A Survey on the Security of Hypervisors in Cloud Computing

Andrew R. Riddle and Soon M. Chung
Department of Computer Science and Engineering
Wright State University
Dayton, Ohio 45435, USA
{riddle.16, soon.chung}@wright.edu

***Abstract*—This survey paper focuses on the security of hypervisors in the cloud. Topics covered in this paper include attacks that allow a malicious virtual machine (VM) to compromise the hypervisor, as well as techniques used by malicious VMs to steal more than their allocated share of physical resources, and ways to bypass the isolation between the VMs by using side-channels to steal data. Also discussed are the security requirements and architectures for hypervisors to successfully defend against such attacks.**

Keywords—security, hypervisors, cloud computing

I. Introduction

Cloud computing is the platform of choice for many web services. The key underlying technology that makes the cloud possible is virtualization, the ability to run multiple virtual machines (VM) on a single hardware platform. This has the benefit of reducing costs for the customer and allowing the customer to quickly change server provisioning to meet the demand on their web services. But it requires multiple VM tenants who do not trust each other to share the same physical hardware.

The hypervisor is the piece of software that manages resources and isolates the VMs from each other. Figure 1 shows the architecture of the Xen hypervisor [10]. If a malicious VM is able to break free from the isolation that the hypervisor is supposed to guarantee, or compromise the hypervisor itself, then all the other VMs on the same physical machine are vulnerable. The malicious VM could also launch a denial of service attack to steal resources and slow down the other co-resident VMs.

This should be very concerning for any organization using the cloud that processes and stores sensitive customer data such as credit card information or social security numbers, and for organizations that perform highly proprietary work on the cloud. A part of the problem is that, as a customer using the cloud, you are not aware of who else is co-resident on the same physical hardware as your VMs.

Techniques discussed in this paper will show how it is possible for someone to get their VMs co-resident on the same physical hardware with their target VMs by using side-channels. Once co-resident, the malicious VM can attempt to compromise the hypervisor, or steal data using a side-channel, or slow down the target VM by stealing physical resources.

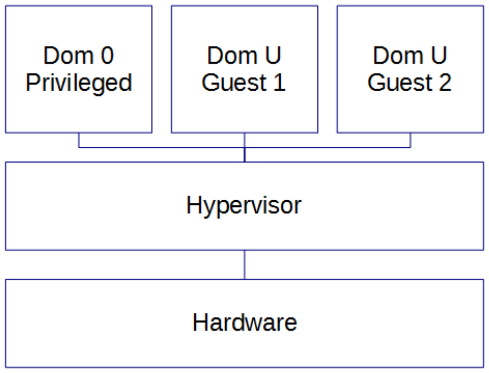

Fig. 1. The Xen Hypervisor [10].

This paper is organized as follows: Section II describes the ways of becoming co-resident with a target VM in detail since that is the first step in many attacks. Section III covers stealing data via side-channels, then Section IV describes how performance based attacks can be used to slow down the target VMs. Section V discusses techniques for securing hypervisors and attacks that compromise the hypervisor. Section VI discusses security requirements and architectures to secure the hypervisor. Section VII contains some conclusions.

II. Co-Residency of VMs

The first step to launching an attack against a target VM is for the malicious VM to be co-resident on the same physical hardware with its target VM. One way this can be accomplished is via a side-channel. A side-channel is a method of communication using a medium not originally designed for data transfer. A side-channel typically makes use of shared resources, such as the CPU cache, memory, network, power consumption, etc. to extract information. By analyzing the behavior of the hardware, information about what is occurring in the software can be inferred. Malicious VMs can then extract information from other co-resident VMs by monitoring the hardware. This usually takes the form of analyzing the memory and cache latency to transfer information [4].

In [16], they make use of the response time of the CPU cache to determine if a target VM is co-resident. They make use of a load preprocessor using a cubic spline and a load predictor using linear regression to analyze the behavior of the cache. First the malicious VM primes as much of the CPU cache as possible, then it places a load on the target VM by generating many normal data requests to the web service. Then the load measurement step is run on the malicious VM to determine the cache access time. The higher the cache access time, the more activity by other co-resident VMs can be assumed because the data that was used to prime the cache would be evicted. They demonstrated that their technique will work even when there is a third noisy VM co-resident with both the malicious and target VMs [16].

