Thoughts on hypervisor-based virtualization threats and vulnerabilities

Pensamentos sobre ameaças e vulnerabilidades de virtualização baseadas no hipervisor

DOI:10.34117/bjdv8n4-460

Recebimento dos originais: 21/02/2022
Aceitação para publicação: 31/03/2022

João Vitor Assis Ribeiro
Bacharel em Ciência da Computação
Institution: Instituto de Criminalística, Polícia Civil do Distrito Federal. Brasília/DF
Brasil
Address: SPO Conjunto A, Lote 23, Complexo da PCDF .Parque da Cidade Sudoeste
Brasília DF
E-mail: joao.ribeiro@pcdf.df.gov.br

ABSTRACT
As vulnerability and threat analysis play a vital role in software security in an ever-increasing digital world of virtualized computer and information systems, it is paramount that key security concepts are understood and that crucial security practices are applied in order to safeguard these types of assets. For that, this work attempts to provide an insight at vulnerabilities and threats related to the hypervisor model of virtualization while also fomenting a discussion about the security demands and challenges that this technology brings.

Keywords: virtualization, security, vulnerability assessment.

RESUMO
À medida que a análise de ameaças e vulnerabilidades desempenha um papel fundamental na segurança de software em um mundo digital de sistemas de informação e computadores cada vez mais virtualizados, é essencial que conceitos-chave de segurança sejam compreendidos e que práticas cruciais de segurança sejam aplicadas para que se possa salvaguardar esses tipos de ativos. Dessa forma, este trabalho almeja fornecer um entendimento a respeito de vulnerabilidades e ameaças relacionadas ao modelo de virtualização utilizando hypervisores enquanto fomenta uma discussão a respeito de demandas e desafios de segurança que esse tipo de tecnologia traz.

Palavras-chave: virtualização, segurança, análise de vulnerabilidades.

1 INTRODUCTION
With the consolidation of virtualization and cloud computing comes an increasing demand to protect these types of assets on the digital world. Virtualized environments, its hardware infrastructure, management software as well as its hosted Information Systems
are all susceptible to being potential targets for attackers looking to perform attacks ranging from DoS - Denial of Service - to data theft, exfiltration, extortion and many others.

In that context, vulnerability assessment and threat analysis play a substantial role in the process of understanding how to protect these systems against the multitude of possible attack vectors in order to meet the demands of the industry the digital society.

Thus, this paper attempts to present an overhaul of some of the most pertinent aspects of documented hypervisor-based virtualization vulnerabilities in the past years while also providing insights on general virtualization security topics, in hope that these insights may be helpful for future virtualization software design and implementation.

This article is organized as follows. The first section provides a general review of virtualization concepts such as virtual machines and hypervisors (virtual machine monitors). The second section provides a conceptual introduction to virtualization security; The third section touches upon hypervisor or virtualization-awareness and detection, The fourth section presents a discussion about some of the virtualization vulnerabilities and threats, divided in topics pertaining to hyperjacking, VMBRs (Virtual Machine-based rootkits), VM DoS (Denial of Service), VM escape and other vulnerabilities. Lastly, the fifth section wraps up with some considerations regarding virtualization security.

Virtualization

Virtualization can be defined as the ability to emulate hardware via software [1]. OS virtualization allows a computer system (e.g., a single machine, a server or a cluster) to run multiple operating systems or multiple instances of a single OS, enabling a hardware infrastructure to host various applications that run in different OSs on a single platform.

The main enabler of virtualization in this context is the Virtual Machine Monitor (VMM) also known as the hypervisor. The hypervisor is in charge of the emulation of specific hardware specifications and configurations for the guest OSes [1].

As a software, the hypervisor also needs to meet three main, basic and minimum criteria according to Popek et al [2]:

1. Provide an (virtual) environment for applications identical to the original machine's;
2. Provide complete control over the system's resources;
3. Applications running in this environment have minimal performance decrease;

The second criterion is of special importance in the context of processor virtualization or more specifically, instruction virtualization, as some instructions can lead to unexpected or odd behavior if not properly emulated or handled by the virtualization architecture.

