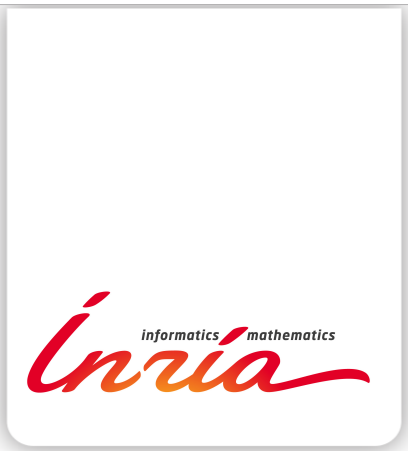
Hiding Virtual Machine Introspection Pauses in Networked Sandboxes with Network Simulation

Léo Cosseron, Martin Quinson, Louis Rilling, Matthieu Simonin

RESEARCH REPORT
N° 9528
November 2023
Project-Teams Myriads

ISSN 0249-6399 ISRN INRIA/RR--9528--FR+ENG



Hiding Virtual Machine Introspection Pauses in Networked Sandboxes with Network Simulation

Léo Cosseron*, Martin Quinson*, Louis Rilling†*, Matthieu Simonin*

Project-Teams Myriads

Research Report n° 9528 — November 2023 — 14 pages

Abstract: Virtual Machine Introspection (VMI) monitors have to constantly interrupt their target VMs, whether it is for intercepting specific instructions or inspecting memory in a consistent state. Previous works have shown that the additional Virtual Machine Monitor activity introduced by VMI impacts the guest performance and can be inferred by the target VM. We propose to manipulate the virtual clock of introspected VMs to hide the VMI overhead, improving the performance fidelity of the virtualized applications. Moreover, to extend this approach to closed network environments of analysis sandboxes, we show how virtual clock manipulation can be used to interconnect the VMs with a network simulator.

Key-words: virtualization, network simulation.

* Univ. Rennes, Inria, CNRS, IRISA, Rennes, France.
† DGA, Rennes, France.

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Dissimulation des pauses d'introspection de VM dans des bacs à sable utilisant un réseau simulé

Résumé : Les moniteurs d'introspection de machine virtuelle (VMI) doivent constamment interrompre les machines virtuelles qu'ils inspectent, que ce soit pour intercepter des instructions spécifiques ou pour pouvoir inspecter la mémoire dans un état consistant. Des travaux antérieurs ont montré que l'activité additionnelle du moniteur de VM due à VMI pouvait induire un impact sur les performances du système hébergé, et que cet impact pouvait être détecté par le système hébergé.

Nous proposons de manipuler l'horloge virtuelle des VMs inspectées afin de masquer les délais liés à VMI, de façon à améliorer la fidélité des performances ressenties par les applications virtualisées. De plus, nous montrons comment les manipulations d'horloges virtuelles peuvent être utilisées pour interconnecter les VMs à un simulateur réseau afin de pouvoir étudier des environnements réseaux dans le bac à sable d'analyse.

Mots-clés : virtualisation, simulation réseau.

Table of contents

1 Introduction

Virtual Machine Introspection (VMI) [6] can be used to detect potential malicious activity in Virtual Machines (VMs), for instance, by reading and writing the memory and registers used by the VM. It is also useful to analyze malware samples in sandbox VMs running in closed network environments. However *Hypervisor introspection* [22, 18] leverages various side-channels measurements (e.g. cache, timing) from the guest perspective to detect whether it is being inspected. Consequently a malware can take benefit from this to evade the introspection. A practical evasion strategy is demonstrated in [22] by exploiting the so-called *VM suspend side-channel* and uses local-to-the-VM timing measurements but also envisions network-based timing measurements (e.g observed latency between two endpoints on the network).

Following this direction, our contribution is twofold:

- First, we make the VMs' virtual clocks oblivious to VMI pauses. We reuse this technique to hide VM suspends caused by other debugging/analysis tools, such as `GDB`.

- Second, we synchronize the VMs with an accurate, discrete-event network simulator. Packets emitted by the VMs are received by their destination at the exact date computed by the simulator, with network-based timings unaffected by any potential VMI pause.

In the remaining parts of this paper we discuss in Section 2 the threat model and our contributions. Section 3 evaluates our approach on different scenarios. Section 4 reviews related work. Section 5 closes this paper and considers future work.

