How to Resuscitate a Sick VM in the Cloud

Xuhua Ding
Singapore Management University

***Abstract*—A guest virtual machine in a cloud platform may fall "sick" when its kernel encounters a fatal low-level bug or is subverted by an adversary. The VM owner is hence likely to lose her control over it due to a kernel hang or being denied of remote accesses. While the VM can be rebooted with the assistance from the cloud server, the owner not only faces service disruption but also is left with no opportunity to make an in-depth diagnosis and forensics on the spot, not to mention a live rectification. Currently, the cloud service provider has neither incentive nor the technology to assist owners to resuscitate their falling VMs. In this paper, we propose a new cloud service termed *VMCare-As-A-Service* (VaaS) with the vision that the owner of a sick VM applies her tools running on a special VM to repair it. VaaS demands innovative cloud technologies for the unique infrastructure support as well as new software security techniques for attacks neutralization and runtime rectification upon a running and corrupted kernel. We examine the ensuing research challenges and present several preliminary approaches to kindle the interests from the community.**

I. Introduction

According to a recent report from Gartner[1], the worldwide market for public Infrastructure-As-A-Service (IaaS) Cloud will reach more than $150 billion in 2023. The surge of IaaS subscription is unsurprisingly companied with the growth of attacks against guest VMs. A sophisticated attack can subvert and control the victim's kernel. As a result, neither the guest owner's programs in the VM nor commands sent to the VM can be properly executed as expected. Even without attacks, the guest kernel may hang or crash due to fatal bugs triggered during its execution. In both cases, the guest VM with kernel failure or compromise becomes *undependable*.

A straightforward and seemingly effective solution to an undependable VM is just to restart it. Unfortunately, this approach is not desirable in many scenarios because it does not eradicate the root cause of VM failure. The attacker may leave malicious code persistently in files so that the subsequent launch, even with a new kernel image, is still susceptible to kernel compromise. If the failure is due to kernel bugs, there is no guarantee whether the bug will not be triggered any longer. For tenants using the VM to host servers, the termination-restart method incurs service disruption. Furthermore, the reset obliterates the important runtime data, which hinders kernel bug troubleshooting and live forensics for attack traces. Thus, an ideal solution is not to kill the the undependable virtual machine but to resuscitate it with dependability and trustworthiness.

In the rest of the paper, we first explain the hurdles encountered by VM owners and CSPs when coping with an undependable VM. We then propose a new type of cloud service and show that existing cloud security techniques are insufficient to enable it. Next, we present the research challenges from three angles. Lastly, we explore different parameters when designing solutions to overcome those challenges.

II. A New Cloud Service

To explain the demand for a new cloud service, we shed light on the difficulty of handling an undependable VM through the lens of the owner and the CSP, respectively, and remark that neither of them can resolve the problem alone.

A. A Tale of Two Predicaments

VM Owner. When an on-premise computer becomes undependable, its owner may use special hardware devices or Intel Management Engine (IME) to gain a foothold in it and carry out the necessary tasks. In contrast, for an undependable VM hosted in the cloud, its owner does not have that foothold. The sole way for the owner to access the VM is via the network channel which often becomes unreliable/unavailable when the VM kernel fails. Hence, it is infeasible for the owner to revive the VM without the CSP's assistance.

Cloud Service Provider. To rectify an undependable VM is obviously beyond the conventional service scope of IaaS. With the privilege of managing all hardware resources in the cloud, the Virtual Machine Monitor (VMM) can directly read/write the VM's physical memory and disk files. Nonetheless, its capability is crippled by the absence the semantics of the kernel and applications running inside the VM. As noted by Jain et al. [1], how to bridge the semantic gap is the key challenge in out-of-VM[2] introspection schemes [4], [5], [6], [7]. Even if the semantic gap challenge was overcome, existing VMI techniques only collect data and help to diagnose the VM. They are incapable of neutralizing attacks or amending the VM runtime. Besides the technical challenge, any CSP based rectification, if technically feasible, unavoidably intrudes the owner's VM which may result in legal and privacy related implications.

