob R. Lorch. 2006. SubVirt: Implementing malware with virtual machines. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. Institute of Electrical and Electronics Engineers, Inc., Oakland, CA, 314–327. http://research.microsoft.com/apps/pubs/default.aspx?id=67911

[81] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*.

[82] Kostya Kortchinsky. 2009. Cloudburst: A VMware guest to host escape story. Retrieved May 10, 2016 from https://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloudburst-SLIDES.pdf

[83] Brian Krzanich. 2018. Advancing Security at the Silicon Level. Retrieved Nov 21, 2018 from https://newsroom.intel.com/editorials/advancing-security-silicon-level/

[84] kuo Lang Tseng. 2015. Intel Kernel Guard Technology. Retrieved May 10, 2016 from https://01.org/intel-kgt

[85] McAfee Labs. 2015. McAfee Labs Threats Report May 2015. pp.37. , 37 pages. Retrieved May 10, 2016 from http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2015.pdf

[86] Tianlin Li, Yaohui Hu, Ping Yang, and Kartik Gopalan. 2015. Privacy-preserving Virtual Machine. In *Proceedings of the 31st Annual Computer Security Applications Conference* (Los Angeles, CA, USA) *(ACSAC 2015)*. ACM, New York, NY, USA, 231–240. https://doi.org/10.1145/2818000.2818044

[87] Siarhei Liakh, Michael Grace, and Xuxian Jiang. 2010. Analyzing and Improving Linux Kernel Memory Protection: A Model Checking Approach. In *Proceedings of the 26th Annual Computer Security Applications Conference* (Austin, Texas, USA) *(ACSAC '10)*. ACM, New York, NY, USA, 271–280. https://doi.org/10.1145/1920261.1920301

[88] David Lie and Lionel Litty. 2010. Using hypervisors to secure commodity operating systems. In *Proceedings of the fifth ACM workshop on Scalable trusted computing*. ACM, 11–20.

[89] Yutao Liu, Tianyu Zhou, Kexin Chen, Haibo Chen, and Yubin Xia. 2015. Thwarting Memory Disclosure with Efficient Hypervisor-enforced Intra-domain Isolation. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. ACM, New York, NY, USA, 1607–1619. https://doi.org/10.1145/2810103.2813690

[90] A. Theodore Markettos, Colin Rothwell, Brett F. Gutstein, Allison Pearce, Peter G. Neumann, Simon W. Moore, and Robert N. M. Watson. 2019. Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. https://www.ndss-symposium.org/ndss-paper/thunderclap-exploring-vulnerabilities-in-operating-system-iommu-protection-via-dma-from-untrustworthy-peripherals/

[91] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. 2014. Confidentiality Issues on a GPU in a Virtualized Environment. In *Financial Cryptography and Data Security*. Springer, 119–135.

[92] Arif Mohamed. 2009. A history of cloud computing. Retrieved May 10, 2016 from http://www.computerweekly.com/feature/A-history-of-cloud-computing

[93] Soo-Jin Moon, Vyas Sekar, and Michael K. Reiter. 2015. Nomad: Mitigating Arbitrary Cloud Side Channels via Provider-Assisted Migration. In *ACM Conference on Computer and Communications Security*.

[94] Scott Mueller and Mark Edward Soper. 2001. Microprocessor Types and Specifications. Retrieved Sep 18, 2016 from http://www.informit.com/articles/article.aspx?p=130978&seqNum=27

[95] Ryan Naraine. 2006. Blue Pill Prototype Creates 100% Undetectable Malware. Retrieved May 10, 2016 from http://www.eweek.com/c/a/Windows/Blue-Pill-Prototype-Creates-100-Undetectable-Malware

[96] Ryan Naraine. 2007. Blue Pill hacker challenge update: It's a no-go. Retrieved May 10, 2016 from http://www.zdnet.com/article/blue-pill-hacker-challenge-update-its-a-no-go/

[97] Ryan Naraine. 2007. Rutkowska faces '100% undetectable malware' challenge. Retrieved May 10, 2016 from http://www.zdnet.com/article/rutkowska-faces-100-undetectable-malware-challenge/

[98] Khang T Nguyen. 2016. Introduction to Cache Allocation Technology in the Intel® Xeon® Processor E5 v4 Family. Retrieved Jan 8, 2017 from https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[99] NVD. 2007. CVE-2007-2455. Retrieved May 10, 2016 from https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-2455 National Vulnerability Database.

[100] Alfredo Andres Omella. 2006. Methods for virtual machine detection. (2006).

[101] Oracle. 2019. About VirtualBox. Retrieved Aug 4, 2019 from https://www.virtualbox.org/wiki/VirtualBox

[102] Tavis Ormandy. 2007. An empirical study into the security exposure to hosts of hostile virtualized environments.