Non-virtualizable instructions

A non-virtualizable instruction can be defined as a sensitive instruction and a non-privileged instruction. A sensitive instruction is one that shows different behavior depending if its running on kernel mode or user mode, if it modifies machine resources or the state of the machine, or both. A non-privileged instruction as the name implies is an instruction that do not require CPU privileges to be executed or handled, i.e., an instruction executed in user mode that don't trap into kernel mode.

In order to meet the second criterion, it is important that the hypervisor is aware of any attempts made by the VM's kernel code or by user code running in the VM to modify the machine state or the system resources. Thus, these non-virtualizable instructions must be handled in a special way. There are several ways to accomplish this, depending on the type of virtualization. Some examples are: in full virtualization, by using interpretation and/or binary translation [3]; in paravirtualization, by using host OS assist [3]; and in hardware assisted virtualization, by using a trap-and-emulate model in hardware. The way non-virtualizable instructions are dealt with dictates the overall strategy used for virtualization and for virtualization software design and implementation.

Types of Virtualization

Concerning emulation, virtualization of Operating Systems can be classified in to three main types:

- **Full virtualization**: the guest OS is completely abstracted (decoupled) from the hardware by the virtualization layer [4]. Some implementation examples are: interpretation (usually low performance), binary translation and binary translation with direct execution of user level code.

- **Para-virtualization or OS Assisted Virtualization**: the guest OS is modified to function along with the VMM to interface virtualized hardware [5]. It comprises the modification of an OS kernel to replace non-virtualizable instructions with
hypercalls to the hypervisor [4]. Compatibility is not its strong point, since OSes that cannot be modified in a feasible manner are not supported.

- **Hardware Assisted Virtualization**: the hardware supports virtualized instructions without the need for translation [5], as privileged and sensitive calls are set to automatically trap into the VMM [4]. The privileged instructions are handled with a new CPU execution mode that allows the VMM to run on a ring below CPU Privilege Rings (ring 0 through ring 3 in the x86 architecture) [4].

- **OS-level Virtualization or Container Virtualization**: a method of virtualization in which the kernel section of the OS is modified in such a way as to provide isolated user space instances, each with its own set of binaries, libraries, drives and an overall software framework needed for the execution of its applications. In this mode, the OS is effectively virtualized.

- **Hybrid Virtualization**: combines paravirtualization and hardware-assisted virtualization [6]. The main idea is to get the most out of each technology, leveraging I/O efficiency and overall performance.

Figure 1 through 3 illustrates the first three types of virtualization.

Figs. 1-3: Types of virtualization schemes

Hypervisor types

There are conceptually two types of Hypervisors, Type 1 and Type 2. The first one is known as bare metal hypervisor, which runs directly on the platform hardware with the guest OS "on top" of it. The second type is also known as hosted Hypervisors, which run on a host OS as an application. Figure 2 illustrates these concepts.
Virtual Machines

A virtual machine (VM) is considered as an efficient, isolated duplicate of a real machine and can be defined as the environment created by the Hypervisor [2]. A VM executes software (one process or a full system) in the same way as a machine (i.e., hardware) in which the software was developed [7]. In the context of optimization and performance, VMs can be viewed as an execution method that combines the opportunities for compiler optimization with the advantages of interpretation [8]. The virtual machine is comprised of specification and configuration files and its execution is managed and monitored by the hypervisor.

By using VMs and virtualization, resource allocation and reallocation are made easier, as direct hardware manipulation is not mandatory. Therefore, a VM has virtual devices that provide the same functionality as a physical hardware and may provide additional benefits in terms of portability, manageability and security. However, while possibilities like access to controllers, storage sharing, and overall resource sharing introduce versatility and performance benefits, they come with their own security complexities.

Virtualization security overview

There are many factors revolving virtualization security: the physical servers the hypervisors and the virtual machines are running on, software within and outside the virtualized environments, security culture and so on. Research topics about VMs and virtualization security abound, as it is an active research area. In the scope of this paper the focus will be on security issues concerning virtual machines and OS virtualization or more specifically, vulnerabilities, threats and risks related to the hypervisor and the VM abstraction, with the focus given on discrepancies between virtualized and native. Container virtualization and other types of virtualization solutions will not be directly
covered in this survey, but key, general concepts that the technology is subject to still apply.