B. VMCare-As-A-Service

Since the existing cloud services cannot resolve the issue, we propose a new cloud service termed **VMCare-as-a-Service** or **VaaS**. Under this computing paradigm, the CSP provisions the hardware infrastructure for an authorized agent (e.g., the VM owner or a third party hired by the owner) to run its chosen software against the undependable VM. In specific,

[1] *Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023*, https://www.gartner.com/en/newsroom/, Oct 31 2022

[2] Those in-VM introspection schemes [2], [3] are infeasible as their security relies on the guest kernel.

the CSP sets up a special virtual machine (denoted as the **Responder VM**) and the agent runs the diagnose and repairing tools of her own choice inside the Responder VM. The key feature of the Responder VM is that *the agent tools therein are empowered by the VMM to introspect and modify any virtual memory in the target VM as if they run inside the target's kernel*. Figure 1 below illustrates the vision of VaaS.

The VaaS model resolves the aforementioned predicaments of the VM owner and the CSP. On the one hand, the owner is accessible to a wide range of toolkits to rectify her undependable VM, which could turn out to be more flexible and versatile than handling an on-premise computer using a hardware tool. On the other hand, the CSP's service remains agnostic and non-intrusive to its tenants VMs as it still centers around resource management and access control.

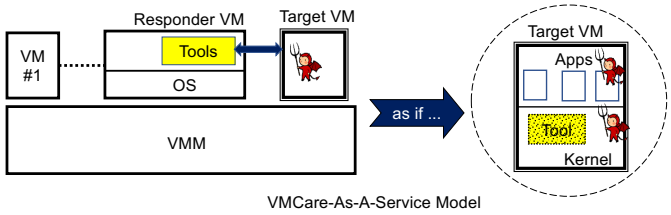


Fig. 1. Illustration of VMCare-As-A-Service. The target VM is undependable due to attacks subverting the kernel.

Analogously, the Responder VM is the "operation theatre" that provides the needed facility and equipments to save a sick VM whereas the agent is the "surgeon" who undertakes a surgical operations by using his VMCare tools deployed inside the facility.

III. Research Challenges

Enabling and supporting VaaS requires innovations of system and software technologies for the CSP and for the VM owners to overcome the following challenges.

A. The Architecture of Responder VM

The primary goal of the Responder VM is to provide the desired system support to the VMCare tools, so that tool developers only focus on their software functionalities. Different from a normal program, the tool is expected to make native virtual memory accesses to processes of the target VM. Hence, the Responder VM needs to satisfy the following requirements.

- Its page tables should not only define the tool's virtual memory, but also embody the same address mappings as those used in the target VM. This requirement is to allow for native target memory accesses.
- It should resist attacks launched by the adversary in the target VM. In other words, the tool's security should not be undermined because of execution in the Responder VM (except its own software vulnerability exploited by the target's data).
- It should not give either the (malicious) tool or the adversary in the target any higher advantage to jeopardize the VMM or other VMs over attacks from a regular guest VM. This requirement is to rule out possible undesirable security side-effects of VaaS.

B. Capability of VMCare Tools

Although there are techniques proposed for forensics, guest kernel object extraction (e.g., [8], [9]) and kernel live patch or update [10], [11], [12], their capabilities are not sufficient for VaaS due to the environment mismatch. We envisage a suite of tools with various capabilities running in the Responder VM.

- **Attack Termination** We need tools to recognize and terminate ongoing attacks. While traditional anti-virus or intrusion detection systems are able to do so *within* the affected system, the tools for the VaaS are expected to achieve the same from the outside.
- **State Cleansing** We need tools capable of correctly identifying and gracefully cleansing the corrupted parts of the target kernel. It is more challenging to design such tools than for forensics and VMI. Besides the demand for fine-grained knowledge of code and data semantics, no existing techniques is universal enough to remove an arbitrary chunk of instructions and/or data during kernel execution without breaking its consistent state.
- **Repairing** We need tools capable of live updating or patching the kernel. For the target VM to resume normalcy, it is necessary to fix the vulnerability. Most existing kernel patching techniques [10] are only applicable to kernels which are not compromised yet. While Kshot [12] addresses this issue using SGX, it does not match the VaaS setting in the cloud.

C. Privacy Protection

The VaaS service may tigger entangled privacy concerns among the CSP, the VM owner, and the entity providing and/or operating VMCare tools. From the VM owner's perspective, the service exacerbate her privacy concern against the CSP. Although a rogue CSP can always peek at the VM memory, it faces the difficulty of locating critical data and extracting its semantics. Nonetheless, the problem is made easier to overcome when the CSP observes the operations made the VMCare tools. Hence, one privacy challenge is how to prevent the CSP from gaining advantages of invading tenant privacy from the VaaS service. Moreover, the VMCare tools may enclose proprietary techniques from their vendors. A rogue CSP may attempt to infringe on their copyrights by copying them, observing their activities or even hacking them. Hence, the second privacy challenge is how to protect the VMCare tools' copyright against various attacks from the CSP.

