Defending against VM Rollback Attack

Yubin Xia† ‡, Yutao Liu† ‡, Haibo Chen†, Binyu Zang‡
†*Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University*
‡*School of Computer Science, Fudan University*

***Abstract*—Recently it became a hot topic to protect VMs from a compromised or even malicious hypervisor. However, most previous systems are vulnerable to rollback attack, since it is hard to distinguish from normal suspend/resume and migration operations that an IaaS platform usually offers. Some of the previous systems simply disable these features to defend rollback attack, while others heavily need user involvement. In this paper, we propose a new solution to make a balance between security and functionality. By securely logging all the suspend/resume and migration operation inside a small trusted computing base, a user can audit the log to check malicious rollback and constrain the operations on the VMs. The solution considers several practical issues including hardware limitations and minimizing user's interaction, and has been implemented on a recent VM protection system.**

I. Introduction

Virtualization is a key enabling technology in today's cloud computing platform. As the code size and complexity of hypervisor and management VM increase constantly, recent researchers tend to remove them from the TCB (Trusted Computing Base) and build systems that can protect VMs from malicious hypervisors [1], [2].

In a virtualization environment, the hypervisor is able to suspend a VM at any point during execution, make a snapshot of its CPU states, memory and disk, and resume the snapshot later, without the guest VM's awareness. This feature supports useful functionalities such as fault tolerance and VM maintenance, and has been widely used. Unfortunately, it can also be leveraged by malicious attackers to launch VM rollback attack.

In a VM rollback attack, a compromised hypervisor runs a VM from an old snapshot without the user's awareness. Here, a *user* is the one who uses cloud service, and is the owner of guest VM(s). Since a part of history of VM's execution is lost, an attacker can bypass some security checks in the VM or undo some security critical updates. For example, an attacker launches a brute-force attack to guess the login password of a VM. Even if the guest OS has restriction on the number of failed trials (e.g., blocking for a while after three times, or erasing all data after ten times), the attacker can still infinitely rollback the VM to an initial state after each trial to clear the counter inside the VM and bypass the restriction. Another example is that an attacker may rollback a permission control module to undo a user's permission change, thus to expose user's later information to those who should be blocked. It is noteworthy that rollback attack is similar with but different from *replay attack*. In a replay attack, an adversary resends a previously seen message to a VM. While in a rollback attack, the VM itself is replayed [3].

Unfortunately, previous systems only focus on the protection of the integrity and privacy of user code and data, leaving rollback attacks a neglected issue. Because the snapshot itself is legal and internally consistent, the rollback attack can penetrate these security checks. There are several challenges to defend rollback attacks.

First, normal functionalities of virtualization must be preserved. It is usually hard to distinguish a rollback attack from a normal suspend/resume operation, since the cloud platform itself has no higher level semantic information. Some prior work simply disables the suspend/resume mechanism for security [4], which, however, renders some normal operations such as snapshot or VM migration unavailable.

Second, unnecessary user interaction should be minimized. Some systems, e.g., [4], require end users to get involved during VM booting, resuming or migration to ensure its freshness. It means every time the hypervisor starts, reboots, resumes or migrates a VM, it has to first ask the user for permission, which is annoying and not practical.

In this paper, we propose a solution to defend VM rollback attack, without sacrificing normal functionalities provided by hypervisor. The solution is based on the observation that the end user is the only one who can tell whether a rollback is malicious or not. Thus we securely log all the rollback activities of VM. By auditing the log, a user can either check suspicious rollbacks and ask the cloud operator to prove the necessity of such operations, or constrain the operations on a VM by define rollback policy in advance, or both. The solution also needs minimal user involvement.

The rest of the paper is organized as follows. The next section discusses the background and related work about researches on untrusted hypervisor and explains why they cannot defend rollback attacks. Section III describes the design and implementation of our solution. In section IV, the paper presents preliminary evaluation on security and performance impact of our solution. Finally, we summarize our conclusion in section V.