2 Contributions

Most VMI operations require to pause the introspected VM. These pauses can be detected by the VM, which has access to both local and remote time sources [7]. These clock sources must be manipulated by the Virtual Machine Monitor (VMM) to ensure timings stay coherent from the VM point of view.

We introduce our threat model in 2.1, then present how VMI pauses work in practice in 2.2 and our proposal to hide these pauses from the guest in 2.3. Finally we show how we extend this approach to network-based timings in 2.4.

2.1 Threat model

In our threat model, we assume the attacker has full access to the VM, including root and kernel rights. The attacker can check that the flow of time stays coherent, in order to detect any gaps or rollbacks that may be caused by a potentially introspecting VMM. The attacker has no prior knowledge that VMI in particular is going to be used on the VM.

Additionally, the VMM is considered to be secure and not controlled by the attacker. We disregard timing-based side-channels of VMs co-located on the same host.

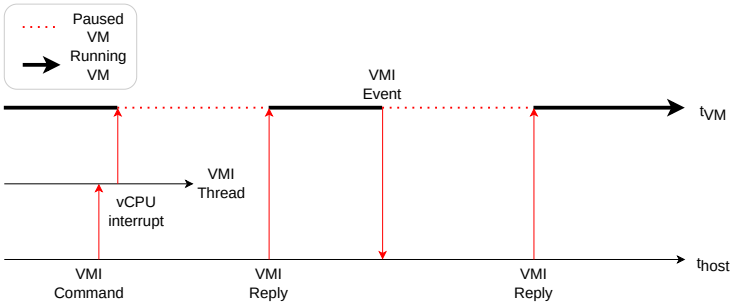


Figure 1: VM pauses caused by VMI.

The VM can communicate with other VMs on the network, and these VMs can be controlled by the attacker as well. However we assume the network environment is closed, with no access to uncontrolled networks or remote servers.

The attacker also knows the topology and the performance characteristics of the network (latency, bandwidth...).

2.2 VMI pauses

In this section we describe the general mechanism of a typical VMI monitor. Without loss of generality we focus on `libVMI` [15], a popular introspection library which supports many VMMs, like Xen [1] and KVM [12]. We focus on the KVM implementation, which is similar to the Xen implementation.

In the general case a VMI request is issued by a program called a VMI Monitor, typically using a dedicated VMI library, to the VMM where the target VM is running. `libVMI` initializes a thread in the VMM, that is dedicated to handle the commands issued by the VMI monitor. Some commands are handled by this VMI thread without pausing any vCPUs (e.g. check the availability of a given VMI feature). Other commands, which target the vCPUs, are delegated to the vCPU threads. The VMI thread thus triggers a VM-Exit (Transitions from guest to the VMM with Intel hardware virtualization) on each vCPU involved, and the rest of the work is done in the VMM by the vCPU threads. It is necessary to pause vCPUs for some VMI operations to ensure the VM state stays coherent. Finally when the VMI monitor has completed its work, it notifies the vCPU thread with a VMI reply message. The vCPU thread can then resume its execution.

`libVMI` also triggers VMI pauses on VMI events, that are fired when a specific condition is met during the vCPU execution (e.g. "about to execute a `MOV to CR3` instruction"). Users of `libVMI` can register custom callback functions on these events. In order to execute the VMI callback a VM-Exit is first triggered and after completion a VMI reply message is sent to resume the vCPU thread.

As a consequence a simple program that keeps reading the current timestamp is able to infer VMI activity by observing large gaps between two consecutive timestamps. Indeed the guest clock is progressing even when a vCPU thread is running in the VMM. Figure 1. illustrates both kinds of VMI pauses, and how they impact the guest clock.

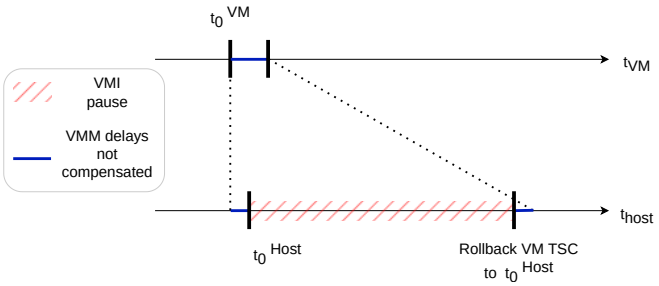


Figure 2: Time compensation of VMI pauses.

