Time-Stealer: A Stealthy Threat for Virtualization Scheduler and Its Countermeasures

Hong Rong, Ming Xian, Huimei Wang, and Jiangyong Shi

State Key Laboratory of Complex Electromagnetic Environment Effects on Electronics and Information System, National University of Defense Technology, Changsha, China
ronghong01@gmail.com, qwertmingx@tom.com,
{freshcdwhm,fangtuo90}@163.com

Abstract. Third-party Cloud Computing, Amazon's Elastic Compute Cloud (EC2) for instance, provides Infrastructure as a Service (IaaS) solutions that pack multiple customer virtual machines (VMs) onto the same physical server with hardware virtualization technology. Xen is widely used in virtualization which charges VMs by wall clock time rather than resources consumed. Under this model, manipulation of the scheduler vulnerability may allow theft-of-service at the expense of other customers.

Recent research has shown that attacker's VM can consume more CPU time than fair share on Amazon EC2 in that Xen 3.x default Credit Scheduler's resolution was rather coarse. Although considerable changes have been made in Xen 4.x Credit Scheduler to improve the performance in case of such stealing attacks, we've found another alternative attack called Time-Stealer which can obtain up to 96.6% CPU cycles stealthily under some circumstances on XenServer6.0.2 platform by analyzing the source code thoroughly. Detection methods using benchmarks as well as a series of countermeasures are proposed and experimental results have demonstrated the effectiveness of these defense techniques.

Keywords: Cloud Computing, Virtualization, Xen, Credit Scheduler vulnerability.

1 Introduction

Cloud Computing provides high efficiency mainly by multiplexing multiple customer workloads onto a single physical machine to host data and deploy software and services. For instance, Amazon's Elastic Compute Cloud (EC2) [1] offers this sort of service. In other words, Cloud Computing is used to refer to a new business model where heavy computing and software capabilities are outsourced on demand to share third-party infrastructure, not only providing a number of advantages, such as scalability, dynamic provisioning, and low cost maintenance, but also introducing a great deal of new risks [2-4].

In virtualization which is fundamental technology of Cloud Computing, a hypervisor (also called Virtual Machine Monitor, VMM) schedules and manages different

S. Qing et al. (Eds.): ICICS 2013, LNCS 8233, pp. 100–112, 2013.

VMs. Operating system scheduler's vulnerabilities may result in inaccurate or unfair scheduling and such malicious behavior has been proved in the past operating systems—DoS (Denial of Service) attack on BSD4.4 [5] and similar attack more recently on Linux 2.6 [6]. F. Zhou, et al [7-8] have demonstrated that Xen hypervisor is vulnerable to timing-based manipulation and a VM using their attack method can obtain up to 98% (on lab experiments) and 85% (on modified EC2 scheduler) of total CPU cycles despite how many VMs running on the same core. Such attacks typically take advantage of the use of periodic sampling or a low-precision clock to measure CPU usage. An attacking process just needs to conceal itself whenever a scheduling tick occurs.

However, since Xen moves onto latest version Xen 4.x, a lot of improvements have been made to ensure scheduler's fairness, meanwhile experiments show the former attack can't reach its theoretic point. In spite of these changes, we discover another attack form: Time-Stealer which turns out to be useful on some conditions. An attacking VM can acquire 96.6% CPU cycles regardless of the number of VMs on the same pinned core, breaching fairness principle once again.

This paper presents a novel analysis of what variations are made in Xen 4.x scheduler and Time-Stealer's attack scheme while some conditions have to be sufficed to guarantee its success. In terms of its stealth, a detection method without CP support is proposed and a series of countermeasures utilizing Xen 4.x' features are discussed afterwards.

The rest of this paper is organized as follows: Section 2 describes details of Xen 4.x scheduling mechanism and potential vulnerabilities. Section 3 explains Time-Stealer attack mechanism and shows experimental consequences in the lab. Next, Section 4 illustrates our detection scheme and countermeasures. Section 5 presents related work and we draw a conclusion in Section 6.

2 Xen 4.x Scheduling Mechanism and Vulnerabilities

In this section, we start with a brief introduction of Xen hypervisor, and then give an analysis of how does Xen 4.x Credit Scheduler work and what improvement are made compared with previous versions. Finally, we present a novel illustration of its potential vulnerabilities that may be exploited by malicious Cloud customers.