III. SIDE-CHANNEL ATTACKS AND DEFENSES

In addition to determining VM co-residency by using the cache access time, side-channels can also be used to covertly steal information from a target VM. In [9] they assume that the target VM has been infected with malware due to a vulnerability in the software running on that VM and the malicious VM which is co-resident with the target. The goal of the attackers is then to steal information covertly from that VM without leaving a trace. This can be done by using memory bus contention. To send a bit of 1, the target VM will issue an atomic CPU instruction to lock the memory bus, thereby increasing the memory latency access time for other CPUs. If the memory bus is unlocked then the latency access time is shorter, and this transfers a bit of 0. Other side-channels can be built in a similar manner by exploiting cache contention to manipulate latency times [9].

In this attack, the length of the overlapping execution times of the target and malicious VMs is important to determine the bandwidth of the side-channel. The longer the VMs are running concurrently, the greater the bandwidth of the side-channel [9].

This attack can be defended by making some changes to the scheduler in the hypervisor. To successfully defend, the scheduler can try to limit the overlapping execution times of any two VMs on the system while maintaining an acceptable level of performance. The scheduler should still maintain fairness because it doesn't know which VM is malicious. In order to maintain an acceptable level of performance, the scheduler should limit the frequency of VM switching which reduces performance [9].

Since limiting the overlapping execution times may decrease performance, another possible defense to disrupt the side-channel is for the hypervisor to inject noise into the side-channel. This will increase the error rate and reduce the effective bandwidth. The noise is implemented by way of atomic memory access to defend against the memory bus contention side-channel. When switching between virtual CPUs, the hypervisor will issue random atomic memory access commands to add noise to any side-channel [9].

Other ways to defend against side-channels are presented in [4, 13]. In [13], they presented XenPump, as an addition to the Xen hypervisor, which is designed to limit the effectiveness of timing channels. XenPump adds random latencies into the system to limit the bandwidth of timing channels. This confuses the VM on the receiving end of the side-channel because it cannot tell whether the latency is from the transmitting VM or from the hypervisor. This has the trade-off of decreasing system performance.

In [4], they proposed a solution to solve the cache-based side-channel. The attack is similar to the one discussed in Section II. It is based on prime-trigger-probe where the probing VM tries to fill up the cache by accessing as many lines as possible and records a baseline cache access time. Then the target VM is triggered to run and encodes a message by accessing parts of the cache. Once the target has completed running, the probing VM then accesses the cache; and each cache line that the target has accessed will cause a cache miss and have a higher access time than the baseline. A diagram of the prime-trigger-probe cache based side channel is shown in Figure 2.

To defend against this side-channel, they proposed disabling the overlapping aspect of the cache for the VM tenants on a machine. Another option is to flush the cache when switching between VM domains. This adds a 15% performance overhead [4].

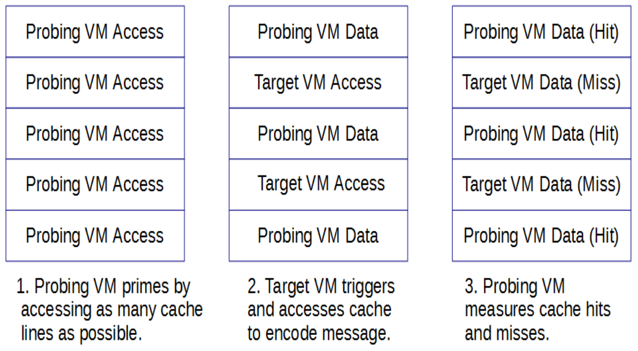

Fig. 2. Prime-trigger-probe method [4].

IV. PERFORMANCE-BASED ATTACKS

Performance-based attacks can be used to slow down other co-resident VMs or use more than the allocated share of resources to steal from the cloud provider. In [17] they designed a scheme for the malicious VM to cheat the scheduler and achieve up to 98% of the CPU usage on the physical machine. By the way, Amazon EC2 charges its customers based on the amount of time the VM is running, not by actual CPU cycles. The Xen hypervisor uses the credit scheduler which is an implementation of the token bucket algorithm. In the token bucket algorithm, credits are distributed at a constant rate to each VM. The VMs are allowed to store up to 300 credits. On each scheduler tick, the scheduler will debit credits from the currently running VM [17].