Note also that the same issue exists with other analysis tools that interrupt the VM, such as the `GDB` debugger.

2.3 Hiding VMI pauses

We propose to skew the virtual clock of an introspected VM to hide the time gaps caused by introspection pauses. This technique makes harder an exploit based on local-to-the-VM timing side-channel measurements.

In the following, we focus on Intel x86 CPUs and the KVM VMM. The main clock source is the TimeStamp Counter (TSC), a register incremented at a constant rate in modern processors. The guest TSC is a linear function of the host TSC, for which the VMM can configure the slope and the offset in the vCPU VMCS (Virtual Machine Control Structure, a data structure used by the processor to control each vCPU).

To hide VMI pauses, just before resuming the vCPU we measure the pause duration and adjust the guest TSC offset.

To only compensate the processing time of VM-Exit events due to VMI, a slight modification of the VMM introspection patch is required to, just before resuming the vCPU, (i) remember if the VM-Exit is a VMI pause (the VM-Exit reason returned by the processor is too generic) and (ii) in this case only, adjust the guest TSC offset. Remembering if the VM-Exit is a VMI pause is straightforward to implement as Libkvmi (A library for KVM introspection which is used by `libVMI`) is implemented as a kernel module and has access to KVM's internal structures.

We extended this approach to QEMU's `GDB` server. The QEMU process informs KVM to compensate the VM-Exit processing time using an extension of the `kvm_run` struct, which is shared between QEMU and KVM with an `ioctl`.

To correct the TSC, we need the timestamp at which the VM-Exit has been triggered. We use a timestamp recorded by the `rdtsc_ordered` kernel function, as soon as possible after the VM-Exit. Then, just before resuming the vCPU, if the VM-Exit is marked as a VMI pause, we rollback the vCPU guest TSC clock to the value recorded earlier. Since this timestamp is recorded slightly after the VM-Exit, the guest VM cannot perceive any clock rollback, as it is suspended during this time.

Figure 2 shows how time compensation works. We take a timestamp t_0^{Host}, as soon as possible after t_0^{VM}. When the VMI processing is over, we rollback the clock to t_0^{Host}. This leaves two uncompensated delays: the difference between

t_0^{Host} and t_0^{VM}, and the delay between the guest clock rollback and the `vmresume` instruction. As these two delays are very short (from a few microseconds to a few hundreds at most, as evaluated in 3.1), they are negligible compared to the duration of the VMI pause. We discuss this issue in Section 5.

On VMs with multiple vCPUs, adjusting the TSC requires additional precautions. Each vCPU has its own TSC register. The TSCs are synced during the boot of the system in order to be usable as a global clock. To avoid observable unsynchronized TSC clocks between vCPUs, the guest TSC of each vCPU must be adjusted by the same value and simultaneously. Moreover, to avoid observable rollbacks, this requires all vCPUs to be paused. Thus the clock correction approach can be used for VMI requests interrupting all vCPUs at once.

However correcting the clock for requests or events that only target some vCPUs remains a challenge. We cannot always interrupt all vCPUs after a request or an event, because this may break VMI scripts that expect that at least one vCPU is still running on the system, or that are waiting for events to trigger on multiple vCPUs to complete a previous event. Thus we currently only compensate pauses that target the whole VM, like `libVMI` function `vmi_pause_vm`, or a VM interrupt by `GDB`. This still covers VMI use cases that manually pause the VM to inspect it, but does not work for other scripts, especially when they rely on the event API.

Depending on the type of the VMM [8], the options to adjust the virtual clocks are different. On the first hand, VMM located in the kernel (Type 1 & sometimes 2, like KVM) have direct access to the underlying clock hardware, including the TSC, and can provide the best accuracy. On the other hand, VMM located in userspace (Type 2, like VirtualBox [21]) may not have full control of the clock hardware, which can lead to a worse precision.

We do not use any KVM specific features in our implementation, so it should work on other type 1 VMMs, and type 2 VMMs that offer options to adjust the virtual clock, albeit the precision may be degraded.