2.1 Xen Hypervisor

Xen is an open-source VMM, or hypervisor, for both 32- and 64-bit processor architectures. It runs as software directly on top of the bare-metal, physical hardware and enables you to run several virtual guest operating systems on the same host computer at the same time. The VMs are executed securely and efficiently with near-native performance [9]. Xen hypervisor is split into two parts: A hypervisor core, which runs directly on the physical hardware and a privileged guest Domain0 (domain means VM in Xen) that provides device drivers. The hypervisor core is responsible for basic

management functions, such as CPU scheduling, interrupt handling and memory management. It runs with ring0 privileges, whereas DomainUs are only allowed to run in the rings 1-3. Therefore, the hypervisor can trap all sensitive processor instructions and remains in full control of the physical machine [10-11].

Many enterprises are making tremendous investments in promoting Xen project, to name a few: Citrix, Amazon AWS, AMD, CA technologies, Cisco, Google, Intel, Oracle and so forth. Motivated by consolidation, reliability and security, a growing demand of server virtualization in industry boosts Xen's step towards Cloud Computing for its opening and excellent performance.

2.2 Inner Workings of Credit Scheduler

As with a multitasking operating system, scheduling in Xen is a tradeoff between achieving fairness for running domains and achieving good overall throughput [9][12]. Xen allows a VM to have one or more VCPUs (virtual CPUs) which are determined to run on which PCPU (Physical CPU) by scheduler. The design and tuning of the scheduler is one of the most important factors in keeping Xen system running well.

On the latest versions of Xen, the Credit Scheduler is used by default [13]. Each domain has two properties associated with it, a weight and a cap. The weight determines the share of the PCPU time that the domain gets, whereas the cap represents the maximum. Weights are relative to each other and decide the initial credits of the VCPU. Each VCPU's credits ensure its runtime compared to other VCPUs. There are five priority states in Credit Scheduler: BOOST (with positive credits, capable of preempting other VCPU if I/O comes to achieve better I/O latency), UNDER (with positive credits, capable of being scheduled), OVER (with negative credits and only can be scheduled in work-conserving mode) and IDLE (represent no VCPU running). The VCPUs marked with Priority States on a PCPU are kept in an ordered queue, with those in BOOST state ahead of those in UNDER state and those in UNDER state ahead of OVER state. Apart from these, BLOCKED state is another VCPU state which means that the current VCPU is not runnable any more, probably awaiting a timing event like I/O response with remaining credits.

The Credit Scheduler uses a fixed-size 30ms quantum. At the end of each quantum, it recalculates all VCPU's credits and priorities and selects a new VCPU to run from a list of those that have not already exceeded their fair allotment. If a PCPU has no UNDER-state VCPUs, it tries to pull some from other PCPUs. The scheduler ticks every 10ms, subtracting 100 credits from the running VCPU and restricting the maximum credits.

Recently, some changes have been made to improve Credit Scheduler performance in Xen 4.1.2, with 195 lines added in source code compared to version 3.1.2. For both, the default weight is 256 and cap is 0. To explain how the new Credit Scheduler behaves, we analyze the source code in detail.