[103] PCI-SIG. 2010. *PCI Express® Base Specification Revision 3.0*.

[104] 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*. ACM, 305–316.

[105] 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* (Hangzhou, China) *(Cloud Computing '13)*. ACM, New York, NY, USA, 3–10. https://doi.org/10.1145/2484402.2484406

[106] Neil Peterson. 2016. Windows Containers. Retrieved Aug 22, 2016 from https://msdn.microsoft.com/en-us/virtualization/windowscontainers/about/about_overview

[107] Global Platform. 2016. GlobalPlatform made simple guide: Trusted Execution Environment (TEE) Guide. Retrieved May 10, 2016 from http://www.globalplatform.org/mediaguidetee.asp

[108] KVM Project. 2015. KVM FAQ. Retrieved Aug, 20, 2018 from https://www.linux-kvm.org/page/FAQ

[109] Xen Project. 2016. Hvmloader. Retrieved Dec 31, 2017 from https://wiki.xenproject.org/wiki/Hvmloader

[110] Xen Project. 2018. Driver Domain. Retrieved Oct 18, 2018 from https://wiki.xenproject.org/wiki/Driver_Domain

[111] Xen Project. 2018. Xen Project Software Overview. Retrieved Aug 4, 2019 from https://wiki.xenproject.org/wiki/Xen_Project_Software_Overview

[112] Niels Provos, Markus Friedl, and Peter Honeyman. 2003. Preventing Privilege Escalation. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12* (Washington, DC) *(SSYM'03)*. USENIX Association, Berkeley, CA, USA, 16–16. http://dl.acm.org/citation.cfm?id=1251353.1251369

[113] Thomas Ptacek. 2007. Joanna: We Can Detect BluePill. Let Us Prove It! Retrieved May 10, 2016 from https://web.archive.org/web/20070810120035/http://www.matasano.com/log/895/joanna-we-can-detect-bluepill-let-us-prove-it/

[114] Quarkslab. 2018. Introduction to Trusted Execution Environment: ARM's TrustZone. Retrieved Nov 18, 2018 from https://blog.quarkslab.com/introduction-to-trusted-execution-environment-arms-trustzone.html

[115] Danny Quist, Val Smith, and Offensive Computing. 2006. Detecting the presence of virtual machines using the local data table. (2006).

[116] Jordan Rabet. 2018. Hardening Hyper-V through offensive security research. Retrieved August 20, 2019 from https://i.blackhat.com/us-18/Thu-August-9/us-18-Rabet-Hardening-Hyper-V-Through-Offensive-Security-Research.pdf

[117] Sanaz Rahimi and Mehdi Zargham. 2010. Security Implications of Different Virtualization Approaches for Secure Cyber Architectures. In *Secure and Resilient Cyber Architectures Conference MITRE 2010*. (McLean, VA).

[118] Thorsten Holz Ralf Hund and Felix C. Freiling. 2009. Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms. In *Proceedings of the 18th USENIX Security Symposium (USENIX Security 09)*. USENIX, Montreal, Canada. https://www.usenix.org/conference/usenixsecurity09/technical-sessions/presentation/return-oriented-rootkits-bypassing

[119] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. 2016. Flip Feng Shui: Hammering a Needle in the Software Stack. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 1–18. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[120] Ryan Riley, Xuxian Jiang, and Dongyan Xu. 2007. An Architectural Approach to Preventing Code Injection Attacks. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '07)*. IEEE Computer Society, Washington, DC, USA, 30–40. https://doi.org/10.1109/DSN.2007.13

[121] Ryan Riley, Xuxian Jiang, and Dongyan Xu. 2008. Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection* (Cambridge, MA, USA) *(RAID '08)*. Springer-Verlag, Berlin, Heidelberg, 1–20. https://doi.org/10.1007/978-3-540-87403-4_1

[122] Matteo Riondatto. 2015. FreeBSD Handbook. Chapter 14:Jails. Retrieved May 10, 2016 from http://www.freebsd.org/doc/en/books/handbook/jails.html

[123] 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* (Chicago, Illinois, USA) *(CCS '09)*. ACM, New York, NY, USA, 199–212. https://doi.org/10.1145/1653662.1653687

[124] Dan Rosenberg. 2014. Reflections on Trusting TrustZone. Retrieved Nov 18, 2018 from https://www.blackhat.com/docs/us-14/materials/us-14-Rosenberg-Reflections-on-Trusting-TrustZone.pdf