In their attack, the attacker VM yields just before the next scheduler tick. This always ensures that another co-resident VM is running when the scheduler executes. In non-boost mode, this will cause the attacker to never have its CPU credit debited. In boost mode, the hypervisor is unable to tell the difference between a VM awaking after it deliberately yielded and the one waking for an event such as an interrupt. So after the attacker deliberately yields during the scheduler tick, it can preempt the currently running VM. On Amazon's EC2, this can get around their 40% cap for any VM and use up to 85% of the CPU. This can also be used by a malicious user for co-residency purposes. By having two co-resident VMs run this attack simultaneously, each VM receives 42% of the CPU. With this knowledge, the malicious user starts up two VMs and starts the attack program on each VM. Each VM then measures the CPU share it receives. If it is 85%, then they are not co-resident; if it is 42%, then they can assume that they are co-resident [17].

There are two solutions to this problem. The first uses an exact scheduler which is based on a high precision clock to measure CPU usage time whenever a VM yields and goes idle. The other is to use a randomized scheduler, where the scheduler will tick at random intervals between 10 and 30 milliseconds. Figure 3 shows how the randomized scheduler tick rate prevents a VM from being preempted. In this example, the attacker yields, attempting to starve VM1 by allowing it to run for only a short period of time before the next scheduler tick, but the tick rate has been randomized so VM1 is still able to get its fair share of time [17].

In [15] they focused on using I/O performance based attacks to slow down a co-resident target VM. They deploy specially crafted I/O workloads and manipulate the shared I/O queues to reduce the performance of a target VM. The first task to carry out this attack is to extract the scheduling characteristics of the hypervisor, and then use that information to reduce I/O performance by overloading the I/O resources. However, this attack could fail if there are multiple hard disks installed on the physical machine.

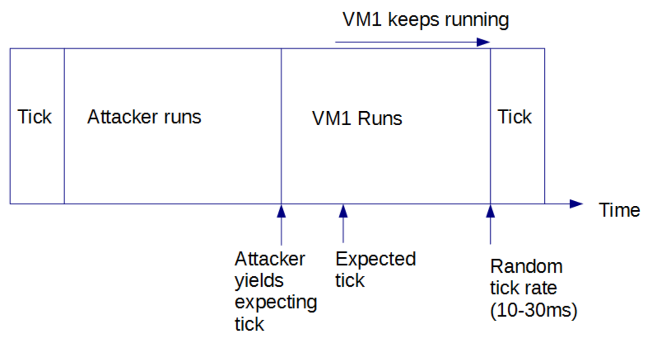

Fig. 3. Randomized scheduler tick rate prevents a VM from being preempted [17].

V. HYPERVISOR ATTACKS AND DEFENSES

In addition to stealing information via side-channels, or launching performance-based attacks to slow down target VMs, a malicious VM can also attempt to compromise the hypervisor. This is known as *virtual machine escape*.

A. *Hypervisor Control Flow Integrity*

HyperSafe [12] is designed to provide control flow integrity to the hypervisor. The Trusted Platform Module (TPM) is a hardware based module that provides secure storage and secure attestation in addition to cryptographic hashes and signatures [7]. TPM can provide load-time integrity for the hypervisor, but the challenge is to provide run-time integrity. Two techniques are proposed in HyperSafe to solve the run-time integrity checking problem.

The first method in HyperSafe is to implement a non-bypassable memory lockdown. This locking down of memory pages prevents any unauthorized writes to the pages. The unlocking process is designed in such a way that it prevents any modifications to hypervisor code and data. This method prevents any malicious code from being injected into the control flow of the hypervisor, and it is implemented as an extension to the Xen hypervisor [12].

The second method in HyperSafe is called restricted pointer indexing to essentially add a layer of indirection to all pointers. It uses the previously discussed technique of memory lockdown, and pre-computes the control flow targets and then stores them in a table. It ensures that the call and return targets follow the control-flow graph. It is implemented as a compiler extension, so it requires no code changes to the hypervisor [12].