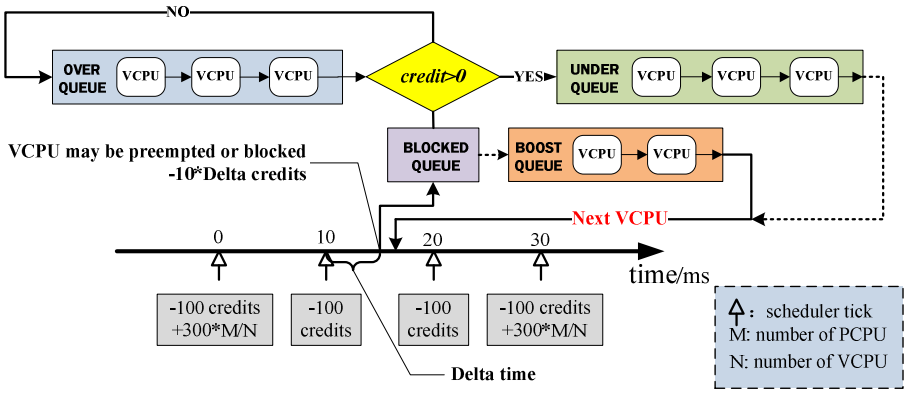

Fig. 1. Xen 4.1.2 Credit Scheduling Procedures

From Fig. 1, the scheduling process can be summarized as follows: a VCPU with adequate credits first goes to UNDER queue waiting to be scheduled; as long as it's on the top of the queue, it can be executed on PCPU, in the meantime its credits will be debited every 10ms; whenever it leaves the PCPU to enter OVER queue or BLOCKED queue, its credits corresponding to the running-time less than 10ms will also be debited. On each 30ms interval, all VCPUs' credits are recalculated and more credits are added. In this way, those in OVER queue can move onto UNDER queue and the whole process will repeat.

The most significant improvement in Xen 4.1.2 Credit Scheduler is smaller granularity in credit calculating. To illustrate further, one of the core scheduler functions called csched_schedule() in */xen/common/sched_credit.c* determines the next runnable VCPU and its duration. A primary advance is that when this function executes, the current VCPU's credits has to be burned regardless of whether clock ticks comes or not, as shown in Fig. 1. This kind of design restricts effectiveness of "Cycle Stealer" [8] which attempts to sleep before scheduler tick arrives to avoid credit deduction and wakes up later with BOOST state to obtain more CPU cycles, because its credits are inevitably deducted by Xen.

2.3 Potential Vulnerabilities

Although Credit Scheduler's resolution has been increased a great deal, as we describe in the following section, this adoption still allows attacker to steal CPU cycles under some conditions and degrade the other VMs performance because of frequent context switches. There are three major vulnerabilities in current Xen version:

(a) Boosting Mechanism

Boosting in Xen Credit Scheduler aims to achieve better I/O latency, simply by prioritizing VCPUs executing I/O work. When a VCPU sleeps waiting for I/O, it will still hold its remaining credits; when it wakes with positive credits, it enters BOOST state and may instantly preempt running VCPU with lower priorities. This mechanism, however, gives attacker's VCPU chances to deprive of other VCPUs' execution by deliberately sleeping first and waking afterwards.

(b) Fixed Scheduling Rate

Each VCPU can run on the PCPU at most 30ms by default, which renders attacker to recognize the exact time when csched_schedule() executes easily. Thus, it is feasible for the attacker's VCPU to evade being scheduled away.

(c) Fixed Credit Deduction

In some cases, attacker's VCPU may quit running just before the debit tick comes, that is, it' s quite unfair that the next VCPU can merely run on the PCPU for less than $\varepsilon(\varepsilon < 10)$ ms while its credits are reduced by 100 compared to those who execute for complete 10ms. This gives an opportunity to breach the justice and launch a Reduction of Quality Attack.

What's more, to achieve the theft-of-service goal like Cycle Stealer, there are two kinds of bewilderments needs to be solved: one is whether VCPU can enter BLOCKED state; the other is whether attacker's VCPU can be aroused apart from I/O events.

Both problems have been perfectly resolved with help of Xen hypervisor. Firstly, when a VM sleeps on the run, an idle process occurs in it. Under of model of paravirtualization, VM's idle process is supposed to be replaced by a hypercall named do_sched_op(), making its VCPU blocked and raising a soft-interrupt-request to Xen. Credit Scheduler then processes the request and reschedules next VCPU. Secondly, instead of hardware timer, VM OS (Operating System) uses a singleshot_timer provided by hypervisor to complete timing events. When VCPU is about to go to sleep for a period, a hypercall named do_set_timer_op() is invoked to set a timer. The moment timer expires, a software interrupt is released to hypervisor. In the end, attacker's VCPU is unblocked. Moreover, it will preempt current VCPU in BOOST priority with positive credits.

3 Time-Stealer Attack

Previous work F. Zhou [8] did has already proven a successful attack that every 10ms the scheduler tick fires and schedules the malicious VM which runs for $10-\varepsilon$ ms and then calls halt() to briefly go idle to ensure another VM will be running at the next tick. Later, the attacker wakes in BOOST priority and preempts the current running VM, namely, it can execute for $10-\varepsilon$ ms out of every 10ms tick debit cycle. On account of improvements in Xen 4.x, the former attack cannot reach up to its threatening impact. In this section, a similar attack scheme is proposed and experiments show Time-Stealer attack is more effective than Cycle Stealer.