[125] Mendel Rosenblum and Carl Waldspurger. 2011. I/O Virtualization. *Queue* 9, 11, Article 30 (Nov. 2011), 10 pages. https://doi.org/10.1145/2063166.2071256

[126] Mark Russinovich. 2015. Containers: Docker, Windows and Trends. Retrieved May 10, 2016 from https://azure.microsoft.com/en-us/blog/containers-docker-windows-and-trends/

[127] Joanna Rutkowska. 2004. Red Pill... or how to detect VMM using (almost) one CPU instruction. Retrieved May 10, 2016 from http://repo.hackerzvoice.net/depot_ouah/Red_%20Pill.html

[128] Joanna Rutkowska. 2006. The Blue Pill Hype. Retrieved May 10, 2016 from http://theinvisiblethings.blogspot.mx/2006/07/blue-pill-hype.html

[129] Joanna Rutkowska. 2006. Introducing Blue Pill. Retrieved May 10, 2016 from http://theinvisiblethings.blogspot.mx/2006/06/introducing-blue-pill.html

[130] Joanna Rutkowska. 2006. Subverting Vista Kernel for Fun And Profit. Retrieved May 10, 2016 from https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf

[131] Joanna Rutkowska and Alexander Tereshkin. 2007. IsGameOver() anyone. Retrieved May 10, 2016 from https://www.blackhat.com/presentations/bh-usa-07/Rutkowska/Presentation/bh-usa-07-rutkowska.pdf

[132] Joanna Rutkowska and Rafal Wojtczuk. 2018. Qubes Architecture Overview. Retrieved Oct 18, 2018 from https://www.qubes-os.org/doc/architecture/#key-architecture-features

[133] F.L. Sang, E. Lacombe, V. Nicomette, and Y. Deswarte. 2010. Exploiting an I/OMMU vulnerability. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*. 7–14. https://doi.org/10.1109/MALWARE.2010.5665798

[134] Fernand Lone Sang, Vincent Nicomette, and Yves Deswarte. 2011. I/O Attacks in Intel PC-based Architectures and Countermeasures. In *Proceedings of the 2011 First SysSec Workshop (SYSSEC '11)*. IEEE Computer Society, Washington, DC, USA, 19–26. https://doi.org/10.1109/SysSec.2011.10

[135] Vasily A. Sartakov and Rüdiger Kapitza. 2014. NV-Hypervisor: Hypervisor-Based Persistence for Virtual Machines. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN*

'14). IEEE Computer Society, Washington, DC, USA, 654–659. https://doi.org/10.1109/DSN.2014.64

[136] Daniel J. Scales, Mike Nelson, and Ganesh Venkitachalam. 2010. The Design of a Practical System for Fault-tolerant Virtual Machines. *SIGOPS Oper. Syst. Rev.* 44, 4 (Dec. 2010), 30–39. https://doi.org/10.1145/1899928.1899932

[137] Innokentii Sennovskii. 2018. Wake up Neo: detecting virtualization through speculative execution. Retrieved August 20, 2019 from https://2018.offzone.moscow/getfile/?bmFtZT0xNS0wMF9XYWtlX1VwX05lby5wZGYmSUQ9NDAy

[138] Fermin Serna. 2012. The info leak era on software exploitation. Retrieved August 20, 2019 from https://media.blackhat.com/bh-us-12/Briefings/Serna/BH_US_12_Serna_Leak_Era_Slides.pdf

[139] 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 Twenty-first ACM SIGOPS Symposium on Operating Systems Principles* (Stevenson, Washington, USA) *(SOSP '07)*. ACM, New York, NY, USA, 335–350. https://doi.org/10.1145/1294261.1294294

[140] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. 2011. Memory deduplication as a threat to the guest OS. In *Proceedings of the Fourth European Workshop on System Security*. ACM, 1.

[141] Medhi Talbi. 2017. Attacking a co-hosted VM: A hacker, a hammer and two memory modules. https://thisissecurity.stormshield.com/2017/10/19/attacking-co-hosted-vm-hacker-hammer-two-memory-modules/

[142] Yinqian Zhang Tianwei Zhang and Ruby B. Lee. 2016. CloudRadar: A Real-Time Side-Channel Attack Detection System in Clouds. (2016).

