Mitigating Cross-VM Side Channel Attack on Multiple Tenants Cloud Platform

Fei Liu, Lanfang Ren, Hongtao Bai
Editorial Department
Institute of Security China Mobile, China

***Abstract*—Virtualization is a key enabling technology in cloud computing. Multiple tenants can share computing resource of cloud provider on demand. While sharing can reduce the expenses of computing, it brings security vulnerability as well since the isolation between different VMs could be violated through side-channel attacks. Recent researches point out that by leveraging memory bus contention, two colluded malware within different VMs (but on the same host) may use diversity of memory access latency as a covert channel to deliver security critical information, such as user passwords or credit card numbers, which can bypass access control policies enforced by the guest OS or even the hypervisor. The bandwidth of such covert channel could be up to hundreds of kilobytes per second, which is fast enough to transfer large data objects. In this paper we propose a covert channel aware scheduler that considers security as first class to mitigate such side-channel attack. The scheduler is able to control the execution time overlapping of different VMs, and can also inject noise periodically to mitigate the threat of potential side channels. We have built a prototype of the proposed scheduler that enables overlapping control and noise injection. The performance evaluations show that the overhead introduced is acceptable. Meanwhile, the new scheduler offers the user to dynamically configure scheduling parameters to adapt to diverse circumstances, in order to make a balance between performance and security.**

***Index Terms*—virtual machine, cloud computing, security, side channel attack, scheduling algorithm**

I. Introduction

Security issue has become the most significant concern of cloud computing, according to xxx's report. However, virtualization technology, which is used heavily by cloud providers to provide workload consolidation, brings new challenge to cloud security. One of the challenges is to enforce the isolation between different VMs (Virtual Machines) and only allow legal communications, in order to protect user's security critical data from leakage. Both cloud provider and users have various mechanisms to defend against data leakage. For example, a guest VM can deploy DIFC (Decentralized Information Flow Control) mechanism to track and control the flow of specific data, e.g., not allowing a private key to be sent through network. However, an attacker may still transfer data across different VM by using side channel to bypass such security mechanism.

Side channel attacks leverage the shared resource between unrelated entities to build communication channel between them. Such channels are usually unintended and thus are not controlled. On cloud platform where a physical host is shared by different VMs, attackers find more ways to construct such side channel. Ristenpart et al. exploit last level cache as side channel on Amazon EC2 platform, with bandwidth of 0.2 bps (bits-per-second), which is considered as a slow one. Later researchers keep increasing the bandwidth. Zhenyu et al. use more ways, including memory bus contention, to achieve hundreds of kilobytes bandwidth, which is fast enough to transfer not only small data, such as password and credit card number, but also larger files such like personal photos or blueprints, which has become a serious threat.

Figure 1 shows how a malware in one VM sends information can by memory bus side channel to another VM. The attack leverages characteristics of atomic instructions on X86 processors. Once a processor issues an atomic instruction (e.g., the ones with "*lock*" prefix), the memory bus will be locked if the operation address is not aligned or cross page boundary. Once the bus is locked, all the memory access requests from other CPUs must wait until the memory bus is unlocked, which will introduce long latency. In this way, a malicious sender (running in VM-0 on CPU-0) can leverage atomic memory access instructions to lock the memory bus. Thus a receiver (running in VM-1 on CPU-1) can get information by measuring the latency of memory access: long latency for bit-1 and short latency for bit-0. There are other methods of constructing side channels, such as using cache contention.

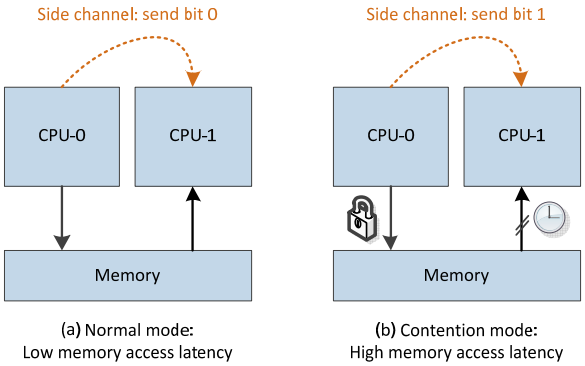

Figure 1. Memory bus side channel attack.

© 2014 ACADEMY PUBLISHER
doi:10.4304/jcp.9.4.1005-1013

Side channel attack is common among different scenarios and is hard to defend. Any sharing computing resources, such as cache, memory, CPU, OS, file system, sound, etc., could be used as a side channel.

In this paper, we have the following observation: one necessary requirement of such attack is that the sender and receiver must run concurrently so that the receiver could get the diversity of memory access latency caused by the sender. Thus we propose a novel mechanism to limit the bandwidth of side channel, by making the VM scheduler aware of such attack. The scheduler controls the overlapping of different VMs execution time under a threshold to reduce the chances of communication between the sender and receiver. Meanwhile, we also extend the hypervisor to periodically call obfuscation functions, in order to inject "noise" to the potential side channel and thus increase the error rate of transmission.