II. Background

A. Protection against Untrusted Hypervisor

Recently there has been plenty of research interest on defending malicious hypervisor [1], [5], [4], [2]. For example, CloudVisor [1] leverages a tiny nested hypervisor to isolate and conceal the states of guest VMs from the hypervisor layer. The tiny hypervisor protects the privacy and integrity of guest VMs by interposing the control transferring between the hypervisor and VMs to check each operation of the hypervisor is legal. It controls the EPT (Extended Page Table) of both guest VM and hypervisor to isolate their memory space, and encrypts guest memory page before hypervisor accessing. It uses a MHT (Merkel Hash Tree) to protect the integrity of VM's disk image, which is also encrypted. Figure 1 shows the architecture of CloudVisor.

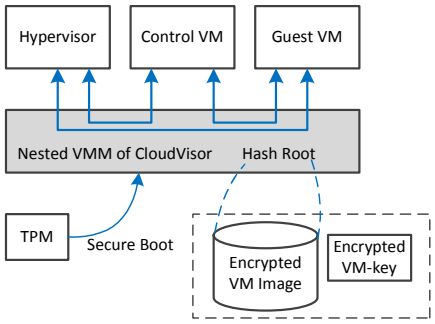

Figure 1: Architecture of CloudVisor

H-SVM [2] also separates the management of memory resources from the security protection. It uses microcode programs in hardware to enforce the memory protection. The hardware encrypts each memory page of guest VMs before it is accessed by the hypervisor, and leverages a hash tree to ensure the integrity of guest memory. The root of the hash tree is saved in protected memory region. H-SVM does not protect data in external devices but insists the guest VM itself to secure its I/O data. HyperWall [4] is another similar work. It extended hardware architecture to support fine-grained ownership control. A guest VM needs to specify explicitly which memory pages are private, and the architecture protects these pages from being accessed by the hypervisor. Those protected memory pages can never be accessed by the hypervisor.

B. Defending VM Rollback Attack

Both CloudVisor and H-SVM suffer from rollback attacks. Since both of them support VM suspend/resume and migration operations, a malicious hypervisor can easily leverage such operations to manipulate rollback attacks mentioned in section I. The VM rollback attack can be manipulated in multiple forms. For example, a hypervisor can resume a stale snapshot of a VM. It can also clone a VM to another machine by VM migration and control different versions. It is difficult to distinguish rollback attack from normal VM operations such as suspend/resume, boot/shutdown or migration.

Meanwhile, few details of the operations' implementation are described in previous researches, e.g., where is the roots of MHT (for memory, disk image and CPU state) saved when making VM snapshot, how to ensure the internal consistency of a snapshot, etc. These details are critical to VM security. For example, a malicious hypervisor may try to restore an old version of CPU state when scheduling a VM in. Previous researches prevented this attack by keep a hash of CPU state in protected area to ensure it is latest. However, an attacker can achieve the same goal by making two snapshots OLD and NEW, then uses snapshot-OLD's CPU state to replace the one of snapshot-NEW, and resumes snapshot-NEW. Therefore, protecting the integrity and continuity of memory, CPU state and disk *alone* is not enough, which, unfortunately, is not mentioned by previous researches.

In order to prevent rollback attack, some systems simply disable the function of suspend/resume or hypervisor controlled migration. For example, HyperWall [4] provides no way to make snapshot, since the hypervisor cannot access the memory pages that guest VM asks to protect. The hypervisor can not migrate a VM neither. Only self-migration [6] is supported, which is done by the guest VM itself. Therefore some features of modern cloud center based on suspend/resume are disabled. HyperWall also requires user to be involved during VM booting and resuming, which makes it annoying and not practical.