In short, the VaaS service gives rise to challenges in system architecture design, automated software repairing and privacy protection. Although similar ones have been studied in other system and application settings and some may even have mature solutions, the VaaS introduces these research problems with unique demands and constraints.

IV. Design Parameters

In the following, we examine a few preliminary approaches that address the aforementioned challenges from different angles. These approaches apply different design parameters with respective pros and cons. We do not claim that they satisfactorily answer the aforementioned research problems. Instead, they serves as teasers that are expected to inspire deeper studies leading to innovative solutions.

A. Live VM vs. Frozen VM

A critical design parameter is whether to keep the VM alive or pause it when the VMCare tool in the Responder VM conducts its diagnosis and repairing work. The former means that all or most of the threads in the target VM continue their executions whereas the latter means that no thread in the target VM is active and all CPU cores are trapped to the VMM.

Keeping the VM full or partially alive can be an application demand from the VM owner. For instance, the VM may host a heavy computation workload (e.g., machine learning model training). It is against the owner's interests to terminate or even temporarily pause the execution. From the security perspective, the benefit can be a revealing of adversarial activities to a fuller extent, which contributes to more effective attack analysis and evidence collection. Clearly, it is difficult to fulfill the requirement because the incurred transparency, race-condition, data consistency issues, to name a few.

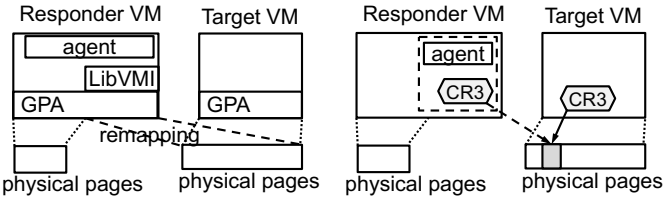
It is much easier to deal with a paused VM. The tool in this approach works with *static* memory data and CPU context of the target. There is no concerns regarding transparency or race conditions. Nonetheless, we emphasize that it still required to preserve the target VM's ability to resume execution after the repairing work. In an ideal scenario, the VaaS empowers the tools to flexibly control the target VM in the pause-resume cycles.

B. Remapping vs. Reusing

There are two approaches to realizing the paging hierarchy for the Responder VM. Note that the objective is to provide the mappings for the VMCare tools to access the target kernel's virtual memory.

One approach is to leverage LibVMI [13], a software-based address translation, as used in out-of-VM introspection. The VMM allocates for the Responder VM an additional GPA region with the same size as the target VM's physical memory. It then creates the Extended Page Tables (EPTs) to map the GPA region to the target VM physical memory. When the tool needs to reference of a virtual address of a target process, LibVMI parses the target's page tables, locates the physical page, and remaps it to the tool's virtual address space. The main benefit of this method is its universal applicability for all architectures. Its drawback is its lower speed than native memory access. Hence, it suits scenarios involving a frozen VM since the captured target memory is not updated during analysis. Figure 2(a) illustrates the remapping approach.

The other approach is to reuse the `CR3` as proposed in OASIS [14]. OASIS builds a special execution environment



(a) Physical Page Remapping (b) Paging Table Reusing

Fig. 2. Illustration of two difference methods of providing target mappings to the tool in the Responder VM.

inside which a user space program can executes within the target kernel's virtual memory. The VMCare tool can directly reference a target VA with the MMU traversing the paging tables. In view of its native speed support and consistent mapping assurance, this approach is suitable for live VM repairing. However, the technique of `CR3` reusing is only applicable for x86-64 guest VMs and the environment requires an update when the tool changes its target from one thread to another. More importantly, OASIS launches the tool as a regular application in the host OS (i.e., the VMM), instead of a guest VM. This unfortunately conflicts with the rationale behind the VaaS service because the VMM is directly involved and exposed to the tool. Figure 2(b) illustrates the paging hierarchy reusing approach.

A combination of the two approaches seems more appealing. The former can be applied for read-only memory in the target while the latter is applied for a specific thread for the tool to make fine-grained and more calibrated operations.