We've implemented a scheduler based on Xen and its credit scheduler. The new scheduler introduces two parameters: *overlap_cap*, which is the threshold of overlap time of execution between every two VMs; and injected *noise_function* for different types of side channel. Both parameters can be set by the user as a scheduling parameter, according to different cases and to achieve a balance between security and performance. The security and performance evaluations show that the new scheduler is both efficient and effective and that the new scheduler is practical and deployable.

The remainder of this paper is structured as follows. Section II describes our threat model of cloud platform. Section III details our design of scheduler, followed by the implementation description. Section V presents our evaluation of the new scheduler to show its effectiveness and efficiency. Section VI surveys related work on side channels and corresponding defending mechanisms. Finally, section VII concludes this paper.

II. Side Channel Attack

In this section, we describe the threat model of cloud platform, and then introduce different ways of constructing side channel attack on cloud environment, followed by our goals.

A. Threat Model

In a cloud computing platform, tenant has several VMs running on host machines of cloud provider. The virtualization software/hardware stack is controlled by the cloud provider, which is considered to be trusted. However, an attacker may also deploy malicious VMs and manage to move such VMs to the same host of victim VMs, and thus share the physical computing resource. Meanwhile, an attack is also able to run a piece of software inside victim VM, which can access critical data inside the VM such as credit card number and password of bank account, but cannot transfer the data out due to some access control policies, either enforced by OS inside the VM itself or mechanisms provided by the VMM, such as decentralized information flow control.

B. Contention based Timing Channel Attack

Timing channel is a common kind of side channel. The main idea is to use time deviation to transmit information. One way is based on leveraging memory-bus contention, and using it as an example to show how side channel is established. The algorithm is similar with the one in [11]. The data is transferred through frames. The sender and receiver both use memory access latency to detect whether each other is running currently, otherwise the frame transmission is aborted and need to retry.

On modern x86 processors (before Intel Nehalem and AMD K8/K10), the implementation of atomic memory operations has been significantly improved so that as long as a memory region could be cached, it is the cache line that be locked instead of the memory bus. However, on these platforms, when the memory region is across an unaligned address that span two cache lines, the atomic instruction still needs to lock the entire memory; otherwise the atomicity could not be ensured. Such locking could also be leveraged to construct the side channel. The latest generations of processor, including Intel Nehalem and AMD K8/K10 have further addressed such problem. Instead of using an entire memory region for all of the processors, the memory is partitioned into different regions. Each processor can access its local memory directly, and leverages an inter-processor communication link to access other processor's local memory. Thus, even if the local memory of one processor is locked, it will not affect other processor's memory access. In this case, an attack could carefully leverage a memory region that not only across the cache line but also across the memory region, which still makes the memory locked while accessing. Thus, the memory contention based side channel exists pervasively on different platforms and is general to be used to construct side channels.

Cache contention based side channel is another kind of attack. It is similar with the memory contention based side channel, except that it uses cache contention as an indication to transfer data. Since cache is not shared among different chips, such attacks cannot success if the sender and receiver are not on the same chip.

C. Our Goals

As long as the attacker's VM and victim's VM are running concurrently on the same host machine, there could be various ways for the two to transfer data through side channels. We are not trying to defend against all kinds of side channel attack in this paper. Instead, in this paper, we will focus on a contention based timing channel attack, e.g., memory-bus contention based side channel, which is more general and efficient in the cloud environment.

© 2014 ACADEMY PUBLISHER

TABLE 1.
MEMORY-BUS CONTENTION BASED SIDE CHANNEL ATTACK

Algorithm 1: Memory-bus Contention based Side Channel Attack [11]

$Base_0$ and *$Base_1$* are threshold of access time, which are used to tell whether the receiver is running
A frame contains a header, data payload and a footer