C. Threat Model

The threat model in this paper is similar as CloudVisor [1]. The hypervisor is untrusted since its code size is increasing so tremendously that it contains more and more bugs and vulnerabilities [7], [8]. Thus we remove it from the trust computing base (TCB). In addition, we assume that the VMs are running in a hardware-safe environment, so that physical attacks such as cold-boot attack [9] or intentional power-off attack are not considered. It is also not our goal to protect from side-channel attacks [10], DoS attack or inside-VM bugs.

III. Design and Implementation

In this section, we present a solution to defend rollback attack from untrusted hypervisor while still provide the abilities of VM suspend/resume and migration. The solution is implemented based on CloudVisor [1].

We first define a *running epoch* of a VM as a period that starts from booting or resuming and ends with shutdown or suspending, as shown in Figure 2. In order to defend rollback attack in virtualized environment, the underlying secure mechanism assures following two properties:

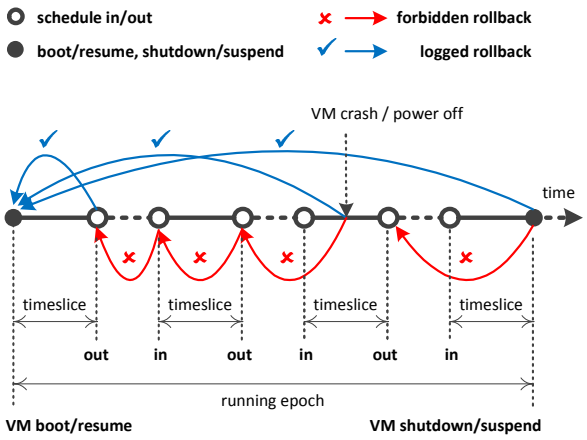

Figure 2: Running Epoch of VM

- It is forbidden to rollback a VM to the middle of an epoch.
- The start and end of epoch are safely logged.

By defining and logging the running epoch of VM, the rollback process is auditable. A user can analyze the log to detect rollback attack, or to constrain the behavior of cloud providers to prevent rollback attack.

A. VM Operations

We add four *hypercall*s in CloudVisor: *vmboot*, *vmshutdown*, *vmsuspend* and *vmresume*. The four *hypercalls* are all invoked by the control domain (domain-0), and thus the hypervisor is kept unmodified. The control domain cannot bypass the *hypercalls*, otherwise the operations would fail.

VM Suspend: When a hypervisor suspends a running VM, it creates a snapshot of the VM, which is composed of three parts: the states of register, the content in memory and the image on disk. During runtime, all the three parts are encrypted and protected by hash trees or hash, and the roots of the hash trees are kept in protected memory zones of CloudVisor, as shown in Figure 3. When *vmsuspend* is invoked, CloudVisor calculates a *snapshot hash* out of the three hash values. The hash is then encrypted with the VM key. The encrypted *snapshot hash* is unique to each snapshot and is used as the version for identification.

VM Resume: When the hypervisor resumes a VM, *vmresume* is called with the snapshot and *snapshot hash* as parameters. Before resuming, CloudVisor first checks the *snapshot hash* is legal by calculating and comparing with the hash of the snapshot. Thus a malicious hypervisor cannot use snapshot-A's CPU states and snapshot-B's memory and disk to resume a VM, It cannot resume a VM into the middle of an epoch as well. CloudVisor then logs the operations just as in VM suspend operation.

VM Boot: *vmboot* is invoked when a VM is booting. It uses a predefined memory image and CPU state. The memory image contains only the vmloader, and the CPU state contains EIP pointing to a fixed address to the first instruction. Thus the hashes of both are also fixed and have already been known by CloudVisor. Therefore, the *vmboot* needs only the hash of disk image to identify the version of the VM.

VM Shutdown: Similarly, when shutting a VM down, the CloudVisor will purify all its memory and CPU registers, thus the *snapshot hash* is just the hash of disk image. However, a VM may be shut down in an abnormal way, e.g., server crashing or power failure. In such cases, *vmshutdown* is not invoked, which means the root hash of disk is not logged. A user can easily detect such failure by the log, as shown in section IV.