C. External Execution vs. Injected Execution

Another dimension of design is whether a VMCare tool or a fraction of its code is injected to the target kernel and executes therein. Running the tool strictly inside the Responder VM is surely a safe approach as threads in the target VM does not have the paging support to access the Responder VM memory. Nonetheless, its capability is constricted by the imposed system setting. For instance, the tool can not *change* the CPU contexts (including MSRs) of the target's vCPU cores which may affect the I/O handling and low-level system states. Another issue is related to caches especially translation caches as they could be different from the corresponding page table entries. Keeping the tool outside of the target VM cannot satisfactorily deal with these issues.

An injected execution resembles the kernel's exception handling which heals itself to some extent. It can in the form of an interrupt handler or a hook on a kernel function. In the extreme case, it can be a self-contained kernel thread in the target VM. The obvious benefit is its effectiveness since it is equipped with sufficient system and software semantics and directly operates on the target. Equally oblivious is its limitation, i.e., the insecurity of tool execution inside the ailing target VM contaminated by malware. That said, it is not entirely infeasible to secure the tool's execution by exploring virtualization-based isolation techniques [15], [16],

[17]. Nonetheless, these isolation techniques require the VMM involvement at runtime, which does not appear appealing to the CSP. Hence, the feasibility of an injected execution largely hinges on the feasibility of an isolation mechanism controlled by one VM upon another VM.

D. Hardened Application vs. Hardened VM

Existing hardware based TEE technologies include Intel Software Guard Extension (SGX), AMD Secure Encrypted Virtualization (SEV) and ARM TrustZone, with Intel Trust Domain Extensions (TDX) [18] and ARM Confidential Compute Architecture (CCA) emerging on the horizon. The popularity of TEE are largely attributed to the growing privacy concern in cloud services. It is thus compelling to explore the TEEs to tackle privacy challenges in VaaS.

a) Enclave: SGX enclaves isolate a virtual address space segment of an application against all system software accesses, including the VMM. A VMCare tool vendor can shield its proprietary code and data using an enclave while use its non-proprietary code outside the enclave to read and/or write the target's memory pages. Since instructions inside an enclave are in user-mode only, they cannot directly reference code or data pages under the target kernel's mappings. Thus, it cannot execute on top of OASIS in harmony. While it may overcome the privilege barrier by proactively inducing context switches with software interrupts, the incurred overhead especially due to exiting from the clave and re-entering it is prohibitively high. Nonetheless, the enclave approach works well with the page remapping approach, since target kernel pages can be remapped to the non-supervisor pages. Note that enclave code accesses VA regions outside of the enclave in the same as regular memory operations.

b) Secure VM: Since enclaves are application centric, it is inconvenient for a VM responding services who may use legacy tools that can not be hardened using enclaves. The hardware based secure VM technology such as Intel TDX and AMD SEV offers a more deployable alternative. Since the Responder VM is protected as a whole by the hardware, the respondent can upload any tool at its disposal without making efforts to harden it individually. In Intel TDX, a Trust Domain (TD) accesses to its *private* memory protected under Intel's multi-key, total-memory-encryption (MKTME) technology as well as its *shared* memory in plaintext. With LibVMI, the target VM memory can be mapped to the shared memory of a TD holding the Responder VM for the tools to read and write. Nonetheless, it is unclear whether the `CR3` reusing technique in OASIS is compatible with TDX.

E. Direct Access vs. Oblivious Access

The hardware TEE techniques, be it for an application or for an entire VM, only thwart direct accessing from the rogue VMM. The adversary can find out which target page is modified and use cache side channels to determine which page is read. With the patten, the adversary can possibly infer the algorithm or objectives of the VMCare tool. As mentioned earlier, the attack also helps the adversary to have an easier semantic extraction from the target memory. It is hence desirable to stall information leakage from the pattens when accessing the target memory.

Oblivious RAM (ORAM) [19] and its derivatives such as Path ORAM [20] are the well-known cryptographic techniques dealing with access patten privacy. Those algorithms can potentially be applied in tandem with a TEE-hardened Responder VM or VMCare tool. Only the accesses to the target VM memory follow the ORAM-style algorithm to generate a randomized patten.

Despite of recent improvement on efficiency, ORAM algorithms still take a heavy performance toll due to its inherent working mechanism (i.e., to pad an intended access with random-looking accesses to make it appear in a uniform distribution). To reduce the overhead, the VMCare tool may apply ORAM for a pool of pages (e.g., kernel objects) instead of the entire target VM. We also note that the algorithms incur frequent write-access to the target memory. Hence, it is more suitable for scenarios using frozen VMs.