[143] UEFI. 2013. UEFI SECURE BOOT IN MODERN COMPUTER SECURITY SOLUTIONS. Retrieved Nov 18, 2018 from http://www.uefi.org/sites/default/files/resources/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf

[144] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-order Execution. In *Proceedings of the 27th USENIX Conference on Security Symposium* (Baltimore, MD, USA) *(SEC'18)*. USENIX Association, Berkeley, CA, USA, 991–1008. http://dl.acm.org/citation.cfm?id=3277203.3277277

[145] Venkatanathan Varadarajan, Thomas Ristenpart, and Michael Swift. 2014. Scheduler-based Defenses Against cross-VM Side-channels. In *Proceedings of the 23rd USENIX Conference on Security Symposium* (San Diego, CA) *(SEC'14)*. USENIX Association, Berkeley, CA, USA, 687–702. http://dl.acm.org/citation.cfm?id=2671225.2671269

[146] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2015. A Placement Vulnerability Study in Multi-tenant Public Clouds. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Washington, D.C.) *(SEC'15)*. USENIX Association, Berkeley, CA, USA, 913–928. http://dl.acm.org/citation.cfm?id=2831143.2831201

[147] VMware. 2006. Virtualization Overview. Retrieved May 10, 2016 from http://www.vmware.com/pdf/virtualization.pdf

[148] VMware. 2015. Mechanisms to determine if software is running in a VMware virtual machine. Retrieved May 10, 2016 from http://kb.vmware.com/kb/1009458

[149] VMware. 2018. Understanding VM snapshots in ESXi / ESX. Retrieved Nov 18, 2018 from https://kb.vmware.com/s/article/1015180

[150] VMware. 2019. VMware and Pwn2Own Vancouver 2019. Retrieved August 20, 2019 from https://blogs.vmware.com/security/2019/03/vmware-and-pwn2own-vancouver-2019.html

[151] VMware. 2019. VMware ESXi: The Purpose-Built Bare Metal Hypervisor. Retrieved Aug 4, 2019 from https://www.vmware.com/products/esxi-and-esx.html

[152] VMware. 2019. Workstation Pro. Retrieved Aug 4, 2019 from https://www.vmware.com/products/workstation-pro.html

[153] Carl Waldspurger and Mendel Rosenblum. 2012. I/O Virtualization. *Commun. ACM* 55, 1 (Jan. 2012), 66–73. https://doi.org/10.1145/2063176.2063194

[154] Jiang Wang, Angelos Stavrou, and Anup Ghosh. 2010. Hypercheck: A hardware-assisted integrity monitor. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 158–177.

[155] Zhu Wang, Tao Huang, and Sha Wen. 2012. A File Integrity Monitoring System Based on Virtual Machine. In *Proceedings of the 2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*. IEEE Computer Society, 653–655.

[156] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F Wenisch, and Yuval Yarom. 2018. *Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution.* Technical Report.

[157] Felix Wilhelm. 2016. PoC for breaking hypervisor ASLR using branch target buffer collisions. Retrieved Jan 15, 2017 from https://github.com/felixwilhelm/mario_baslr

[158] Rafal Wojtczuk. 2008. Subverting the Xen hypervisor. Retrieved May 10, 2016 from https://www.blackhat.com/presentations/bh-usa-08/Wojtczuk/BH_US_08_Wojtczuk_Subverting_the_Xen_Hypervisor.pdf

[159] Rafal Wojtczuk. 2014. Poacher turned gamekeeper: Lessons learned from eight years of breaking hypervisors. Retrieved May 10, 2016 from http://www.bromium.com/sites/default/files/wp-bromium-breaking-hypervisors-

wojtczuk.pdf

[160] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang. 2015. Toward a software-based network: integrating software defined networking and network function virtualization. *IEEE Network* 29, 3 (May 2015), 36–41. https://doi.org/10.1109/MNET.2015.7113223

[161] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. 2016. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 19–35. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/xiao

[162] Yuanzhong Xu, Weidong Cui, and M. Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *Security and Privacy (SP), 2015 IEEE Symposium on*. 640–656. https://doi.org/10.1109/SP.2015.45

[163] Lok-Kwong Yan, Manjukumar Jayachandra, Mu Zhang, and Heng Yin. 2012. V2E: Combining Hardware Virtualization and Software Emulation for Transparent and Extensible Malware Analysis. *SIGPLAN Not.* 47, 7 (March 2012), 227–238. https://doi.org/10.1145/2365864.2151053

[164] 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* (Raleigh, North Carolina, USA) *(CCS '12)*. ACM, New York, NY, USA, 305–316. https://doi.org/10.1145/2382196.2382230

[165] Yinqian Zhang and Michael K. Reiter. 2013. DüPpel: Retrofitting Commodity Operating Systems to Mitigate Cache Side Channels in the Cloud. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. ACM, New York, NY, USA, 827–838. https://doi.org/10.1145/2508859.2516741

[166] Yongbin Zhou and DengGuo Feng. 2005. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing.

[167] Dino A. Dai Zovi. 2006. Hardware Virtualization Rootkits. Retrieved May 10, 2016 from https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zovi.pdf