**Hypervisor security**

Virtualization allows a multitude of possibilities, like running multiple virtualized environments for different guest OSs (different VMs), and thus various applications (e.g., VPS Hosting, Cloud Computing, distributed applications). This technology come with potential security issues for environments where multiples VMs are running under the same hypervisor. For type 1 hypervisors, the attack surface area is usually smaller than that of type 2, because its code space is typically smaller. As for Type 2 Hypervisors, the attack surface is usually larger, since more code is being executed by the host OS running the VM. Also, one could argue that the type 2 hypervisor - and by consequence the VMs under its monitoring - is only as secure as the host OS it runs on.

Virtual machine and hypervisor security often intertwine in the sense that what is usually protected or safeguarded is the interface between the two. However, it can be said that the hypervisor deserves more attention because while a compromised virtual machine is still undesired, a compromised hypervisor usually entails the compromise of all the virtual machines operating under it. Thus, the hypervisor is a very tempting target for attackers. One attacker could subdue the services of all these VMs (VM Denial of Service), use the VMs to host its own servers with malicious process (e.g., phishing server), or even use then as zombie machines to perform Denial of Service attacks or other illicit activities.

**Virtual machine security**

Among the security benefits of using virtualization, two stand out: resource sharing and isolation [9]. Resource sharing simplifies resource allocation and management across different virtual machines under the same hardware rig.

As for isolation, a VM runs at a lower level of permissions than the hypervisor, as illustrated in Figure 2. This feature can at first be seen as a inhibitor of potential escape and privilege escalation attacks, but it should be noted that typically, there is no *air lock* between the hypervisor and the virtual machines [1]. The isolation provided in typical virtualized environments is usually only a software layer/interface alongside some hardware support [10] that may or may not have security in mind.
Virtual environment or virtualization detection

One feature malicious code can achieve is the detection of virtualization. While this may not always be a threat **per se**, it is reasonable, from the IT professional's point of view to treat it as sensible information. Understanding how the \texttt{isVirtualized()} information can be obtained (and then concealed or protected) can be a good starting point for threat modeling when dealing with virtualization security.

The successful detection of virtualization also has implications in the context of malware analysis assisted by OS malware sandboxes, as the detection of a virtual environment may lead to inhibition of further code execution and hinder the analysis of multi-path exploration systems that aim to scrutinize the behavior of a malicious program [11]. One approach the attacker may use is to search for and explore virtualization anomalies and discrepancies between virtual environments and actual computer systems. The premise is that in a secure virtualization using a passive defensive approach [12]:

\[\text{“[...] virtual hardware must be sufficiently similar to physical hardware to be indistinguishable to an adversary in the guest OS”}\]

As for an active defensive approach, such as the identification and the preemptive modification of VM detection guest code is also possible, although it requires some \textit{a priori} knowledge about malware code or patterns and a more robust hypervisor - with its own overhead. Thus, it can be said that almost all the discrepancies are associated with the level of compliance to the above premise in the design and implementation of virtualization software, as well as being akin to the effort in dealing with detection in an active and preemptive manner.

Logical discrepancies

Logical discrepancies are related to semantic differences in the interfaces of virtual and real hardware [12]. They are associated with the actual implementation of the ISA (Instruction Set Architecture), the execution and behavior of non-virtualizable instructions and also to off-chip specificities.

It has been shown that some non-virtualizable instructions such as SGDT, SIDT and SLDT can exhibit inconsistent and inaccurate behavior in a virtualization context [13], allowing user level inspection of virtualization software signature, such as VMWare [14]. The SGDT, SIDT and SLDT instructions handle, respectively, GDTR (Global Descriptor Table Register), the IDTR (Interrupt Descriptor Table Register), and LDTR (Local Descriptor Table Register) registers. They contain OS-related information but the
aforementioned instructions can be executed by applications without causing an exception.