2.4 Network interactions

Section 2.3 focuses on hiding the VMI pauses from the guest perspective when the timing source is local to the VM. However this technique is not sufficient with network-based time sources. VMI pauses can indeed impact the Round Trip Time of a packet (e.g. ping) whether VMI pauses occur on the sender or receiver side. Furthermore with our time compensation technique, the clocks of the different VMs on the network would be out of sync, which can be easily noticed and introduce issues with some applications. To address this we show how, using compensation, we can interconnect introspected VMs with a discrete-event network simulator.

The next sections present how the simulator events can be used to synchronize the VMs clocks (in Section 2.4.1) and how the integration with VMI is performed (in Section 2.4.2).

2.4.1 Synchronization algorithm

To make the network timings of introspected VMs consistent, we propose to implement the whole network with a discrete-event network simulator and to

synchronize the VMs' clocks with the clock of the simulator. Discrete-event simulators run the simulation in an abstract time and make their clock progress forward by jumping to the next event time calculated.

Our synchronization algorithm works by dividing the time into slots of variable duration and bounded by *deadlines*. During each time slot the VMs run independently, we intercept the packets sent by the VMs, and append them in a queue with their timestamps. The VMs are paused when a deadline is reached. During a deadline pause, we drain the packet queue in chronological order, advance the simulator clock up to the send timestamp of each packet and inject the packet in the simulation. Then we advance the simulator clock to the deadline. If the simulator reports that a packet should be delivered at the deadline, we inject it in the destination VM.

The following two properties allow the accurate delivery of the packets:

1. each packet is delivered at the very beginning of a time slot,
2. no packet sent in a time slot should be theoretically delivered before it is injected in the simulator.

To achieve property 2, the duration of time slots is capped to Λ, the lowest network latency between any two VMs in the network simulation. To achieve property 1, at each deadline δ the next deadline is set to the minimum of $\delta + \Lambda$ and the earliest delivery date of packets currently in the simulator.

To ensure the synchronization of all the clocks, all the vCPUs of a VM are interrupted when it reaches a deadline. Moreover the time spent during the synchronization with the network simulator must not be taken into account in the guest clock, as detailed in 2.4.2.

2.4.2 Integration with VMI

To schedule a deadline, we use the VMX Preemption Timer, an Intel-specific timer that takes the form of a register, in which we can load a duration in TSC ticks. When the corresponding vCPU is running, this register counts down at a rate proportional to the host TSC, and triggers a VM-Exit when it reaches 0.

Intel mentions no delay or precision issues with the VMX Preemption Timer [10], although it is known to be broken on some processors [11]. The achievable precision to pause VMs is thus close to the CPU clock resolution.

Similarly to VMI pauses, deadlines pauses must be compensated by adjusting the VMs' virtual clocks. The process and implementation are the same as for the compensation of VMI pauses introduced in 2.3. The only difference is that we rollback the guest clocks to the deadline date instead of the approximate VM-Exit timestamp, in order to ensure all the VMs are synchronized at each deadline.

A special case to consider is when a VMX Preemption Timer VM-Exit is triggered late because another VM-Exit is being processed at the time of the scheduled deadline.

In this case, just before the end of the VM-Exit processing, the VMX Preemption Timer is loaded with the value 0, which causes an immediate VM-Exit, before executing a single instruction. Then, after the synchronization and the clock correction, we put the vCPU thread to sleep until the end date of the VM-Exit that finished late, before resuming execution. This guarantees that

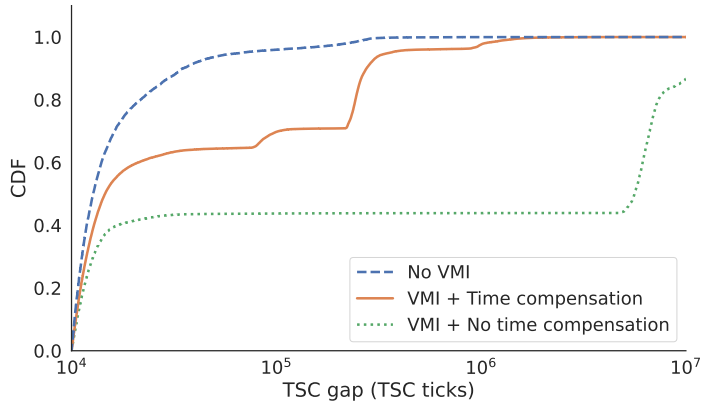


Figure 3: Cumulative distribution of observed TSC gaps which are greater than 10000 ticks.

the vCPU cannot see some events that usually cause VM-Exits (notably I/O events) last much less than reality.