Sending data frame:	**Receiving data in a frame**:
Result := *SendBits(FrameHead)*;	*Result* := *RecvBits(Data)*;
if *Result is* **not** *Aborted* **then**	**if** *Result* is *Aborted* **then**
Result := *SendBits(Data)*;	**return** *Retry;*
if *Result* is **not** *Abort* **then**	**end if**
SendBits(FrameFoot);	*Result* := Match frame footer;
return *Succeed*;	**if** *Result is* **not** *Matched* **then**
end if	**return** *Erased*;
end if	**else**
return *Retry;*	**return** *Succeed*;
	end if
Sending a block of bits:	**Receiving a block of bits**:
for *each Bit* in *Block* **do**	**for** each *Bit* in *block* **do**
if *Bit = 1* **then**	**for** an amount of time **do**
for an amount of time **do**	Atomic operation on memory
Atomic operation on memory	**end for**
end for	**if** *Mean(AccessTime) > Threshold* **then**
Latency := *Mean(AccessTime)-$Base_1$*;	*Bit* := 1;
else	**else**
for an amount of time **do**	*Bit* := 0;
uncached memory access;	**end if**
end for	**if** too many consecutive 0 or 1 bits **then**
Latency := *Mean(AccessTime) – $Base_0$*;	{Sender not running}
end if	Sleep;
if *Latency* < *Threshold* **then**	**return** *Aborted*;
{Receiver not running}	**end if**
return *Abort*;	**end for**
end if	**return** *Succeed*;
end for	
return *Succeed*;	

III. DESIGN

In this section, we first analysis the critical factor of timing channel attack: the execution overlapping. Then we propose a new scheduler to control the overlap of different VMs and support noise injection.

A. Execution Overlapping Analysis

The length of overlapping of VM executions per time unit is a critical metric that indicates the capability of potential side channel. The longer such overlapping per time unit, the larger bandwidth of side channel could be. Thus, one of the primary goals of this paper is to limit the overlapping time.

Since the sender and receiver run in different VMs, the data transmission between the two is not synchronized. A typical solution is to adopt network coding scheme, such as differential Manchester encoding, to transfer data bits. Considering that the side channel is also error-prone, the sender needs to use checksum and error correct coding mechanism to detect and recover those flipped bits, such as forward error correction (FEC) like Reed-Solomon coding. These encoding schemes generate fixed-length output. For example, the (N_{out}, N_{in}) Reed-Solomon coding will generates N_{out} symbols from the input of N_{in} symbols, where each symbol has 8 bits. Meanwhile, the differential Manchester encoding doubles the bits. Assume that for each bit, the sender keeps the memory bus in contention mode for *Tc*, the transfer time for each frame is:

$$T_{frame} = T_c \times N_{frame} \quad (1)$$

Since frame is the basic unit of data transmission, if the execution overlapping time between the sender and the receiver is less than T_{frame}, the transmission cannot finish and must be redone. Thus, if every time the frame transferring is aborted, then the sender must retry the first frame forever.

As shown in Figure 2, if two VMs are assigned 4 VCPUs on a quad-core machine, the overlap of execution time could be large enough for a sender and receiver to transfer data using side channel. This is because the hypervisor doesn't take overlap of execution time into account, which is the key idea of our work.

© 2014 ACADEMY PUBLISHER

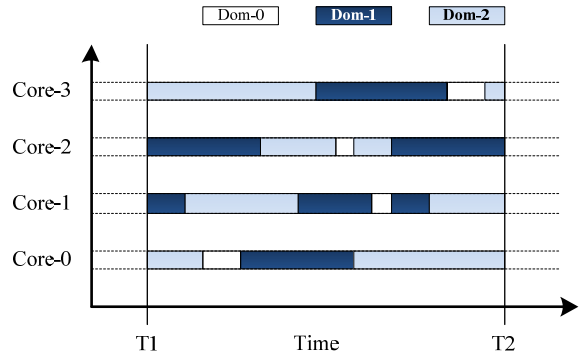

Figure 2. Overlap of execution time of two VMs on a quad-core machine. Both VM-1 and VM-2 have 4 VCPUs and are running CPU-intensive tasks.

B. Scheduling Requirements

In order to defend against the side channel attack, we propose a new scheduler to minimize the overlapping of execution time for different VMs. Beside security mentioned above, there are also several requirements for VM scheduling:

- **Requirement-1**: *Fairness*. Since the hypervisor has no information on which VM is malicious, it assumes that all of the VMs are normal and deserve fairness of scheduling. Thus the proportion of CPU utilization for each VM is determined by the scheduling parameters.
- **Requirement-2**: *Utilization.* The best way to isolate the execution of a specific VM is to stop other VMs when the VM is running. However, it is not practical for cloud providers; otherwise they cannot leverage the low cost benefit of brought by resource sharing. Therefore, a scheduler should keep the utilization as high as possible.
- **Requirement-3**: *Performance.* Another meaning to reduce the bandwidth of side channel is to minimize the length of time slide to increase the error rate of bit-transfer through the channel. However, too frequent VM switching hurt system performance, since more time is used for switching, and data locality decreasing also makes performance penalty such as higher cache miss rate, etc.

C. Overlapping Control

We've proposed two policies suitable for different scenarios.

- **Policy-1**: Minimize overlapping of execution time of different VMs.
- **Policy-2**: Zero-overlapping for specific VMs.