V. Summary

To summarize, we envision a new cloud service named as VMCare-as-a-Service (VaaS) to cope with situations where a guest virtual machine hosted in a cloud environment encounters fatal system failures due to attacks or kernel crashes. The spirit of VaaS is for the CSP to provision the infrastructure in the form of a Responder VM and for a third-party VMCare service provider or the VM owner to run their own tools to attend to the concerned virtual machine. This role-splitting approach suits the interests of all stakeholders.

We identify the challenges related to the VaaS service from the system, software and privacy perspectives, and discuss an array of design parameters with preliminary analysis upon the pros and cons. In general, it is a research area with abundant exciting problems. While some of them are related to known problems, they do present fresh demands due to the unique system setting in VaaS.

We also note that the VaaS model can be generalized for other applications. For instance, the technology can be used by law enforcement to carry out forensics against VM-based vice and crime. It can also be applied to personal computers, phones, and servers since modern commodity operating systems such as Windows 11, Linux and Android 13 have built-in VMM functionality. The VaaS approach could be more flexible and versatile than existing hardware based tool chains.

Acknowledgement. This research / project is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme, National Satellite of Excellence in Mobile Systems Security and Cloud Security (NRF2018NCR-NSOE004-0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Cyber Security Agency of Singapore.

REFERENCES

[1] B. Jain, M. B. Baig, D. Zhang, D. E. Porter, and R. Sion, "SoK: Introspections on trust and the semantic gap," in *Proceedings of the 35th IEEE Symposium on Security and Privacy*, 2014.

[2] M. I. Sharif, W. Lee, W. Cui, and A. Lanzi, "Secure in-VM monitoring using hardware virtualization," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 477–487.

[3] Z. Gu, Z. Deng, D. Xu, and X. Jiang, "Process implanting: A new active introspection framework for virtualization," in *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*. IEEE, 2011, pp. 147–156.

[4] D. Srinivasan, Z. Wang, X. Jiang, and D. Xu, "Process out-grafting: an efficient out-of-VM approach for fine-grained process execution monitoring," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 363–374.

[5] M. Carbone, M. Conover, B. Montague, and W. Lee, "Secure and robust monitoring of virtual machines through guest-assisted introspection," in *Research in Attacks, Intrusions, and Defenses*. Springer, 2012, pp. 22–41.

[6] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," in *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011, pp. 297–312.

[7] Y. Fu and Z. Lin, "Space traveling across VM: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 586–600.

[8] T. Garfinkel, M. Rosenblum *et al.*, "A virtual machine introspection based architecture for intrusion detection." in *Proceedings of NDSS*, vol. 3, 2003, pp. 191–206.

[9] S. Suneja, C. Isci, E. de Lara, and V. Bala, "Exploring vm introspection: Techniques and trade-offs," in *Proceedings of the 11th ACM International Conference on Virtual Execution Environment (VEE'15)*, 2015.

[10] S. Kashyap, C. Min, B. Lee, T. Kim, and P. Emelyanov, "Instant OS update via userspace checkpoint-and-restart," in *Proceedings of 2016 USENIX Annual Technical Conference (ATC)*, 2016.

[11] S. Farhang, M. M. Kamani, J. Grossklags, and P. Liu, "Take it or leave it: A survey study on operating system upgrade practices," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018.

[12] L. Zhou, F. Zhang, J. Liao, Z. Ning, J. Xiao, K. Leach, W. Weimer, and G. Wang, "KShot: Live kernel patching with smm and sgx," in *Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.

[13] B. D. Payne, "Simplifying virtual machine introspection using LibVMI," Sandia National Laboratories, Tech. Rep. SAND2012-7818, 2012.

[14] J. Hong and X. Ding, "A novel dynamic analysis infrastructure to instrument untrusted execution flow across user-kernel spaces," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2021.

[15] O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel, "Inktag: secure applications on an untrusted operating system," in *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS '13, 2013.

[16] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "Trustvisor: Efficient tcb reduction and attestation," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, 2010.

[17] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, "Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems," in *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS XIII, 2008.

[18] Intel, "Architecture specification: Intel Trust Domain Extensions (Intel TDX) module," June 2022.

[19] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM*, vol. 43, no. 3, 1996.

[20] E. Stefanov, M. V. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path ORAM: An extremely simple oblivious RAM protocol," *Journal of the ACM*, vol. 65, no. 18, 2018.