This scenario was more critical in the past where single-core architecture computers were more dominant and the virtualization implementations had somewhat limited options to avoid conflicts between the host and the guest OS, having as a side effect inconsistencies and inaccuracies in the execution of these instructions. As with multicore processors where each processor has its own interrupt and memory data structures and also due to the coming and maturity of hardware assisted virtualization, this problem has been mitigated over the last hardware generations.

According to the latest Intel 64 and IA-32 Architectures SDM [15], a general-protection exception is generated if the User Mode Instruction Prevention (UIMP) is set and the current privilege level is not kernel mode. This affects SGDT, SIDT, SLDT, SMSW (Store Machine Status Word) and STR (Store Task Register) instructions. This feature was implemented starting with Intel’s Cannon Lake architecture, and helped to address the threat of kernel information leak and hypervisor detection artifices that rely on these instructions.

Some documented vulnerabilities associated with DoS - Denial of Service - attacks of virtual machines or virtual environments in the past were linked to semantic differences and logical discrepancies. For ones related to instruction emulation, some examples found were: CVE-2019-3840 (null pointer dereference), CVE-2018-6977 (3D rendering bug), CVE-2007-1322 (VM halt), CVE-2007-1366 (VM crash), CVE-2007-2455 (VM abort), CVE-2010-0435 (null pointer dereference and host OS crash), CVE-2013-2077 (unhandled exception and hypervisor crash), CVE-2013-2078013-2078 (hypervisor crash). Another example of DoS-related vulnerabilities associated with debugging are: CVE-2007-1876 (VM register context corruption) and CVE-2007-5906 (hypervisor crash).

Off-chip discrepancies

As hardware assisted virtualization has its focus mainly on CPU virtualization (that is, instruction set compatibility and digital logic implementation), off-chip discrepancies between physical and virtual hardware are plentiful. For example, in VMWare Workstation Pro 16.0, for the guest OS, the virtual processor presented is the same as the host OS, but the chipset and BIOS support are fixed: an Intel 440BX-based motherboard, a NS338 SIO chip set with and a Phoenix BIOS 4.0 Release 6 with VESA
In Oracle VirtualBox, the chipset provided can be the PIIX3 or the ICH9 and there is support for EFI alongside a legacy BIOS. These virtual setups can lead to absurd hardware configurations such as AMD processors running on intel-based motherboards.

Another source of discrepancy and anomalous hardware configurations are I/O peripherals and devices, often due to the associated cost of emulating and maintaining virtual models of a wide range of physical devices and the increased risk of vulnerabilities and bugs that such code could bring. For example, VirtualBox graphics device emulates a SVGA device (Windows) or a VMWare SVGA Graphics device (Linux) while previous VMWare products provided network, SCSI (Small Computer System Interface) and video that did not resemble any physical device.

In a similar manner, there were in the past some vulnerabilities associated with insecure SCSI virtualization: CVE-2011-3346 (buffer overflow) and CVE-2017-12190 (memory leak and system lockup). There were also vulnerabilities related to the virtualization of specific devices and subsystems which could lead to DoS attacks, such as CVE-2007-1337, CVE-2013-1210 (Virtual Ethernet Module) and CVE-2016-1465 (Application Virtual Switch). Also, CVE-2019-5516, CVE-2019-5517 and CVE-2020-5961 are all DoS vulnerabilities related to virtualization of graphics devices.

The design choice of implementing devices that do not resemble their physical counterparts can also theoretically enable VM detection by flagging and white-box testing, and even lead to other threats such as virtual machine escape, as some of these in the past were associated with vulnerabilities in device virtualization. For example, the aforementioned SVGA driver in some of VMWare products: CVE-2007-2454 (arbitrary code execution on the host OS), CVE-2009-1244 (arbitrary read and write and code execution on the host OS), CVE-2017-4902 (code execution on the host OS) and CVE-2017-4903 (code execution on the host OS).

In contrast, advancements have been made in the field to provide the same interfaces and semantics of I/O devices running on native environment and also to improve real-time response for latency-sensitive applications, as one of the main issues with I/O virtualization is overhead, which have considerable negative impact in performance of some applications due to I/O bottlenecks. While before device pass-through was the go-to solution, providing transparency, there was the potential for inconsistency and complications when dealing with multiple virtual machines and device
states, interrupt which and caching of shared (DMA - Direct Memory Access) memory [18].