The first scheme aims to achieve no overlapping of execution time for specific VMs. As long as one of such VMs is running, all other VMs, except the domain-0, are not allowed to be scheduled. For example, assume that a host has 16 CPU cores, and a protected VM has 16 VCPUs (Virtual CPU). Once the VM is running, all of its VCPUs are scheduled to run and all VCPUs from other VMs are waiting, until all of the VCPUs of the protected VM are scheduled out.

This scheme is simple and is security by ensuring no overlapping at all. However, it decreases the utilization of computing resources in following aspects. First, all of the VCPUs of the protected VM must run simultaneously. Even if one of the VCPUs is idle, it could not be scheduled out for other VMs. Second, if the number of VCPUs is less than the number of physical CPUs, there will always be some CPUs idle.

The second scheme aims to minimize the overlapping of the execution time, so that each overlapping is shorter than the time needed to transmit one frame of bits. Thus the receiver could not get any useful information. In this way, the computing resource could be leveraged in a more efficient way.

Assume there are n VMs (VM_1 to VM_n) running on a host with m CPU cores (CPU_1 to CPU_m). VM_i has been assigned p_i VCPUs. For each consecutive execution period of each running $VCPU_{ij}$ of VM_i, the start time is S_{ij} and the expected end time is E_{ij}, which means the corresponding time slice P_{ij} is (E_{ij} - S_{ij}). The time of transmitting a frame is T_{frame}, and the time of a regular execution slice is T_{slice}. Thus, once the scheduler puts a VCPU to run, the corresponding time slice should be:

$$P_{ij} = \begin{cases} T_{slice}, if\ T_{now} + T_{frame} \geq MAX(E_{mj,m\neq i}) \\ T_{frame}, if\ T_{now} + T_{frame} < MAX(E_{mj,m\neq i}) \end{cases} \quad (2)$$

The value of T_{frame} could be considered as a parameter of the scheduler, which could be decided by the user for different scenarios. The smaller the value is, the more secure the system could achieve, but the worse performance. This is because smaller time slice will introduce more VM switches, which not only takes more time themselves, but also may violate the cache locality of applications. In order to solve the problem, we design a mechanism, named noise injection, as described in the following section.

D. Noise Injection

Controlling the overlapping of execution time could achieve good security properties. However, it also introduces performance problem since it brings more context switches of VMs. In this section, instead of control the *real overlapping* time of VMs, we limit the *logical overlapping* time by introducing noise injection.

A *logical overlapping* means that two VMs running without any interrupt, during which the sender and receiver could send data by side channel without any interference. In order to mitigate such side channel, we inject some noise from the hypervisor so that the overlapping is interrupted for a minimal side effort, which can break the transmission through a side channel.

Figure 3 shows a set of injected noise marked by red blocks. For each VCPU, its time slice is also calculated through the equation (2). However, once the time slice has passed, instead of switched to another VCPU, the hypervisor inserts some noise in order to break the side channel.

© 2014 ACADEMY PUBLISHER

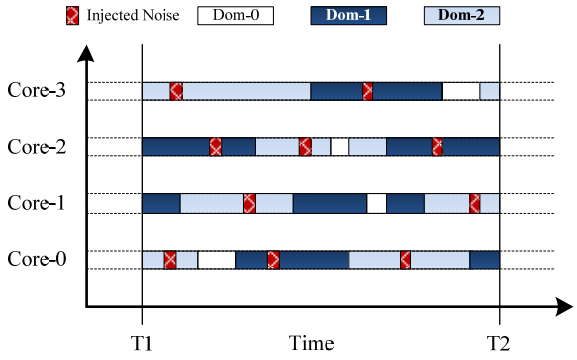

Figure 3. Period noise injection done by the hypervisor.

The noise is specific to the type of side channel. For example, if the administrator aims to defend against memory bus contention based side channel, then he should set the noise injection function as some atomic memory access. Thus the receiver could not tell whether a signal is from the sender or caused by the hypervisor's noise. Meanwhile, if the goal is to defend against cache-contention based side channel attack, then the hypervisor needs to adopt functions that can obfuscate the cache access pattern. Another way is to uses both methods of noise injection to defend the two at the same time. Since the number of functions adopted is related to performance overhead, the proposed scheduler in this paper only offers mechanism and leaves the specific policy to the system administrator who can make a choice after considering the tradeoff between security and performance, and according to different requirements of various scenarios.

It is noted that the number of noise injection can be further reduced, by optimizations that dynamically considers the overlapping of different VMs. For example, assume that one VCPU of VM-1 is running on core-0, and three VCPUs of VM-2 are running on core-1, core-2, and core-3. Instead of injecting noise on every VCPU of VM-2, the hypervisor can only add one noise on core-0. Thus the number of injected noise decreases from 4 to 1.

IV. Implementation

We implemented a prototype based on Xen-4.1.3, which adopts a credit-based scheduler as default. In this section, we first briefly introduce the credit scheduler, and then describe our new scheduler, followed by some optimizations.