3.1 Time-Stealer Attack Description

Time-Stealer attack mainly relies on the fixed scheduling rate used by Xen Credit Scheduler, not only in version 3.1.2, but also in 4.1.2 as well as newest 4.2.1, the default scheduling period is always 30ms. When this schedule tick arrives, no matter how many credits current VCPU may have, it has no choice but to leave the PCPU while next VCPU on the head of UNDER queue is probably scheduled at next time slice.

Since any VCPU credits are inevitably debited, attackers might as well choose to execute on the PCPU for $30-\varepsilon$ ms and then go to sleep to prevent be scheduled away as shown in Fig. 2. After a little while, it just wakes up again to consume CPU cycles. Via this scheme, attackers can acquire more CPU time than fair share on conditions described as follows.

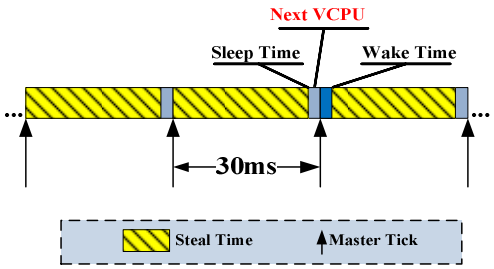

Fig. 2. Time-Stealer scheme

According to scheduling policy, attacker's VCPU can't wake up in BOOST state without positive credits. Here's a critical question: can Time-Stealer still work if its VCPU credits are burned? The answer is affirmative. Through theoretical analysis, the probability of attacker's VCPU running out of credits is relatively small, whereas in contrast, the probability of other VCPUs' being preempted is rather large.

To illustrate, consider a simplified scenario consisting of n identical VMs on a Xen host server. Each VM has only one VCPU and the server has m PCPU cores. Let Ra denote the minimum periods that attack's VCPU runs continually and let Rn denote the maximum scheduling periods that other VCPUs executes. We assume attacker's VCPU runs for 29ms then sleep 1ms in every scheduling period and its initial credits is set to be 300. As has been mentioned earlier, each active VCPU will get $300m/n$ credits every 30ms. Thus, the total credits of attacker's VCPU within Ra are given by:

$$Ca = 300 - 290Ra + (300m/n) \bullet Ra \tag{1}$$

When attacker's credits are negative, it will wake up in OVER state; that is, by $Ca < 0$, we can get Ra:

$$Ra = ceil(\frac{300}{290 - 300m/n}) \tag{2}$$

The function "$ceil(x)$" means the smallest integer which is larger than x.

When attacker's VCPU enters OVER queue, there are at most $n-1$ VCPUs ahead waiting for execution, thus: $Rn = n-1$.

As long as the attacker's VCPU credits return to positive (300 supposed) and stays on the top of UNDER queue, the whole process above will just repeat. Then, the probability of attacker's occupation of PCPU can be defined as:

$$Pa = \frac{Rn}{Rn + Ra} \tag{3}$$

Normally, the fair share probability of every VCPU is supposed to be $1/n$. On the condition that $Pa > 1/n$, attacker's purpose of obtaining more CPU cycles can be achieved. Notice that this sort of attack is closely related to number of PCPUs and VCPUs. When $n \le m$, Ca will remain positive forever. Usually, this is not the case. When $n > m$, for instance, $n = 6, m = 4$, then Pa is 32.3% greater than normal rate 16.7%. Once taking the credits loss of other VCPUs into account, this proportion will be magnified incredibly.

3.2 Implementation and Evaluation

Our implementation of Time-Stealer attack is simple: attacker' VM chooses to run on one PCPU for $30 - \varepsilon$ ms and then invokes a systemcall—usleep() to go into BLOCKED state, hoping to be scheduled away from the PCPU. On the next scheduler tick, another victim VM is charged by 100 credits, and immediately the attacker' VM wakes up in BOOST state and preempts the victim VCPU. TSC hardware register is applied to provide precise time sampling.