This scenario motivated solutions such as direct I/O coupled with hardware IOMMU (Input-output memory management unit) and the development of hardware solutions [19] catching up to CPU hardware assisted virtualization, providing close-to-native performance, complete device semantics, and isolation. However, the main driving factors for these improvements are mostly performance, scalability, as well as compatibility. The idea that transparency is then ensured by these factors is erroneous, due to, as discussed thus far, technical and engineering limitations as well as practical demands [12].

Solutions such as IOMMU also are susceptible to DoS-attacks. For example: CVE-2020-27670 and CVE-2020-27671 (half-update in AMD IOMMU page-table entry), CVE-2010-0730 (guest OS crash), CVE-2010-2784 (guest OS crash, privilege escalation), CVE-2010-0306 (guest OS crash, privilege escalation), CVE-2014-2986 (NULL pointer dereference and host crash), and CVE-2014-8867 (host crash) are a few.

**Resource discrepancies**

As hypervisors share physical resources with its virtual machines, such as RAM, persistent memory, CPU cycles, etc., they can introduce attack surfaces related to this process. One example is the TLB (Translation Lookaside Buffer) mechanism which caches recent mapping of virtual pages to physical ones.

Another possibly problematic operation is context switch within the hypervisor and hosted VMs, namely VM Exit. This has been diminished with solutions such as Intel's VPID - Virtual Processor Identifier - and the EPT - Extended Page Tables - technology, making hardware take a bigger role in virtual memory management and cache [20]. Still there is some overhead left by TLB entry evictions.

In the past, research [11] showed concrete results with reasonable true positive rates for VM detection by using a method based off TLB eviction overhead in Intel's VT-x-based hypervisors. Also, the improper use or implementation of these solutions can introduce vulnerabilities. For example, CVE-2020-15567 (unauthorized writing of leaf page table entries due to improper restriction in shared page table mode on VT-d IOMMUs), CVE-2020-15567 (privilege escalation through a non-atomic modification of a live EPT PTE), CVE-2010-2938 (Xen and EPT support flaw related to host OS crash
Timing discrepancies

Although advancements in hardware assisted virtualization and paravirtualization made the performance gap between virtualized and native hardware shrunk, there are still undeniable differences between the two. Among the main factors is the evident virtualization overhead causing timing discrepancies, latency issues, or both.

Concerning overhead, it can be said that its intrinsic to virtualization. Depending on the type of implementation, device virtualization can be a source of timing anomalies not limited to the latency differences for a given operation but also due to variance between multiple operations introduced by the reliance on underlying data structures, such as CPU cache [12].

As for non-virtualizable instructions, they are often a substantial source of timing discrepancies. Both software (full virtualization) and hardware solutions (hardware assisted) showing different levels of overhead. An attacker can take advantage of these overheads and timing discrepancies to detect the presence of a hypervisor. To introduce the discussion on timing-based detection, a summary analysis of possible timing sources an attacker might leverage to perform virtual machine detection is:

- **TSC (Time Stamp Source):** One of the naivest forms of timing-based detection is to check the local time source, for example the TSC (Time Stamp Source) time. This is circumvented by using an offset and TSC scaling [15] to hide delays and latency discrepancies in hardware assisted virtualization;

- **HPET (High Precision Event Timer) and RTC (Real-time clock):** The HPET and other local timers can be manipulated by the hypervisor via interrupt interception [21]. As for the RTC it can be manipulated via I/O interception;

- **External clocks and time servers:** External time servers often offer poor resolution and accuracy but they can be used if the detection method parameters are scaled. Ferrie [22] argues that protocols such as NTP are documented and are easily interceptable by the hypervisor, but there is still the possibility that this type of traffic is encrypted and the hypervisor cannot deal - using a generic approach - with trusted-time-sources-based detection.[21].
Instruction timing and VM Exit overhead

The instruction timing and VM exit overhead methodology relies on the fact that the repeated execution of certain instructions and code may take more time within a virtualized environment than in a non-virtualized one [22. For example, a sample code containing a primality test or a storage I/O benchmark, or just simply a fixed number of instructions is executed many times in native and in virtualized environments and the time is measured. This approach by itself is not practical, since virtualized user code may run at or very close to native speed and to show discrepancies the number of instructions would need to be significantly large. Even then, there is too much variance (CPU types, optimization at CPU level, CPU cache, etc.) in order for one to obtain reasonable thresholds.