A. Xen's Credit Scheduler

Xen's credit scheduler uses credit as time slice for controlling the execution of different VCPUs. The major objective of the Credit scheduler is to fairly allocate CPU resources among different VMs and VCPUs by using credits. It has several parameters, for example, if a VM's credit-weight is 40% of the sum of all running VM's credit-weight, then it can get 40% of the CPU utilization.

There are four running states of each VCPU: BOOST, UNDER, OVER and IDLE, with the priority from highest to lowest. All the VCPUs in the run queue are sorted by their priority. When the *schedule*() function is invoked, the current running VCPU is inserted into the run queue, and the VCPU at the header of the queue is picked up as next VCPU to run. For each VCPU that is scheduled to run, the time slide is about 30ms.

In Credit scheduler, credits are used to track each VCPU's execution time. As long as a VCPU has not burned all of its credit, it is in UNDER state. The speed of credit consumption is 100 per 10ms (10ms is a *tick*). Once a VCPU's credit has dropped under 0, the priority is set to OVER. Every 30ms, the scheduler will recharge credit of VCPUs waiting in the queue. The credit-weight of each VCPU determines the quantity of its charged credit.

B. The New Scheduler

In order to control execution overlapping of different VMs, we add a new parameter to the credit scheduler: *overlap_cap*, as a threshold in terms of millisecond. The scheduler will ensure that for every two VCPUs, the runtime overlapping will not exceed the threshold.

When scheduling a VCPU to run, the scheduler determines the length of its time slice considering three situations: first, if the VCPUs on other cores are from the same VM, the slice is not limited by *overlap_cap*. Second, if some of the running VCPUs are from other VMs, but the left time of running is less than *overlap_cap*, then the time slice is also not constrained. Otherwise, the time slice is set to *overlap_cap* to ensure that the hypervisor gets a chance to run before a large enough overlap happens. If the time slice is reduced, the original one will also be recorded.

By adjusting the value of *overlap_cap*, a user can make a balance between security and performance. If the *overlap_cap* is set to zero, the scheduler will ensure that only the VCPUs from the same VM can run concurrently. In this way, the isolation of runtime is enforced with the price of higher performance overhead. The tradeoff is made by user, according to specific applications and scenarios.

There's another new parameter for the scheduler: the function of noise injection. Every time the hypervisor gets a chance to run, it will call the noise function to defend against potential side channels if the overlap of runtime is approaching the *overlap_cap*. After that, it will extend the time slice of the corresponding VCPU, if the slice is previously reduced. The noise functions can be various, including memory contention or cache contention, or even defined by the user. The more complicated the noise function is, the better security that the system will achieve, but will also bring higher performance overhead.

C. Optimizations

We also made several performance optimizations. First, when selecting a next VCPU to run, the scheduler prefers the one from the same VM in order to get longer time slice and to avoid too frequent context switches. This optimization is especially useful for CPU intensive workloads. As long as the number of VCPUs of a VM is equal or larger than the number of physical, it could achieve good performance, since all of the VCPUs could

© 2014 ACADEMY PUBLISHER

be scheduled to run at the same time. The effect is similar with Gang scheduling [12].

Another optimization is that when a VCPU is in BOOST status, it has the highest priority and will preempt to run even if the VCPUs on other CPUs are from other VMs. Thus the performance of I/O workload is not hurt.

V. Evaluation

In this section, we evaluate the effectiveness and efficiency of our scheduler prototype. We did our evaluation on a machine with Intel quad-core. All of the experiments are done on the same machine. The hardware and software configurations are listed as TABLE 2:

TABLE 2.
HARDWARE/SOFTWARE CONFIGURATIONS

Guest OS	Debian-7 x86-64 with Linux-3.2
Domain-0	Debian-7 x86-64 with Linux-3.2
Hypervisor	Xen-4.1.3
Hardware	Intel i5 2300 with quad-core, with hyper-threading disabled. 16GB memory, 1TB hard disk, 1000 Gbps NIC

TABLE 3.
OVERLAP EXECUTION TIME COMPARISON

	original	mini overlap	0 overlap
average (μs)	12,773	724	0
min (μs)	20	1	0
max (μs)	27,742	983	0

A. Security Evaluation

We measure the security by using memory bus contention based side channel attack. First is to determine the T_{frame}. If the sender uses standard (32,28) Reed-Solomon code, and Tc is 2 μs, then T_{frame} is about 1.0 ms.

In the first experiment, we run two VMs on the same host; each is configured with 4 VCPUs. Each VM runs CPU intensive workload that takes all of the four VCPUs as much as possible. This is to emulate the behavior of colluded sender and receiver, which are attempting to get overlap execution time as much as possible. The credit-weights of both VMs are configured to be the same. We measure the overlap time of the two VMs under default credit scheduler and our side-channel attack aware scheduler which is first configured to use the minimal overlapping policy, and then use the zero overlap policy. The results are shown in TABLE 3.