VM Migration: The process of migration can be seen as first suspending on the source node, then resuming on the destination node. In live migration, the suspend/resume happens at last iteration. Meanwhile, the source node needs to transfer the *snapshot hash* to the destination node along with the snapshot. The physical server must has the corresponding VM key, which is done through the process of secure key-switching that is already supported by CloudVisor.

B. Secure Logging

The logging is done by CloudVisor within the four hypercalls. Every single entry of the log contains a triple of {operation, timestamp, snapshot version}. The operation is either "boot", "shutdown", suspend" or "resume". In order to get trusted timestamp, we cannot depend on the clock of the hardware which is controlled by the potentially malicious hypervisor. Instead, a commercial secure time service [11], or a dedicated secure server can be used to offer real time. The snapshot version is composed of three parts: the root hash of disk image, memory content and CPU state, as illustrated in Figure 3.

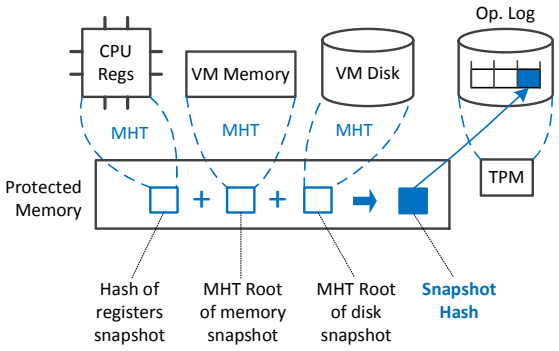

Figure 3: Root of Snapshot

To make the log tamper resistant, we leverage TPM to protect its integrity. Continuous integrity measurement is used that every time a new entry is appended to the log, a hash saved in NVRAM of TPM is updated according to the value of the new item. TPM is also used to make signature on the log as reply to user's challenges. The logging does

not depend heavily on NVRAM of TPM since such VM operations are rare. Assuming a VM runs one epoch on a platform per day, and 16 VMs running on a single platform, then the TPM can be used for more than 17 years.

Because a VM can be migrated or cloned to different physical machines, it log may be distributed on multiple machines. Once the user asks for log to audit, the cloud provider should provide the entire set of log. However, a malicious cloud operator may hide some log from user, which may contains evidence of rollback attacks.

In order to keep the integrity of the whole log set, we need to record all the physical machines that have ever hosted the VM. At deploying phase, the user sends an encrypted VM image to a physical machine, as well as the key of VM image, which is encrypted by the public key of the TPM of the machine. The machine is considered as the *root node*. Once the VM is migrated or cloned from the *root node* to other nodes, the VM-key is transferred in advance. During this key transferring, both the source and destination nodes are logged. Thus the user can get the full set of nodes one by one alone the log chain as long as he knows the root node, and a malicious operator cannot hide any node.

C. Discussion

There are two potential approaches: one is to always verify a snapshot by CloudVisor, the other is to make a snapshot "self-proven". The major difference is that whether the *snapshot hash* is under the full control of CloudVisor. In the first method, CloudVisor never exposes *snapshot hash* to the hypervisor. The hypervisor gets only the snapshot content when it suspends a VM, while the hash of the snapshot is saved inside CloudVisor's memory and disk, both of which are protected from the access of the hypervisor. During resuming, CloudVisor will recalculate the hash of the snapshot and compare with the hash it saved before. Since there maybe multiple snapshots of a single VM, a list of *snapshot hash* must be saved for verification. Meanwhile, CloudVisor also needs to send the corresponding hash to the destination nodes before migration.