Another possible approach would be to measure non-virtualizable instructions execution time. This also suffers from similar problems as mentioned before. Another more refined way would be to use a baseline instruction and an instruction such as CPUID to base comparisons off a ratio between the two. This was one of the methodologies used in past research [11], which showed reasonable results, even though this method is still susceptible to TSC manipulation. However, the excessive use of instructions like CPUID, RDTSC and similar timing instructions might raise hypervisor suspicion. This motivated research to stealth methods that may avoid excessive reliance on the repeated execution of specific instructions. One example is another method presented by the referred authors, called "thread counting" [11]. This method uses threads and busy waiting to infer if a system is virtualized based on the number of completed CPUID instructions made in a given period of time, with the research showing noticeable results.

Rutkowska [21] proposed a substantial discussion on hypervisor detection based on timing discrepancies, in the context of VMBRs (Virtual Machine-based Rootkits) and from an attacker's perspective. Here, the intent is to hide the VMBR from detection, so it uses similar methods in which a hypervisor might avoid detection coming from code inside the virtual machine.

All internal timing methods are bound to fail if the hypervisor has a more fine-grained control of the data structures or resources used in the process and performs proactive, defensive measures, which is not always the case, as this approach might introduce considerable and undesired overhead.

Going back to the virtualization security premise, all comes down to the level of compliance to it, and it does not come without inconvenience. While manipulating
internal timing sources like TSC, RTC and HPET is usually not troublesome, external timing sources can be difficult for the hypervisor to deal with, as this type of detection often requires a deeper level of active defense and intervention. In order to succeed, an attacker could use heuristics to gather as much information as possible about the system (using multiple strategies) as it is running and then proceed to narrow possibilities using combinations of different detection methods.

**Transparency**

In the design of software hypervisor, efficiency and compatibility of the execution environment are usually the main concerns, while transparency is a secondary requirement [12]. Thus, the *sufficiently similar to physical hardware* premise is put to the test by attackers exploring architectural differences mainly in CPU virtualization and non-virtualizable instructions.

As for the paravirtualization, the guest OS is modified in usually non-transparent ways (e.g. modified software MMU in Xen [23] and Denali [24] Even if transparency is provided, it comes with the cost of providing one layer of the VMI (Virtual Machine Interface) for each deployment environment (e.g. native hardware, VMWare, Xen) [25] which may lead to a larger attack surface area.

Often VM tools leave an extensive footprint of running processes and services, registry keys modifications [26], and register values signatures [14] that can be used as information for VM-based malware decision making. An attacker has in its disposal a wide range of methods for detecting virtualization. By combining different types of detection, it can minimize the rate of false positives, (that can lead to malware scrutiny by sandbox software in analysis environments), even at the expense of false negatives, which is far more reasonable from the attacker's perspective. A fully transparent, detection-proof hypervisor is not feasible as the trade-off between security and convenience (performance, flexibility, scalability, etc.) is always there and the level of suspicion goes parallel to the level of inconvenience (overhead, constraints, unwieldiness, etc.).

**OS Virtualization security threats and vulnerabilities**

After the discussion on the security implications of the discrepancies between virtualized environments and their native counterparts, a brief study of some
vulnerabilities related to OS virtualization (except container virtualization) will be presented.

**Hyperjacking**

An attack which takes control over the hypervisor thus control over the virtualization environment and all the virtual machines running on it. The success of the attack depends on the attacker having physical access to a server or having a user or administrator install or run malicious code [27].

**VMBR - Virtual Machine-based Rootkits**

A VMBR is a type of malware which installs a hypervisor underneath an OS (host OS) and heaves it to a virtual machine [28] and the running OS is migrated into a VM [29].