As the table shows, in the original scheduler, the maximum overlap is about 30ms, which is the default time slice. Once the new scheduler is used, the overlapping of execution time doesn't exceed the T_{frame}, which means the sender could not even send a single frame through a possible side channel. When the zero overlap policy is used, the overlapping is entirely eliminated.

B. Performance Evaluation

Previous section shows that our new scheduler is managed to control overlap of execution between different VMs. In order to measure the performance overhead, we also adopt two VMs on the host, each with 4 VCPUs. This time, the first VM is running SPEC-2006 as benchmark and the second VM still runs CPU intensive workload as before. Then we run dbench in the two VMs to measure how the scheduler could impact the I/O performance. Both experiments are done using three different scheduler configurations: the original one, the one with minimal overlap policy and the one with the same policy plus noise injection. The results are shown as in Figure 4.

As the figure shows, once we use minimal overlap policy of the new scheduler, only 8.7% performance overhead is introduced for CPU intensive benchmark. For I/O intensive workload, the overhead is only 3.2%. This is because for most of the time, the scheduler prefers to run the VCPUs of the same VM concurrently. By using noise injection, the performance of SPEC-2006 drops to 5.4% and for dbench the overhead is only 2.5%. This is because by using noise injection, the system could further reduce unnecessary context switches.

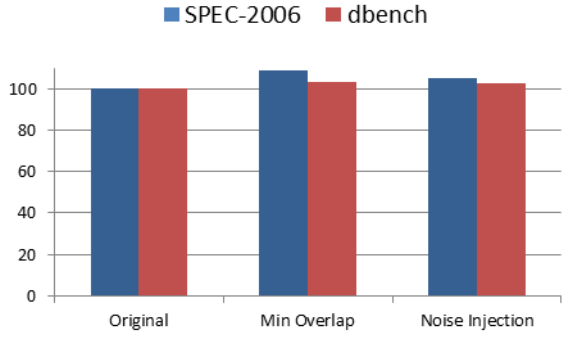

Figure 4. Performance overhead of SPEC-2006, each VM has 4 VCPUs running concurrently.

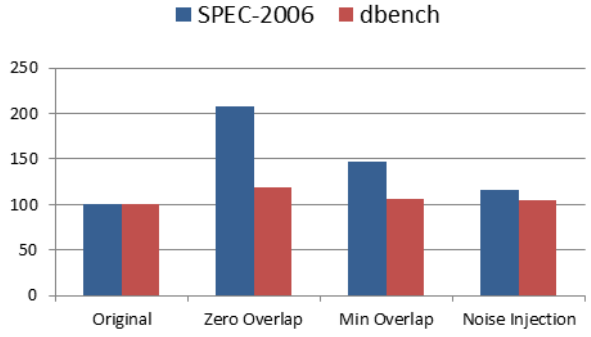

Figure 5. Performance overhead of SPEC-2006, each VM has only one VCPU running.

© 2014 ACADEMY PUBLISHER

We also did an experiment to measure the worst case of performance degrading. Two VMs are adopted, each has only one VCPU. Further, we use both the minimal overlapping and zero overlapping policy. In the original scheduler, since the number of CPU is larger than the number of VCPU, both VCPU is able to run without being scheduled out, which could lead to a large number of runtime overlap. Thus the overlapping control policies will hurt the performance more significantly than the previous case. The results are shown in Figure 5.

As the figure shows, when using zero overlap policy, the performance overhead is as high as 108.3% of the original one. This is because all of the concurrency is eliminated as well as the overlap execution. But for I/O intensive workload, the impact is as low as 18.6%, since the CPU is not the bottleneck, and the BOOST optimization helps to make the I/O VM get higher priority to handle I/O requests. By using minimal overlap, the overhead of CPU intensive workload drops to 47.2%, and the one of I/O workload drops to 6.3%. It shows again that the major overhead comes from the more frequent context switches.

When using noise injection, the overlap is still limited, and the context switch is also reduced. Since instead of switching a VCPU, the hypervisor only executes the obfuscation function and continues to run the interrupted VCPU. In this case, neither the cache nor the TLB is flushed, which keeps the locality. Thus the overhead of noise injection is only 15.7%.

VI. Related Work

Side channel (also as known as covert channel) has been a well-known category of security attack in multiple-user system. Lampson [6] has indicated that the threats of side channel widely exist as long as there are resources sharing in the system, such as processor cache [17][20], virtual memory [23], file system [6], I/O device [4], etc. [2][9].