In the second way, the *snapshot hash* is encrypted with the VM-key and exposed to the hypervisor as well as the snapshot itself. Then the hypervisor can copy the both to any node that has the corresponding VM-key and resume the snapshot. The integrity of both are ensured since they are co-related with each other and are encrypted. Thus no other information of the snapshot needs to be maintained by CloudVisor except the log. In this paper we adopt the second method because it is more simple, flexible and practical.

IV. Preliminary Evaluation

A. Security Evaluation

Once a user gets the log set of a VM, he can check the log and analyze the number, frequency, and timestamps of

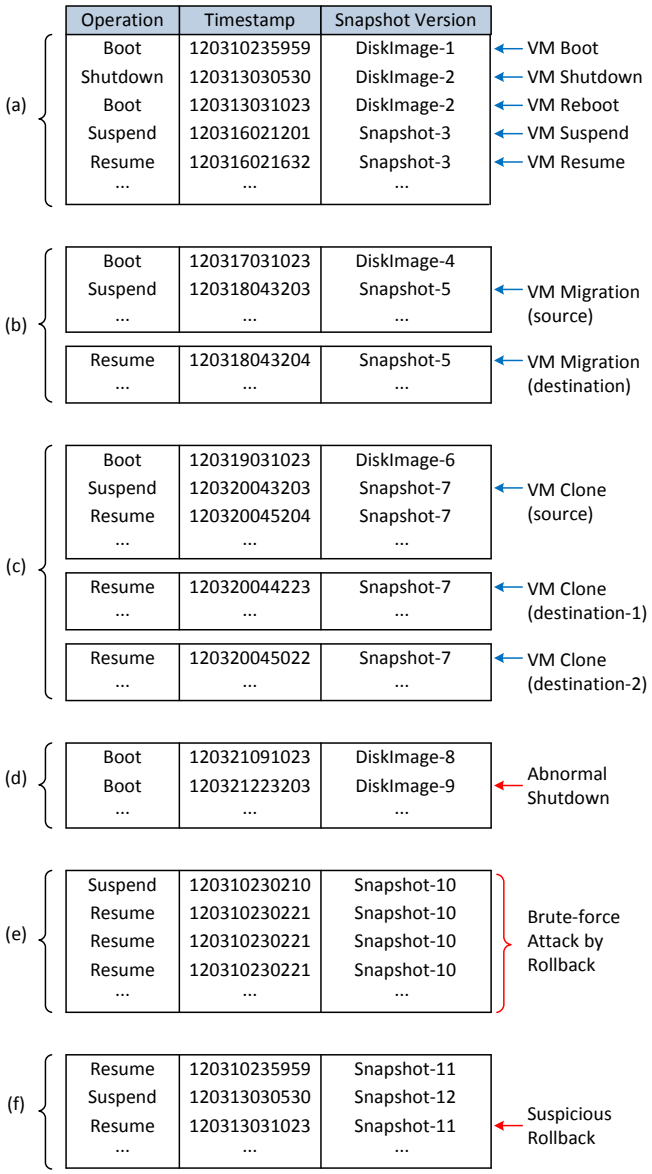

Figure 4: Log Samples

VM behaviors to audit the operations on the VM and detect malicious rollbacks.

Figure 4-a shows a log of normal VM execution process. The frequency of logging is quite low since it is rare to reboot or migrate a VM. Figure 4-b shows the process of VM migration. When the VM is migrated, it is suspended at the source node and resumed at the destine node. If the VM resumes to run after migration, it is called *VM clone*, as illustrated in figure 4-c. The snapshot from one source node can be cloned to multiple destination nodes.

Figure 4-d presents the case of abnormal shutdown, e.g., power failure or whole system crash. Thus the end of the epoch is not logged. A malicious operator can fake a system crash to do rollback attack. A user can then ask for other

evidence to show the reason of such unexpected situation.

Figure 4-e shows the log of brute force password attack as mentioned before. The attacker tries to bypass the limitation of false password, since all the rollback will be logged, the attacker cannot hide his behavior. The user can easily finds that there are abnormally large number of rollback and thus detect the attack.