3 Evaluation

In this section, we begin by evaluating the effectiveness of the time compensation mechanism to hide VMI pauses in 3.1. Then we focus on network interactions in 3.2.

These experiments were performed on a machine with an i7-1165G7 CPU and 32 GB of RAM. The i7-1165G7 CPU has a TSC frequency of 2.8032 GHz. Each VM is configured to use 8GB of RAM, 1 vCPU that is pinned to a physical CPU, and a Debian 11 guest OS.

We used the Linux/KVM 5.15 VMM, `libVMI` with the kvmi-v12 driver as the VMI library, and SimGrid 3.32 [2] as the network simulator.

To perform VMI on the VMs, we re-used the "mem-event-example" script of `libVMI`. This script performs memory access interception on the current RIP pointer. We chose this script because it triggers a large number of VMI pauses, and thus is representative of a scenario where VMI has a noticeable impact on the VM performance.

3.1 Local time sources

Our goal with this experiment is to evaluate our approach to compensate VMI pauses, while focusing only on local timing measurements. We aim to detect time gaps that may have been caused by the use of a VMI monitor.

We wrote a script that compares two consecutive TSC timestamps, obtained with the `rdtsc` instructions, separated by `lfence` instructions to avoid reordering. A large gap between two consecutive TSC measurements can be explained by an interrupt, a change in scheduling, or a VM-Exit occurring in between two measurements. When this difference exceeds a threshold of 10000 TSC ticks

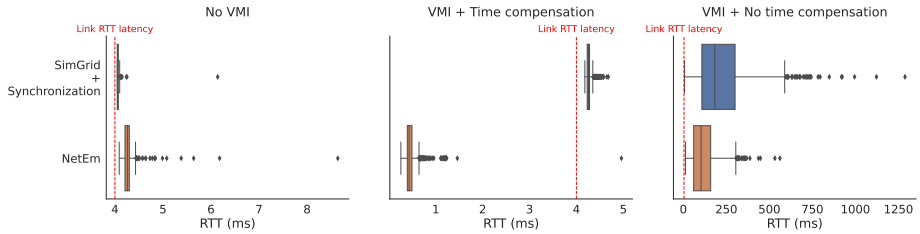


Figure 4: Distribution of ping RTT on different scenarios.

(that is 3.57µs), we log it. We chose this threshold to only keep gaps that may be caused by a VMI request, which is certain to last longer.

We executed this script on a single VM in three scenarios:

- No VMI script running;
- VMI script running, without time compensation;
- VMI script running, with time compensation.

The execution lasted 2 minutes in each scenario, getting between 12K and 38K samples for each scenario.

Figure 3 shows the empirical cumulative distribution function of the observed TSC gaps for each scenario, accounting only the gaps above the threshold.

In all three cases, we can see that a large part of the values are under 10^5 TSC ticks, and are likely gaps unrelated to VMI. With VMI and no time compensation, we can see a significant increase slightly before 10^7 TSC ticks gaps (About 3.6ms). These large gaps are caused by VMI pauses.

We observe the same increase with time compensation, but much earlier, in between 10^5 and 10^6 TSC ticks. These gaps likely result from the residual uncompensated delays of VMI pauses (see Section 2.3). We discuss potential improvements in Section 5. Nevertheless, there are observable TSC gaps of this order of magnitude even without VMI. It is thus much harder for an attacker to assert that there are long pauses caused by the VMM than in the case without time compensation, where he can observe gaps larger than 1ms.

Finally, although not shown here, we have not observed any TSC rollback with time compensation.

3.2 Network time sources

Now we focus on evaluating our approach in a closed network scenario. We propose an experiment that aims to detect the use of VMI by using network timings.

Our network topology is constituted of 2 VMs, connected with a 100Mbps link with 2ms of latency. The VMs are running on the same host, and connected as explained below.

We executed the “ping” command 1000 times from one of the guest to the other. By comparing the Round Trip Time (RTT) of packets to the expected result (Slightly over 4ms in average with our topology and usual packet processing delays), we can infer potential VMI activity.

We executed this script on two different scenarios:

- With the synchronization to a network simulator;
- Without synchronization, using `NetEm` [9] to perform the network emulation of the link. The VMs are connected with libvirt NAT forwarding.