This type of malware is stealthier than usual OS rootkits and measures such as virus scans and rootkit detection may not be very effective. A VMBR can provide the attacker sensible information such as keystrokes, networks packets, system events, memory allocation [30] and overall valuable user and system information. Furthermore, the attacker may as well use the control of hardware infrastructure to run malicious services alongside the compromised (now virtualized) OS, this is done by using an alternate OS hosted by the newly created hypervisor.

For the successful installation of a VMBR, the attacker needs access and certain privileges to modify the boot sequence of the target system [28], to assure that the VMBR loads before the OS. So the threat of this kind of malware is dependent of an exploitation of an already existing vulnerability, leveraging it for further malicious use. After the boot sequence is modified and the VMBR is installed, the target OS disk space is enclosed in a virtual disk and the VMBR can run malicious services [28].

**VMBR examples and proofs-of-concept**

Some examples of VMBRs are:

- *SubVirt*: An academic example and a proof-of-concept VMBR [28];
- *Vitriol*: proof-of-concept VMBR for MacOS X using intel VT-x [29]. It is installed by a kernel extension and unloaded by itself. The main functionality is comprised of [28]: 1. VT-x capabilities detection and initialization (*Vmx_init()*);
2. followed by a CPU fork ($Vmx\_fork()$) and finally 3. VM exit events $On\_vm\_exit()$;

- **BluePill**: A malware that exploits AMD64 SVM to move the OS into the VM, providing a lightweight hypervisor for the virtualized OS [21].

**Defense against VMBRs**

Lawson [31] discusses a way to detect a VMBR made up of VT-x exploitation and a software VMM by using functional and side-channel detection heuristics. The referred authors suggest a set of measures to hamper this type of attack, such as: the introduction of data-dependence; the compelling of attackers to emulate microarchitecture and behavior of specific CPU, chipsets or architecture-dependent features such as counters and performance indicators.

Nowadays, as with the increasing use and deployment of TPM chips, this attack is made substantially harder, as this type of attack has to deal with Trusted Computing protocols such as the setting, handling and modification of PCRs - Platform Configuration Registers which are aimed towards securing the boot chain.

**VM denial of service**

Denial of Service (DoS) are attacks where illegitimate users attempt to deny or impair resources to legitimate users [32]. Resources can be CPU usage, volatile or persistent storage, I/O devices, network resources and others. It has been shown [32] that the combination of virtualization overhead and performance deterioration in even light DoS attacks (i.e., 10Mb/s TCP SYN flood) can lead to massive performance decrease in virtualized systems compared to native counterparts. This is due to usual defenses deployed in native environments not being fully adequate in virtualization environments.

Although hypervisor implementations may force resource reservations and constraints to prevent a virtual machine from consuming resources from other virtual machines or from the host platform, the intended result is not always achieved. Some examples of vulnerabilities related to this problem are: CVE-2007-5498 (block back-end driver vulnerability) and CVE-2017-8158 (improper authorization for a specific file in the host).

The VM abstraction by itself can still be crippled by DoS attacks or vulnerabilities that target holes in the virtualization architecture or its implementation. For example, instruction emulation, as shown in previous sections. Also, there have been, in the past,
vulnerabilities associated with poor or improper isolation, for example: CVE-2010-0298 (improper privilege level enforcement for CPL3 code), 2010-0306 (improper CPU and I/O privilege levels instruction execution restriction) and CVE-2013-4356 (VM live migration improper constraints). These vulnerabilities can in other cases culminate in privilege escalation or even VM escapes.

**VM Escape**

The VM Escape threat can be defined as the breaking out of a virtual machine and the direct and unintended interaction with the hypervisor or the host OS. In others words, it is the escape of boundaries of the virtualized environment, which can lead to the compromise of the virtual machine, the hypervisor, the host OS or all of them.

The list of documented vulnerabilities associated with potential VM escapes is extensive, and for that only some examples will be given. First, flaws in the validation of parameters used in nested virtualization (CVE-2021-3653 and CVE-2021-3656) that could be used for achieving a VM Escape are documented. Also, there are records of exploits that leverage buffer overflow in device virtualization, namely, in CD-ROM device emulation allowing code execution on the hypervisor (CVE-2021-22045).