To examine the performance of our attack scenario in practice, we evaluate in Citrix XenServer6.0.2 (Free) which is an enterprise-level complete virtualization platform delivering uncompromised performance, scale, and flexibility at no cost [14]. XenServer allows a deployment of virtual x86 computers based on Xen hypervisor. Time-Stealer effectiveness is testified in two ways: one tested is by a simple loop; the other is tested by a CPU benchmark. Experiment configurations are given in Table 1.

Table 1. Experiment Setup

ITEMS	CONIFGURATIONS
Type	Dell PowerEdge T410
CPU	Intel Xeon E5606@2.13GHz
RAM	16GB
Platform	XenServer6.0.2
Kernel	Xen 4.1.2
Guest OS	CentOS 5.8_32bit (paravirtualized)

As our CPU has 4 cores, we intentionally separate our testing instances in order to prevent interferences: 2 cores for test instances and the other 2 for Domain0. Eight identical VMs are setup by Xen "VM copy" function and every VM has only one VCPU. Note that sleep time cannot go beyond ε to make sure waking up timely, meanwhile, ε should be as small as possible to obtain maximum stealing cycles. However, too small ε increases the risks to be ticked by the scheduler. In the overall tests, sleep time is set to be 0.5ms.

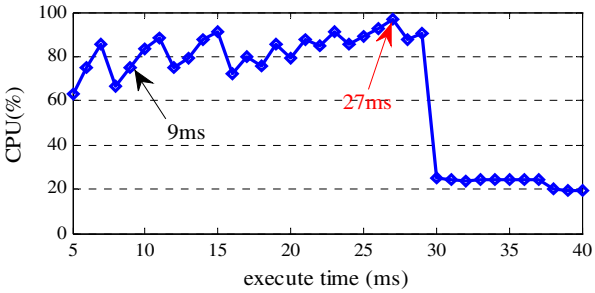

Fig. 3. Attack performance vs. execute time

Our first test concerns about the effect of Time-Stealer execute time on CPU utilization. Fig. 3 shows that there's unstable growth of attacker's VM CPU utilization before 30ms execution while a dramatic drop appears at exact 30ms point, after which the usage remains to be around 20%. This figure demonstrates our attack effectiveness, whereas the curve reaches its top at 27ms instead of 29ms probably because of tick timing jitter. In addition, execution of 9ms proposed in [7-8] cannot reach its theoretical point (98%).

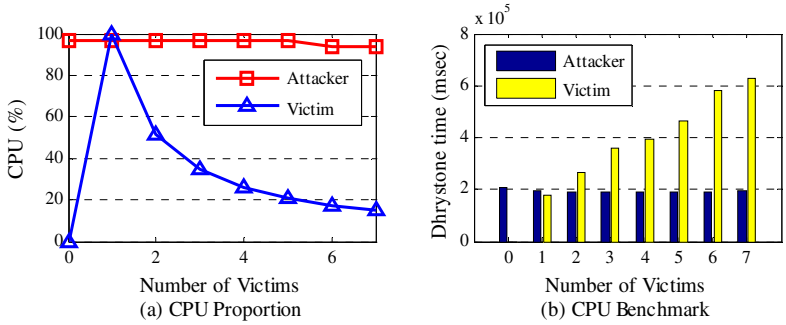

Fig. 4. XenServer experiments- CPU and application performance for attackers and victims

In Fig. 4 (a), we can see attacker and victim CPU utilization on two-pinned-core scenario. As with an increase of victim VM numbers, attacker can obtain up to 96% CPU of a single core all the time while resource contentions of other VMs are getting worse and worse.

Fig. 4 (b) presents consequence of CPU benchmark. In this test, we modified the source code of a famous CPU benchmark—Dhrystone 2.1, by adding Time-Stealer's control part that runtime interval is checked every 9,000 iterations of Dhrystone runs. Via comparing the time the modified VM and unmodified VMs takes to complete 1.0×10^9 runs of Dhrystone operation, the performance of Time-Stealer can be further testified. The touchstone measurement was achieved by a VM using unmodified Dhrystone without CPU usage competition; it finished the total iterations in 177,105ms compared with 207,935ms recorded by mere one VM with modified benchmark. From Fig. 4 (b), it's quite obvious to see that time costs have a steady

growth with growth of victim VMs while attacker's time keeps rather low and stable; in the severest situation; the cost of victim VMs is 3 times higher than attacker's.