B. *Hypervisor Integrity Checking*

Another solution to secure the hypervisor is called HyperSentry [1]. HyperSentry uses a software component that is isolated from the hypervisor to provide stealth and in-context integrity checking of the hypervisor. This solution does not introduce a higher and more privileged layer, but instead uses existing hardware and firmware for the software integrity component and isolates it from the hypervisor by using TPM. The key for HyperSentry to work is stealth, so that it will not be vulnerable to the scrubbing attack which removes all evidence of an attack when a measurement by a higher software layer is detected. This is done by using an out of band channel to trigger HyperSentry. The out of band channel used by HyperSentry includes the Intelligent Platform Management Interface (IMPI), System Management Mode (SMM), and Baseboard Management Controller (BMC).

IMPI is a platform management interface implemented in hardware and firmware. All its functions operate completely independently of the CPU and all software on the system. The BMC is installed on the motherboard and is the interface between the remote verifier and the hardware management component. IMPI triggers the SMM which provides a secure environment that the software running on the machine cannot manipulate [1]. Figure 4 shows a graphical representation of the interaction between these components. HyperSentry also provides a measurement agent that is verifiable, deterministic and non-interruptible. The measurement agent is in-context, so it will save the CPU state and restores it later which allows checking the current state of the CPU. Finally HyperSentry provides attestation to the authenticity of the integrity measurement outputs [1].

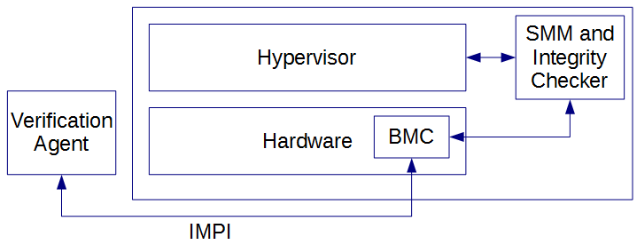

Fig. 4. HyperSentry architecture [1].

C. *Return Oriented Programming Attack on Hypervisors*

An attack using return oriented programming (ROP) can be used to mount a successful attack against the Xen hypervisor [2]. Return oriented programming attacks make use of existing code. By chaining sequences ending in a return statement together, a Turing-complete language can be created. This gives a work around to the data execution prevention (DEP) implemented on many systems today as a security measure. The goal of the ROP attack is to modify the data in the hypervisor that controls the VM privilege level. This way, an attacker can escalate their VMs to a privileged state.

A defense to ROP is presented in [5], in which they continuously analyze the stack, looking for possible ROP attacks and then quarantine for further investigation. They use a key feature of ROP that requires many addresses in the range of a program and its libraries to search for such attacks.

D. *Modifying Non-control Data*

Other attacks attempt to modify a hypervisor's non-control data are described in [3]. There are three types of non-control data an attacker could attempt to modify. The first is the privilege level data which could escalate a VM to a more privileged state. The second is resource utilization data which could let an attacker gain more than their fair share of physical resources. The third is security policy data which could let a sensitive VM run on a machine with other VMs that would then attempt to steal its data via side-channels.

To successfully execute a non-control data attack, it would be helpful to know the version number of the hypervisor. This would allow the attacker to more accurately calculate the memory offsets for the non-control data it wishes to modify. To maintain non-control data integrity, they proposed to use hardware features to only allow certain functions to write the memory locations where the non-control data is located. This could use a non-bypassable memory lockdown as previous discussed in HyperSafe [3].

E. *VM Rollback Attack*

The VM rollback attack [14] assumes that the hypervisor has already been compromised. The compromised hypervisor will then execute a VM from an older snapshot. This happens without the VM owner's awareness. This attack results in a part of the target VM's execution history being lost, which could let the attacker bypass security systems or undo security patches and updates applied to the target VM. This is demonstrated in Figure 5. An attacker is trying to brute force a password. When the target VM triggers a security alert, the compromised hypervisor rolls back to the previous snapshot to allow it to continue the brute force attack [14].