Among all these types of side channel media, the contention-based timing channel is more trivial to be exploited and more pervasive existing. Meanwhile, the bandwidth of such channel is large enough to transmit not only small object but also large files such as design blueprint. The hardware platform characteristics determine that such attack is hard to be eliminated. Thus, such attacks are attractive to many attackers.

Cache-contention and memory-bus contention based timing channels have been extensively studied. Percival [17] presented a way to construct inter-process bandwidth covert channel with caches that achieves high bandwidth. Wang et al. [20] further improve such channel to virtual machine execution environment and stated that the isolation by hypervisor is not enough, which could lead to cross-VM side channel attack. Ristenpart et al. [3] demonstrated such attack in the Amazon EC2 platform, which uses Xen as virtualization stack. Xu et al. [16] measured the bandwidth of such attack and gave the result that cache covert channels could only limited harm due to its low capacity. Wu et al. [11] further constructed a new way to exploit memory-bus contention based side channel to achieve both larger bandwidth and more general adoption. Since memory is shared across different CPUs, the channel could be constructed without the limitation of being on the same processor. Different from these solutions, our new scheduler doesn't depend on new designed hardware and is a pure software solution that can be deployed on existing hardware and software platform without modification of guest OS.

Previous researchers also presented a lot of hardware solutions in order to detect cache side channel attack, such as RPcache [18], PLcache [15] and Newcache [20]. However, the hardware solutions costs too much and cannot be used in existing systems. There are also many solutions that require software only. HomeAlone [21] detects a malicious VM by faking to be side channel receiver to observe cache timing anomalies. However, the solution requires semantic of normal behavior and could leads to false negative, especially in the cases that only small data objects are transmitted through the channel. Our scheduler uses a different way which mitigates the side channel by reduce the overlap execution time of different VMs, which could achieve better security than previous software solutions.

Besides the efforts from researchers, the industry is also addressing such threat. Amazon EC2 cloud now provides a specific service that allows tenants to use dedicated instances [1] to ensure that the VMs of the tenant run without others' VMs on the same host. This solution could effectively eliminate some side channels which leverage shared hardware to transmit message, including cache side channel and memory side channel. However, the resource of the hardware could not be fully utilized, which makes the user pay higher price to use it. Our scheduler could achieve similar degree of isolation while still keep the high utilization of cloud platform and thus achieves better cost-benefit.

VII. Conclusion

Side channel attack in the cloud platform is now getting more attention. By leveraging virtual machine technology, multiple tenants are able to share computing resource from cloud provider on demand. While sharing can reduce the expenses of computing, it brings security vulnerability as well since the isolation between different VMs could be violated through side-channel attack.

Recent research points out that by leveraging contention based timing channel, two colluded malware within different VMs that sharing the same host may use diversity of memory/cache access latency as a covert channel to deliver confidential information, such as user passwords or credit card numbers, which can bypass access control policies enforced by either the guest OS or the hypervisor. The bandwidth of the covert channel could be up to hundreds of kilobytes, which is fast enough to transfer larger data.

In this paper we propose a covert channel aware scheduler that considers security as first class to mitigate such side-channel attack. We introduced two parameters for the scheduler: *overlap_cap* and *noise_function*. By *overlap_cap*, the user can control the maximum of

© 2014 ACADEMY PUBLISHER

execution runtime of different VMs to mitigate the threat of pontential side channel. The value could be adjusted to achieve a balance between performance and security. By using *noise_function*, the user could defend against specific types of side channel in a more efficient way that further improves the performance by reducing unnecessary context switch. We have built a prototype that implemented both parameter, and can be dynamically configured to adapt to diverse circumstances. The performance evaluation results show that the performance overhead is acceptable.

For the future work, we will further improve the performance of the proposed scheduler. One possible way is to discover more effective and efficient way for noise injection, according to different types of side channel. Meanwhile, we are planning to develop more ways to construct side channel and use them for evaluations to measure the security achieved by our scheduler.

REFERENCES

[1] Amazon Web Services. Amazon EC2 dedicated instances. http://aws.amazon.com/ dedicated-instances/.

[2] Department of Defense. "TCSEC: Trusted computer system evaluation criteria". Technical Report 5200.28-STD, U.S. Department of Defense, 1985.

[3] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. "Hey, you, get off of my cloud: exploring in- formation leakage in third-party compute clouds". In *Proceedings of the 16th ACM conference on Computer and communications security (CCS' 09)*, pages 199–212, 2009.

[4] G. Shah, A. Molina, and M. Blaze. "Keyboards and covert channels". In *Proceedings of the 15th conference on USENIX Security Symposium*, pages 59– 75, 2006.

[5] G. Shah and M. Blaze. "Covert channels through external interference". In *Proceedings of the 3rd USENIX conference on Offensive technologies (WOOT'09)*, pages 1–7, 2009.