4 Time-Stealer Detection and Mitigation Measures

The theft-of-service attack on Credit Scheduler mainly takes advantage of BOOST mechanism and fixed scheduler timing. It's rather easy and feasible to launch a similar attack on Xen 3.x platform proven by earlier work [7-8]. Due to more changes has been made to Xen scheduler, especially, right before attacker's VCPU chooses to sleep, its credits will be subtracted even if credit tick does not arrive yet. In this section, we propose some countermeasures to detect and mitigate Time-Stealer attack.

4.1 Detection Measures

Generally, it's a little bit difficult to detect Time-Stealer attack, since it has no collateral effect except for performance degradation like lowering CPU usage as well as cache polluting because of frequent context switches. If users of victim VMs paid more attention to their CPU utilization and time gap of finishing the same job, theft-of-service evidences would be collected to detect attacker's existence.

For our detection method, we take Dhrystone CPU benchmark as our detection tool. Support from CP is not basically required. Specifically, it involves continual surveillance on benchmark data and judgment on whether a turning point might come. Assume that T_i means the time Dhrystone takes in i th round; T_{ave} shown in (4) is the average time tested many times starting from the very beginning of purchasing service from CP like Amazon EC2; T_{curr} shown in (5) is the average time tested recently ($m \gg n$). Note that the quantity of samples should be large enough to be precise.

$$T_{ave} = (1/m)\sum_{i=1}^{i=m} T_i \tag{4}$$

$$T_{curr} = (1/n)\sum_{i=1}^{i=n} T_i \tag{5}$$

$$|T_{curr} - T_{ave}| < \delta \tag{6}$$

(6) can be used as a judgment formula where $\delta(\delta > 0)$ is the judgment factor set by VM owner's before the test. There might be Time-Stealer attack if the time difference were larger than δ .

In our experiments, we also apply more simple strategy that it's more efficient to detect abnormity by observation of VM CPU usage under CPU-bound workloads like Dhrystone. Generally, CPU-bound workloads consume a large proportion of CPU (82% on average through various tests) while only 2~3% under Time-Stealer attack cases. This can be considered as an alert that attacks like Time-Stealer may exist, but

more accurate judgments and forensics require longer observations using the method we mentioned earlier.

4.2 Mitigation Measures

A. Be Careful to Pin

In the experimental scenario, pinning method is exploited on purpose of excluding the interferences of Domain0. However, we find that results can be various with different CPU affinity. Fig. 4 presents a test VMs sharing 2 cores and Fig. 5 shows CPU usage under 3 different conditions. From Fig. 5 (a), it can be concluded that Time-Stealer attack has no obvious effect under No-CPU-Affinity condition since they experience the same drop as with the growing number of VMs. From Fig. 5 (b), we can see that on 3 cores condition, attacker still occupies more CPU cycles though its plot drops when the number of VM increases. From Fig. 5 (c), the competition seems worse for victim VCPUs than (a) and (b), because the total CPU resources is restricted to one PCPU, however, this does not affect attacker's VCPU much. Obviously, CPU affinity does have some relations with Time-Stealer performance. A primary reason for this is that affinity restricts VCPU floating among the cores, offering a good chance for launching the attack, whereas in no affinity case, each PCPU's queue constantly changes and load can be balanced among all cores.

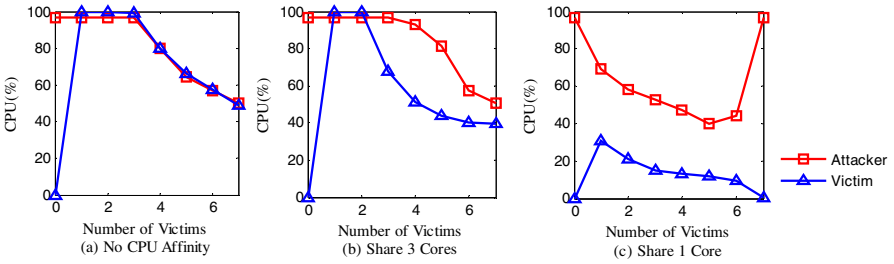

Fig. 5. Attack performance vs. CPU affinity

B. Take Care of Default Parameters

As Xen has made a great deal of improvement in VCPU scheduling, we should make the most of it to secure customer's VM. For example, Xen 4.1 provides a feature called cpupools, which allows users to divide PCPU into distinct groups [15]. Each pool has its entirely separate scheduler, which can protect our CPU cycles from be stolen by attackers from other cpupools.