For both setups, we performed the experiment three times: without VMI, and with VMI paired with or without time compensation. The VM issuing the pings is introspected during the whole duration of the experiment, while the other VM is not introspected at all.

First, without VMI, on the left of Figure 4, we observe both distributions are close to the theoretical link RTT latency.

Next we can see on the right of Figure 4 that, without any time compensation, the RTT of the pings varies greatly, from milliseconds up to a few hundreds of milliseconds. This happens in both scenarios. This is caused by the VMI script, which delays the processing of network packets, resulting in RTT much longer than the actual link RTT latency.

With time compensation enabled, we observe two different results as shown in the middle of Figure 4, whether we use our synchronization algorithm with a network simulator, or `NetEm`. With `NetEm`, the median RTT is about 0.5ms, much lower than the emulated link RTT latency. `NetEm` works by adding a FIFO queue in between the network protocols and the network card. The packets are held there and are forwarded to the network card only according to `NetEm` current configuration. With time compensation enabled, the packets stay in the FIFO queue during 2ms of host time. Then the packets travel on the network, the ping replies stay in the second `NetEm` queue and come back. While the RTT lasts slightly above 4ms of host time, compensated VMI pauses make it last much less in introspected VM time.

When using time compensation and the synchronization algorithm with the SimGrid network simulator, the simulator's and both VMs' clocks stay synchronized, resulting in consistent network timings, with in particular no value below the Link RTT latency, as shown on the plot.

4 Related work

Some works already studied the manipulation of a guest clock to mitigate timing side-channels, but focus mostly on micro-architectural side-channels. Indeed, the availability of a high-resolution timer like the TSC is a prerequisite for side-channels that measure cache timings for instance.

[19] authors propose to reduce the resolution of the TSC timer on Xen VMs to mitigate timing side-channels, while ensuring the system still behave normally. They can degrade the resolution of the TSC up to 2µs. [16] presents adjustments to the TSC and other time sources to mitigate timing side-channels, by adding randomized delays. However in both approaches the timer resolution is too high to prevent the detection of VMI pauses. Disabling the TSC entirely is not an option either, because this breaks multiple applications [16].

StopWatch [14] tackles the issue of timing side-channels attacks by co-resident VMs. The authors replace the real-time clock with a virtual clock that only counts the number of instructions executed in the guest. Although this can prevent VMI pauses from being detected by the VM, it provides unrealistic timings, because VM-Exit events unrelated to VMI (e.g. for I/O) take time

inherent to virtualization, and should count towards this virtual clock. Ether [5] is a malware analysis system that corrects time queries from a guest to subtract the overhead of the malware analysis framework. DRAKVUF [13] authors take this idea over in their design as well. As they manipulate the guest TSC they are close to our approach, however Ether does not tackle the issue of VMs with multiple vCPUs, and both papers acknowledge that it is not sufficient when network timings are involved.

[18, 22] focus on how a VM can detect a VMM performing introspection, and offer countermeasures. [18] authors propose to use time sources that are hard to tamper with, such as the HPET timer. Although in this paper we only address the TSC, other virtualized local clocks could be addressed similarly. [22] briefly mentions a virtual clock approach that is similar to ours, but does not address networking.

Finally many works have studied the pairing of a network simulator with VMs, but none in the context of VMI to the best of our knowledge. SliceTime [23] synchronizes Xen VMs with the ns-3 [17] network simulator to accurately evaluate the performance of distributed applications. They cut the time into slices, and perform the synchronization at each pause. A major limitation of SliceTime is that the VMs in the simulation have to run sequentially, whereas they can run in parallel with our approach. Finally our synchronization algorithm uses time slices of dynamic duration in order to improve accuracy, as proposed in [4], where they pair QEMU VMs with a SystemC emulator. However unlike our approach their synchronization algorithm does not take into consideration the network topology and does not address the case of hardware virtualization.

5 Conclusion and future work

In this paper we introduced time compensation to make the virtual clocks of VMs oblivious to VM suspends caused by VMI and debuggers. By synchronizing the VMs with a network simulator, we are able to mitigate the effect of VM suspends on network timings as well.

In our experiments, we have shown that large time gaps introduced by the use of a VMI monitor are largely compensated by our virtual clock, and that network timings are significantly more realistic as well.