The first requirement for a solution to the rollback attack is that it must keep the suspend/resume functions. Keeping the suspend/resume functions makes the task of developing a solution more difficult, because the solution needs to be able to distinguish between a normal suspend/resume and a rollback attack. The second requirement is that the solution should not place a heavy burden on the owner of the VM [14].

One solution is to securely log all the rollback actions, and then the users can audit the log as they see fit. TPM would be used to protect the integrity of the log. This would require four hypercalls for VM boot, VM shutdown, VM suspend, and VM resume, that would all securely log the corresponding action. Another solution is using protected memory to isolate and encrypt the VM's memory pages from the hypervisor, thereby preventing the hypervisor from modifying or reading the pages [14].

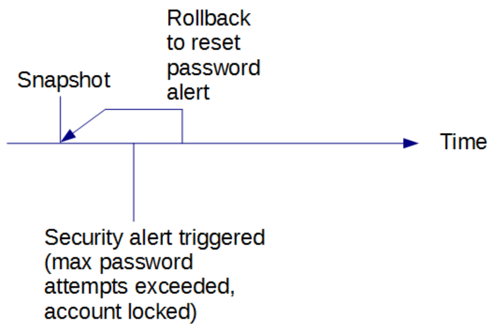
Rollback to reset password alert
Snapshot
Time
Security alert triggered (max password attempts exceeded, account locked)

Fig. 5. Rollback of the compromised hypervisor to the previous snapshot to continue the brute force attack [14].

VI. VM ISOLATION TECHNIQUES

There is a wealth of literature on security architectures and security requirements for virtualization. We will discuss four different approaches in this section. An architecture called Secure Turtles [8] makes use of nested virtualization to protect a guest VM. The Level 0 hypervisor is the most privileged and protects the VM running in Level 2 from the attacks by Level 1, even in the case Level 1 has been compromised. There are four requirements for Secure Turtles. The first is that there should be lifetime kernel code integrity of the Level 1 hypervisor. The second is that there should be code and data integrity of the qemu-kvm daemons. The third states that there should be data integrity of the Level 2 guest VM. The fourth requirement is that the Level 2 guest VM should be aware of any violations of the first three requirements. Secure Turtles assumes that the Level 0 hypervisor is protected from outside attacks.

In [11], they took a different approach altogether and proposed eliminating the hypervisor almost entirely. Virtualization software is a prime target for attacks, but minimizing the code base to reduce vulnerabilities is not a good answer because it reduces functionality. They also stated that maintaining multi-tenancy is a requirement for the cloud to provide on-demand resources for customers and to leverage economies of scale to reduce costs. Their solution is for a temporary hypervisor which only runs at initialization that pre-allocates hardware resources such as processor cores and memory, sets up virtualized I/O devices, and avoids indirection to bring the VM in more direct contact with the hardware. Virtualized I/O devices are used because allocating individual I/O devices to each VM may not be practical. Once allocated, the hardware will enforce the isolation between the virtual machines without the need for a hypervisor.

In [10], a mandatory access control (MAC) policy is described to allow coalitions of VMs to share resources such as network, disk, memory, events, and domain operations. This coalition could even be distributed among multiple hypervisors. It uses bind-time authorization for high performance and includes a Chinese-wall. In the Chinese-wall, each VM is assigned a type, and the types that conflict with each other are not allowed to run concurrently. This prevents covert channels by not allowing a VM that processes sensitive data to run at the same time with a competitor's VM [10]. Type-enforcement is used to specify which VMs have access to which resources.

In [6], a multi-level set of security requirements is described. Two types of hypervisors are presented: the pure isolation and sharing hypervisors. The pure isolation hypervisor divides a machine into partitions and does not allow any sharing of resources other than CPU and memory. The sharing hypervisor allows the sharing of files. A high security partition has read-only access to a lower level security data. The low level partition gets read/write to the same low lever security data. This can be implemented as a one way network. Another implementation uses a secure shared file store. Access is controlled by a subsystem of the hypervisor using a cross-ring call or as a separate partition using message passing.