In Xen 4.2, parameters like Schedule Rate Limiting (ratelimit_us) and Timeslice (tslice_ms) were added for easier control and customization. "ratelimit_us" is used to restrict the schedule rate to ensure the minimum amount of time which a VM is allowed to run without being preempted [15]. The default value is 1000 (that is, 1ms). This kind of parameter helps a lot to constrain Time-Stealer activities which depends on BOOST priority.

"tslice_ms" is Credit Scheduler timing period which is fixed at 30ms for the Xen4.1.2 Credit Scheduler. Remember that our attack scheme relies a lot on this fixed period. If this parameter can be changed, it'll take much longer before attackers find the real Timeslice. However, since its default setup is 30ms, we'd better not to use the default parameter.

5 Related Work

A lot of security analyses have been conducted on the traditional operating system scheduler; meanwhile, proof of concepts of timing attack were not received enough focus until recently. McCanne and Torek [5] showed the timing attack on BSD4.4 and created a uniform randomized sampling clock to estimate CPU utilization. Effectiveness of a similar attack on the Linux 2.6 scheduler has been proven by Tsafrir [6], which brought about receiving higher priority without consumption of CPU.

As I/O performance comes as a bottleneck for most hypervisors, researchers dedicate great efforts on improving I/O [16-20]. Most of them handle the problems like long-term fairness between various VMs such as CPU-bound, I/O bound, but malicious VM's activities are not taken into consideration. Recently, hypervisor security is attached a great importance [21]. Hosting many VMs onto one physical server brings various security threats to Cloud Computing, such as performance isolation violation [22-29], scheduler timing attacks [7-8], and side-channel attacks [30-31]. Several projects have demonstrated side-channel attacks through the shared LLC that can be used to extract information from co-resident VMs.

6 Conclusion

Scheduling processes in hypervisors like Xen, VMware ESXi, KVM play a significant role in multiplexing VMs as well as ensuring the fair share of CPU resources, which should be considered carefully in commercial services like Cloud Computing. Owing to vulnerabilities of schedulers, ordinary customers couldn't get the service they paid for if CPU cycles were stolen by a co-resident malicious VM.

We have demonstrated that this vulnerability still exists on the platform of XenServer6.0.2. Under our test scenario, a malicious VM using Time-Stealer scheme obtained up to 96.6% cycles of a PCPU regardless of competitions from other VMs and effects were more obvious with modified Dhrystone CPU benchmark. In addition, a feasible method to detect Time-Stealer existence without support of CP is proposed. Finally, we further illustrate that this kind of attack can be mitigated if proper configurations are made like changing the default parameters.

References

1. Amazon Elastic Compute Cloud, EC2 (2013), `http://aws.amazon.com/ec2/`
2. Vaughan-Nichols, S.J.: Virtualization Sparks Security Concerns. IEEE Computer Society 41, 13–15 (2008)

3. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Computer Clouds. In: ACM CCS, pp. 199–212 (2009)
4. Tanzim Khorshed, M., Shawkat Ali, A.B.M., et al.: A Survey on Gaps, Threat Remediation Challenges and Some Thoughts for Proactive Attack Detection in Cloud Computing. In: Future Generation Computer System, vol. 28, pp. 833–851 (2012)
5. McCanne, S., Torek, C.: A Randomized Sampling Clock for Cpu Utilization Estimation and Code Profiling. In: USENIX, pp. 387–394 (1993)
6. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Secretly Monopolizing the CPU without Superuser Privileges. In: The 16th USENIX Security Symposium, pp. 239–256 (2007)
7. Zhou, F., Goel, M., Desnoyers, P.: Scheduler Vulnerabilities and Coordinated Attacks in Cloud Computing. In: IEEE International Symposium on Network Computing and Applications, pp. 123–130 (2011)
8. Zhou, F., Goel, M., Desnoyers, P.: Scheduler Vulnerabilities and Attacks in Cloud Computing. In: Distributed, Parallel, and Cluster Computing, pp. 1–23 (2011)
9. Williams, D.E., Garcia, J.: Virtualization with Xen, pp. 43–91. Syngress Publishing (2007)
10. Barham, P., Dragovic, B., Fraser, K., et al.: Xen and the Art of Virtualization. In: ACM SOSP, pp. 164–177 (2003)
11. Jaeger, D., Krentz, K.-F., Richly, M.: Xen Episode IV: The Guests still Strike Back. In: Cloud Computing Security Summer Term, pp. 1–15 (2011)
12. Chisnall, D.: The Definitive Guide to the Xen Hypervisor, pp. 217–223. Prentice Hall PTR (2007)
13. Cherkasova, L., Gupta, D., Vahdat, A.: Comparison of the Three CPU Schedulers in Xen. SIGMETERICS Performance Evaluation Reviews, 42–51 (2007)
14. Citix, Inc.: Citrix XenServer 6.0 Administrator's Guide. 1.1 Edition (2012)
15. Credit Scheduler (2013), `http://wiki.xensource.com`
16. Kim, H., Lim, H., Jeong, J., Jo, H., et al.: Task-aware Virtual Machine Scheduling for I/O Performance. In: ACM VEE, pp. 101–110 (2009)
17. Govindan, S., Nath, A., Das, A., Urgaonkar, B., Sivasubramaniam, A.: Xen and Co.: Communication-aware Cpu Scheduling for Consolidated Xen-based Hosting Platforms. In: ACM VEE, pp. 126–136 (2007)
18. Ongaro, D., Cox, A.L., Rixner, S.: Scheduling I/O in a Virtual Machine Monitor. In: ACM VEE, pp. 1–10 (2008)
19. Weng, C., Wang, Z., Li, M., et al.: The Hybrid Scheduling Framework for Virtual Machine Systems. In: ACM VEE, pp. 111–120 (2009)
20. Gulati, A., Merchant, A., Varma, P.J.: Mclock: Handling Throughput Variability for Hypervisor IO Scheduling. In: OSDI, pp. 1–7. USENIX, CA (2010)
21. Luo, S., Lin, Z., Chen, X., et al.: Virtualization Security for Cloud Computing Service. In: International Conference on CSC, pp. 174–179. CSC, Hong Kong (2011)
22. Bhadauria, M., McKee, S.A.: An Approach to Resource-aware Co-scheduling for CMPs. In: ICS, pp.189-199. ACM (2010)
23. Merkel, A., Stoess, J., Bellosa, F.: Resource-conscious Scheduling for Efficiency on Multicore Processors. In: EuroSys, pp. 153–166. ACM (2010)
24. Zhuravlev, S., Blagodurov, S., Fedorova, A.: Addressing Shared Resource Contention in Multicore Processors via Scheduling. In: ASPLOS, pp. 129–142. ACM (2010)
25. Raj, H., Nathuji, R., Singh, A., England, P.: Resource Management for Isolation Enhanced Cloud Services. In: CCSW, pp. 77–84. ACM, Chicago (2009)
26. Shieh, A., Kandula, S., Greenberg, A., Kim, C.: Seawall: Performance Isolation for Cloud Datacenter Networks. In: HotCloud, p. 1. USENIX (2010)

27. Verghese, B., Gupta, A., Rosenbum, M.: Performance Isolation: Sharing and Isolation in Share-memory Multiprocessors. In: ASPLOS, pp. 181–192. ACM (1998)
28. Cardenas, C., Boppana, R.V.: Detection and Mitigation of Performance Attacks in Multi-tenant Cloud Computing. In: ICACON (2012)
29. Varadarajan, V., Kooburat, T., et al.: Resource-Freeing Attacks: Improve Your Cloud Performance (at Your Neighber's Expense). In: ACM CCS, pp. 281–292 (2012)
30. Xu, Y.J., Bailey, M., Jahanjan, F., Joshi, K., Hiltunen, M., Schlichting, R.: An Exploration of L2 Cache Covert Channels in Virtualized Environments. In: CCSW, pp. 29–40. ACM, Chicago (2011)
31. Zhang, Y., Juels, A., Oprea, A., Reiter, M.K.: Homealone: Co-residency Detection in the Cloud via Side-channel Analysis. In: Security and Privacy IEEE Symposium, Berkeley, CA, pp. 313–328 (2011)