Likewise, as it had been mentioned before, the VGA (or SVGA) device can be an attack vector for buffer overflow exploits (CVE-2007-2454 in Parallels, CVE-2017-4902 and CVE-2017-4934 in VMWare); memory leak and arbitrary memory write in VMWare products (CVE-2009-1244); uninitialized stack memory also in VMWare products (CVE-2017-4903).

Other examples of buffer overflow vulnerabilities were related to a floppy disk device emulation CVE-2015-3456 (VENOM vulnerability) and to uninitialized memory exploit in a XHCI controller (CVE-2021-22040, CVE-2017-4904). Finally, other examples of VM Escape are: the exploit of vulnerabilities in page table code. Some examples were the improper validation of level 2 page entries (CVE-2015-7835), flaws in fat-paths utilized for updating the entries (CVE-2016-6258), flaws in L3 recursive page tables (CVE-2016-7092) and others.

**Pwn2Own 2017 hacking contest - complete VM Escape**

Earlier in 2017, a team of contestants at the Pwn2Own 2017 hacking contest chained a Microsoft Edge vulnerability, a bug in Windows 10 kernel and a vulnerability in VMWare to obtain a complete VM escape [33]. Another team also exploited a
Windows kernel bug and two VMWare bugs to obtain guest-to-host VM escape. Some of the aforementioned SVGA virtualization-related vulnerabilities were exploited in the event. After the exposition, the vulnerabilities were addressed in a patch by VMWare [34].

While it can be said that these exploits are not for the average attacker, it should be reminded that the teams started with an unprivileged account and minimum resources. In other words, a minimal attack surface. Events such as these can contribute to solidify security policies and defenses, given that vulnerabilities related to virtualization are often more valued than browser and OS ones, which can lead to further encourage the scrutiny by the security teams.

For overall defense and countermeasures against VM escapes, the basic still applies. Isolation should be enforced to ensure that applications running inside a VM do not have access to applications running inside other VMs or in the host OS, even in the case of break-in. For that it may be required more attention on implementation details and overlooks such as uninitialized memory areas and possible use-after-free scenarios. As the industry matures, this type of exploit is arguably becoming more difficult to achieve due to the design of more robust hypervisors and virtualization interfaces that achieve better isolation. Still, successful exploits are still emerging from time to time, especially in the BlackHat community [35], while there is the tendency to move to container virtualization in Cloud environments.

**Other virtualization vulnerabilities**

Besides attacks target specifically at hypervisor software, denial of service and VM escape attacks, virtualization software is also susceptible to the same kinds of vulnerabilities and threats that their native counterparts might face, such as privilege escalation, remote code execution, information leakage and others. These vulnerabilities can be diverse, ranging from simple, insecure, default password policy such as having an "!" (exclamation mark) as a root password for a VM (CVE-2008-510), to more complex ones such as flaws in functions in a block I/O layer in the Linux kernel that could possibly lead to information leak and system lockup (CVE-2017-12190). Some examples are CVE-2021-22045 (heap overflow allowing remote code execution) and CVE-2020-3992 (vulnerable implementation of openSLP in VMWare ESXi that could lead to remote code execution).
2 CONCLUSIONS

While in the past hardware assisted virtualization techniques in its inception introduced noticeable performance gains, it was still outperformed by software solutions in some configurations and workloads [36]. With the introduction of IOMMU, EPT, VPID (for Intel processors) as well as reduced latency of VM transitions [15] and context switch, the driving force of virtualization technology is still mainly directed at performance, usability, scalability and other industry demands, which can set security aside or be overlooked.

The indiscriminate use and deployment of hypervisors, virtual machines as well as many virtualization solutions in commercial, corporate as well as in government settings with little or insufficient attention to threat and risk analysis can have dangerous consequences, especially in a digital world with ever-increasing cases of ransomware attacks target at virtualization architecture [37].

When dealing with virtualization technology, security practices still apply. Risks should be reduced to manageable levels and is arguable that the security mindset of thinking about how things can be made to fail [38] can require a lot more cunning in such complex environments than in their native counterparts.
REFERENCES