To improve the precision of time compensation, we have to account for the delay before timestamping the VM-Exit and the delay between the clock correction and the `vmresume` instruction, as explained in 2.3. To fix this issue, we could use a dynamic adjustment algorithm similar to the one used for the local APIC timer emulation in KVM.

Compensating VMI pauses of a single vCPU in multi-vCPUs VMs remains an open challenge. We are considering strategies with different trade-offs between compensation and concurrent execution of the vCPUs.

In addition to pauses caused by VMI, I/O requests trigger VM-Exits that can last for a duration noticeably different than the emulated hardware I/O operation. We consider studying performance models for I/O devices, and adjusting the virtual clock of the VM similarly to the technique presented in this paper for the VMI pauses.

The predictions of network simulators are based on network models, which

are tuned to produce results that are close to reality. These models have to be validated beforehand, otherwise the results cannot be trusted. For instance, SimGrid has several models which have been validated for the TCP protocol [20], but not for UDP. ns-3 has more models that have been more thoroughly tested and can be a more reliable choice for the network simulator [3].

In this paper, we focused on closed network environments. As future work, we can introduce access to remote hosts on the Internet, while ensuring the timings on the local network are coherent. As we have full control over the local network thanks to the simulator, we can adjust the timings of outgoing and incoming packets to conserve consistent timings.

References

[1] Paul Barham et al. "Xen and the art of virtualization". In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003*. Ed. by Michael L. Scott and Larry L. Peterson. Bolton Landing, NY, USA, October 19-22, 2003: ACM, 2003, pp. 164–177. DOI: 10.1145/945445.945462. URL: https://doi.org/10.1145/945445.945462.

[2] Henri Casanova et al. "Versatile, scalable, and accurate simulation of distributed applications and platforms". In: *J. Parallel Distributed Comput.* 74.10 (2014), pp. 2899–2917. DOI: 10.1016/j.jpdc.2014.06.008. URL: https://doi.org/10.1016/j.jpdc.2014.06.008.

[3] Maurizio Casoni et al. "Implementation and Validation of TCP Options and Congestion Control Algorithms for Ns-3". In: *Proceedings of the 2015 Workshop on Ns-3*. WNS3 '15. Barcelona, Spain: Association for Computing Machinery, 2015, pp. 112–119. ISBN: 9781450333757. DOI: 10.1145/2756509.2756518. URL: https://doi.org/10.1145/2756509.2756518.

[4] Massimiliano d'Angelo et al. "A Simulator based on QEMU and SystemC for Robustness Testing of a Networked Linux-based Fire Detection and Alarm System". In: *Embedded Real Time Software and Systems (ERTS2012)*. Toulouse, France, Feb. 2012. URL: https://hal.science/hal-02192275.

[5] Artem Dinaburg et al. "Ether: malware analysis via hardware virtualization extensions". In: *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008*. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. Alexandria, Virginia, USA: ACM, 2008, pp. 51–62. DOI: 10.1145/1455770.1455779. URL: https://doi.org/10.1145/1455770.1455779.

[6] Tal Garfinkel and Mendel Rosenblum. "A Virtual Machine Introspection Based Architecture for Intrusion Detection". In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003*. San Diego, California, USA: The Internet Society, 2003. URL: https://www.ndss-symposium.org/ndss2003/virtual-machine-introspection-based-architecture-intrusion-detection/.

[7] Tal Garfinkel et al. "Compatibility Is Not Transparency: VMM Detection Myths and Realities". In: *Proceedings of HotOS'07: 11th Workshop on Hot Topics in Operating Systems*. Ed. by Galen C. Hunt. San Diego, California, USA: USENIX Association, 2007. URL: http://www.usenix.org/events/hotos07/tech/full%5C_papers/garfinkel/garfinkel.pdf.

[8] Robert P Goldberg et al. "Architectural principles for virtual computer systems". PhD thesis. Harvard University Cambridge, MA, 1973.

[9] Stephen Hemminger. "Network emulation with NetEm". In: *Linux conf au*. 2005.

[10] Intel. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3 (3A, 3B, 3C & 3D): System Programming Guide*. Dec. 2021. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[11] *Intel Xeon Processor 5500 Series Specification Update (Errata AAK139)*. 2015. URL: https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-5500-specification-update.pdf.