VII. CONCLUSION

In conclusion, virtualization is a key enabling technology of the cloud, allowing many guest machines to share the same physical hardware. The hypervisor is supposed to be secure from attacks and effectively isolate the VMs, yet potential security flaws are evident. VM escape is the most serious of all the attacks discussed because the escaped VM would be free to compromise all the other co-resident VMs. Solving the VM escape problem could potentially require a paradigm shift in the architecture and design of hypervisors. Other attacks that focus on stealing data via side-channels can be mitigated by adding noise to the side-channel. Solving the problem of hypervisor security is an important step in providing a secure cloud environment for businesses and consumers to utilize.

REFERENCES

[1] A. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. Skalsky, "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity," Proc. of 17th ACM Conference on Computer and Communications Security, 2010, pp. 38–49.

[2] B. Ding, Y. Wu, Y. He, S. Tian, B. Guan, and G. Wu, "Return-Oriented Programming Attack on the Xen Hypervisor," Proc. of 7th International Conference on Availability, Reliability and Security, 2012, pp. 479–484.

[3] B. Ding, Y. He, Y. Wu, and J. Yu, "Systemic Threats to Hypervisor Non-control Data," Information Security, 7(4), 2013, pp. 349–354.

[4] M. Godfrey and M. Zulkernine, "A Server-Side Solution to Cache-Based Side-Channel Attacks in the Cloud," Proc. of 6th IEEE International Conference on Cloud Computing, 2013, pp. 163–170.

[5] X. Jia, R. Wang, J. Jiang, S. Zhang, and P. Liu, "Defending Return-oriented Programming Based on Virtualization Techniques," Security and Communication Networks, 6(10), 2013, pp. 1236–1249.

[6] P. Karger, "Multi-level Security Requirements for Hypervisors," Proc. of 21st Annual Computer Security Applications Conference (ACSAC 2005), 2005, pp. 267–275.

[7] M. Kim, H. Ju, Y. Kim, J. Park, and Y. Park, "Design and Implementation of Mobile Trusted Module for Trusted Mobile Computing," IEEE Transactions on Consumer Electronics, 56(1), 2010, pp. 134–140.

[8] F. Liu, L. Ren, and H. Bai, "Secure-Turtles: Building a Secure Execution Environment for Guest VMs on Turtles System," Journal of Computers, 9(3), 2014, pp. 741–749.

[9] F. Liu, L. Ren, and H. Bai, "Mitigating Cross-VM Side Channel Attack on Multiple Tenants Cloud Platform," Journal of Computers, 9(4), 2014, pp. 1005–1013.

[10] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. Griffin, and L. van Doorn, "Building a MAC-based Security Architecture for the Xen Open-source Hypervisor," Proc. of 21st Annual Computer Security Applications Conference (ACSAC 2005), 2005, pp. 276–285.

[11] J. Szefer, E. Keller, R. Lee, and J. Rexford, "Eliminating the Hypervisor Attack Surface for a More Secure Cloud," Proc. of 18th ACM Conference on Computer and Communications Security, 2011, pp. 401–412.

[12] Z. Wang and X. Jiang, "HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity," Proc. of IEEE Symposium on Security and Privacy, 2010, pp. 380–395.

[13] J. Wu, L. Ding, Y. Lin, N. Min-Allah, and Y. Wang, "XenPump: A New Method to Mitigate Timing Channel in Cloud Computing," Proc. of 5th IEEE International Conference on Cloud Computing, 2012, pp. 678–685.

[14] Y. Xia, Y. Liu, H. Chen, and B. Zang, "Defending against VM Rollback Attack," Proc. of 2nd International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV 2012), 2012.

[15] Z. Yang, H. Fang, Y. Wu, C. Li, B. Zhao, and H. Huang, "Understanding the Effects of Hypervisor I/O Scheduling for Virtual Machine Performance Interference," Proc. of 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2012), 2012, pp. 34–41.

[16] S. Yu, X. Gui, J. Lin, X. Zhang, and J. Wang, "Detecting VMs Co-residency in the Cloud: Using Cache-based Side Channel Attacks," Elektronika Ir Elektrotechnika, 19(5), 2013, pp. 73–78.

[17] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram, "Scheduler Vulnerabilities and Coordinated Attacks in Cloud Computing," Journal of Computer Security, 21(4), 2013, pp. 533–559.