[6] B. W. Lampson. "A note on the confinement problem". *Communications of the ACM*, 16:613–615, 1973.

[7] W. Hu. "Reducing timing charmers with fuzzy time". In *Proceedings of the 1991 IEEE Symposium on Security and Privacy (S&P'91)*, pages 8–20, 1991.

[8] J. W. Gray III. "On introducing noise into the bus-contention channel". In *Proceedings of the 1993 IEEE Symposium on Security and Privacy (S&P' 93)*, pages 90–98, 1993.

[9] F. G. G. Meade. A guide to understanding covert channel analysis of trusted systems. Manual NCSC-TG-030, U.S. National Computer Security Center, 1993.

[10] I. S. Reed and G. Solomon. "Polynomial codes over certain finite fields". *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[11] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: high-speed covert channel attacks in the cloud," *USENIX Security*, pp. 9–9, 2012.

[12] D. Feitelson and L. Rudolph. "Gang scheduling performance, benefits for fine-grain synchronization," *Journal of Parallel and Distributed Computing*, 16(4):306–318, 1992.

[13] J. Wu, L. Ding, Y. Wang, and W. Han, "Identification and Evaluation of Sharing Memory Covert Timing Channel in Xen Virtual Machines," presented at the *CLOUD*, 2011, pp. 283–291.

[14] J. Kong, O. Aciicmez, J.-P. Seifert, and H. Zhou. "De-constructing new cache designs for thwarting software cache-based side channel attacks". In Proceedings of the 2nd ACM Work- shop on Computer Security Architectures (2008), pp. 25–34.

[15] J. Kong, O. Aciicmez, J.-P. Seifert, and H. Zhou. "Hardware-software integrated approaches to defend against software cache-based side channel attacks". In *Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture (HPCA'09)*, pages 393–404, 2009.

[16] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. "An exploration of L2 cache covert channels in virtualized environments". In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop (CCSW'11)*, pages 29–40, 2011.

[17] C. Percival. "Cache missing for fun and profit". In *Proceedings of the BSDCan 2005*, 2005.

[18] Z. Wang and R. B. Lee. "Covert and side channels due to processor architecture". In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 473–482, 2006.

[19] Z. Wang and R. B. Lee. "New cache designs for thwarting software cache-based side channel attacks". In Proceedings of the 34th International Symposium on Computer Architecture (2007), pp. 494–505.

[20] Z. Wang and R. B. Lee. "A novel cache architecture with enhanced performance and security". In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture (MI- CRO'41)*, pages 83–93, 2008.

[21] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. "HomeAlone: Co-residency detection in the cloud via side-channel analysis". In *Proceedings of the 2011 IEEE Symposium on Security and Privacy (S&P'11)*, pages 313–328, 2011.

[22] XenSource. Xen credit scheduler. http://wiki. xensource.com/xenwiki/CreditScheduler.

[23] T. V. Vleck. "Timing channels. Poster session", IEEE TCSP conference, 1990.

[24] S. Cabuk, C. E. Brodley, and C. Shields. "IP covert timing channels: design and detection". In Proceedings of the 11th ACM conference on Computer and communications security (CCS' 04), pages 178– 187, 2004.

[25] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning. "Managing security of virtual machine images in a cloud environment". In *Proceedings of the 2009 ACM workshop on Cloud computing security (CCSW'09)*, pages 91–96, 2009.

[26] D. G. Murray, S. H, and M. A. Fetterman. "Satori: Enlightened page sharing". In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC' 09)*, pages 1–14, 2009.

[27] Raj, H., Nathuji, R., Singh, A., and England, P. "Resource management for isolation enhanced cloud services". In Proceedings of the 2009 ACM Cloud Computing Security Workshop (2009), pp. 77–84.

Fei Liu Beijing China, October, 1972.

She is the deputy head of the security research institute of China Mobile Research Institute. She works on security infrastructure, terminals and smart card/business/network security.

Ms. Liu is the deputy head of the security group of China Communication Standards Association (CCSA-TC5). Ms. Liu

© 2014 ACADEMY PUBLISHER

was awarded with the second place of National Institute of Communications Science and Technology Progress Award once, the first place of China Moble and Technological Progress Award once, and the second place of China Moble and Technological Progress Award multiple times.

Langfang Ren Beijing China, July, 1982. Master of Engineering, Beijing Jiaotong University, March, 2007.

She is an imtermediate engineer of the security research institute of China Mobile Research Institute. She works on cloud security and network security area

Hongtao Bai Beijing China, December, 1982. Master of Engineering. Peking University, January, 2008

He is the technical manager, intermediate engineer of the security research institute of China Mobile Research Institute. He works on security protocol, cloud security, mobile security and web security area.

© 2014 ACADEMY PUBLISHER