Once the cloud operator make a rollback, the user can only consider it as a suspicious action, as shown in Figure 4-f. It is the cloud provider's responsibility to prove the rollback is benign by showing other evidence to explain why it is necessary to do so, e.g., the log of crash or record of power failure.

Although the mechanism cannot ensure 100% security that a sophistic attacker may still bypass the log checking once knows user's policy, our solution offers users a meaning to constrain the behavior of cloud provider. For example, a user may require that a VM cannot be rollbacked to a version earlier than some timestamp, since after that time several security critical configurations have been changed. Meanwhile, the solution also make the attack significantly harder, and thus increase the cost of attack.

B. Performance Impact

Overhead of Suspending: In original CloudVisor, when a VM is being suspended, all of its memory pages are encrypted and a hash tree is calculated. The hash of CPU state is already there during the VMEXIT, and the disk hash is always kept fresh as every disk write. Thus the only additional work is to get a *snapshot hash* out of the three hash values, get a timestamp and log the corresponding operation.

Overhead of Resuming: When resuming a VM, CloudVisor needs to calculate the hash of the snapshot before continue the execution of the VM, which would take long time, especially if the VM has been configured with large memory. It may increase the downtime of live migration. As an optimization, the hash tree of memory and the disk are kept with the contents. Thus for memory, the check can be delayed until the first access of the page, similar with the process of disk access.

Overhead of Boot and Shutdown: For VM booting and shutdown, since no hash check is needed for CPU state and memory, the only new overhead comes from the logging, which needs to access the disk and update storage in TPM.

Finally, we argue that these VM operations are rare in common use. Thus the overhead is not critical. Meanwhile, the log size is small enough to be ignored which takes little disk space.

V. Conclusion

In this paper, we discussedd the issue of VM rollback attack from malicious hypervisor in a virtualization execution environment. We showed that the biggest challenge is to identify rollback attacks from normal VM operations such as suspend/resume for fault tolerance or VM migrations. It needed to make a tradeoff between security and functionality. We have made the following two observations: First, the most significant characteristic of rollback attack is "user-unawareness". By safe logging start/resume and suspend/shutdown operation of a VM, a user can audit the log to detect rollback attack, and can constrain the operations on a VM in advance. Second, normal suspend/resume operation for VM maintenance is rare. Multiple factors were considered in the solution, including hardware limitations, unnecessary user interaction minimizing, and overhead on performance and storage. We implemented the solution based on CloudVisor and did some preliminary evaluation of its security and performance impact.

References

[1] F. Zhang, J. Chen, H. Chen, and B. Zang, "CloudVisor : Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization," in *Proc. SOSP*, 2011, pp. 203–216.

[2] S. Jin, J. Ahn, S. Cha, and J. Huh, "Architectural Support for Secure Virtualization under a Vulnerable Hypervisor," in *Proc. MICRO*, 2011.

[3] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune, "Memoir : Practical state continuity for protected modules," in *IEEE Symposium on Security and Privacy*, 2011.

[4] J. Szefer and R. B. Lee, "Architectural Support for Hypervisor-Secure Virtualization," in *Proc. ASPLOS*, 2012.

[5] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the Hypervisor Attack Surface for a More Secure Cloud," in *Proc. CCS*, 2011.

[6] J. G. Hansen and E. Jul, "Self-migration of operating systems," in *Proc. of the 11th ACM SIGOPS European Workshop*, 2004.

[7] N. V. Database, "Cve and cce statistics query page," http://web.nvd.nist.gov/view/vuln/statistics.

[8] J. Rutkowska and A. Tereshkin, "Bluepilling the Xen Hypervisor," *Black Hat USA*, 2008.

[9] J. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[10] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proc. CCS*. ACM, 2009, pp. 199–212.

[11] "e-timestamp: A web-based security service for data authentication," http://www.digistamp.com.