[12] Avi Kivity et al. "KVM: the Linux virtual machine monitor". In: *Proceedings of the Linux symposium*. Vol. 1. 8. Ottawa, Ontorio, Canada. 2007, pp. 225–230.

[13] Tamas K. Lengyel et al. "Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system". In: *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014*. Ed. by Charles N. Payne Jr. et al. New Orleans, LA, USA: ACM, 2014, pp. 386–395. DOI: 10.1145/2664243.2664252. URL: https://doi.org/10.1145/2664243.2664252.

[14] Peng Li, Debin Gao, and Michael K. Reiter. "Mitigating access-driven timing channels in clouds using StopWatch". In: *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Budapest, Hungary: IEEE Computer Society, 2013, pp. 1–12. DOI: 10.1109/DSN.2013.6575299. URL: https://doi.org/10.1109/DSN.2013.6575299.

[15] *LibVMI: Simplified Virtual Machine Introspection*. 2010. URL: https://github.com/libvmi/libvmi.

[16] Robert Martin, John Demme, and Simha Sethumadhavan. "TimeWarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks". In: *39th International Symposium on Computer Architecture (ISCA 2012)*. Portland, OR, USA: IEEE Computer Society, 2012, pp. 118–129. DOI: 10.1109/ISCA.2012.6237011. URL: https://doi.org/10.1109/ISCA.2012.6237011.

[17] George F. Riley and Thomas R. Henderson. "The *ns-3* Network Simulator". In: *Modeling and Tools for Network Simulation*. Ed. by Klaus Wehrle, Mesut Günes, and James Gross. Springer, 2010, pp. 15–34. DOI: 10.1007/978-3-642-12331-3_2. URL: https://doi.org/10.1007/978-3-642-12331-3%5C_2.

[18] Tomasz Tuzel et al. "Who watches the watcher? Detecting hypervisor introspection from unprivileged guests". In: *Digit. Investig.* 26 Supplement (2018), S98–S106. DOI: 10.1016/j.diin.2018.04.015. URL: https://doi.org/10.1016/j.diin.2018.04.015.

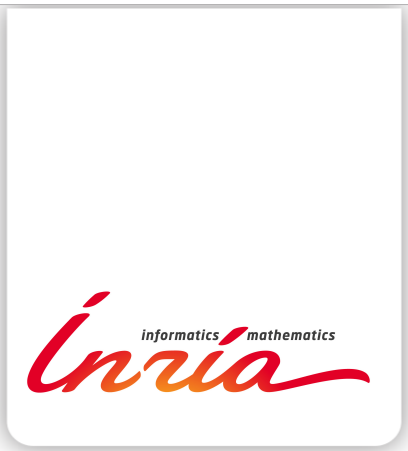
[19] Bhanu Chandra Vattikonda, Sambit Das, and Hovav Shacham. "Eliminating fine grained timers in Xen". In: *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011*. Ed. by Christian Cachin and Thomas Ristenpart. Chicago, IL, USA: ACM, 2011, pp. 41–46. DOI: 10.1145/2046660.2046671. URL: https://doi.org/10.1145/2046660.2046671.

[20] Pedro Velho and Arnaud Legrand. "Accuracy Study and Improvement of Network Simulation in the SimGrid Framework". In: *SIMUTools'09, 2nd International Conference on Simulation Tools and Techniques*. Rome, Italy, Mar. 2009. URL: https://inria.hal.science/inria-00361031.

[21] *VirtualBox*. 2007. URL: https://www.virtualbox.org/.

[22] Gary Wang et al. "Hypervisor Introspection: A Technique for Evading Passive Virtual Machine Monitoring". In: *9th USENIX Workshop on Offensive Technologies, WOOT '15*. Ed. by Aurélien Francillon and Thomas Ptacek. Washington, DC, USA: USENIX Association, 2015. URL: https://www.usenix.org/conference/woot15/workshop-program/presentation/wang.

[23] Elias Weingärtner et al. "SliceTime: A Platform for Scalable and Accurate Network Emulation". In: *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011*. Ed. by David G. Andersen and Sylvia Ratnasamy. Boston, MA, USA: USENIX Association, 2011. URL: https://www.usenix.org/conference/nsdi11/slicetime-platform-scalable-and-accurate-network-emulation.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399