ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 800-807
doi: 10.18178/wcse.2017.06.140

An Anti-Side-Channel Virtual CPU Scheduling Algorithm Based on Leakage Evaluation for Virtual Machine Security

Yuanzhi Du [+], Xuehui Du and Zhi Yang

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China

Abstract. *Infrastructure as a service* (IaaS) which is one of the cloud computing's service modes provides virtual machines to clients via shared physical machines. This Service provides convenience for many enterprises, but also introduces new security threats. Many studies have shown that the co-residency side-channel can be used to extract sensitive information by malicious users. Remarkably, Soo-Jin has present a migration-based system called Nomad to mitigating known and future side-channel which is more universal than the traditional method. However, large scale migration will lead to huge network overheads. To solve the above problems, the characteristics of co-residency side-channel on the single physical server is analyzed, based on which two virtual machine schedule algorithms according the leakage model in Nomad were proposed. The simulation results show that the algorithm can mitigate the threats effectively.

Keywords: Side-channel, Scheduler, IaaS, Xen.

1. Introduction

Cloud computing is providing cheap and convenience services to many organizations and users, however, it introduces new threat either. For example, cloud computing offer cheap services to users by resource sharing, although resources sharing improves the utilization, malicious users are using shared resources to destroy the security of cloud computing.

IaaS is one of the cloud computing's service models. In the IaaS Service, the user's virtual machine usually running in a physical server which it is assigned to. Once virtual machines are running in the same physical server we called the virtual machines are co-residency. Co-resident virtual machine will reuse the hardware by time-sharing which introduce the leakage channel, which the researcher called cross-vm side channel attacks.

Jin-Soo[1] proposed Nomad is different from conventional methods for a particular side-channel attacks. Soo-Jin believes that the current anti-side-channel attack technology is lack of universality. Before the system fixes vulnerabilities, the threat will be exist a long time. To this end, Soo-Jin in the assumption that the attacker and the attack method is unknown, proposed an anti-side-channel system Nomad based on the virtual machine migration. The basic idea of the method is migrating some virtual machines to other physical servers during a given time interval, reducing the connection between virtual machines. Because as co-residence time gets longer, the possibility of information disclosure and leakage are increased.Generally, attackers and attacks are often unknown, so the method proposed by Nomad in CCS'15 is close to reality and can play a role in the cloud computing environment. However, it is conceivable that the method will also lead to a huge network overhead. Because every few minutes Nomad need virtual machines' migration. In view of this problem, we reconstruct the leakage model in a physical server: an attacker wants to perform a side-channel attack through a virtual machine. The virtual machines must be run on a physical server and use

[+] Corresponding author. Tel.: +86 18906083268.
E-mail address 1776253483@qq.com.

some shared resources. By reducing the frequency of virtual machines bi-using these shared resources to achieve the goal of mitigating side-channel attack.

2. Background and Relate Work

2.1. Side-channel

As literature[2] defined, side-channel is the use of a variety of physical state information to recover sensitive information. And most side channels in cloud computing are based on access. These channels had grown from single cross-process side-channel in single machine to cross-virtual machine side-channel in IaaS cloud.

ZJRR[3] is the first cross-virtual machine attack which can extract ElGamal key by L1 cache. After that, researchers try to extract information from last-level cache[2, 4]. However, the information that the attacker extracts is very little[5]. Until Yaron[4] proposed Flush + Reload technology based on the literature[6]. In addition, it is worth mentioning that Irazoqui[7] proposed Invalidate + Transfer technology is first realizing cross-processor side-channel.

After the realization of side-channel attacks, researchers often prevent attacks from stopping the necessary conditions to achieve side-channel. According to their implementation, we can classify these channels as follow:

First, the defend bases on hard isolation. Hard isolation prevents leakage from avoiding attackers and victims' hardware sharing[8-11]. For example, partition memory, or avoid content being shared by software. These techniques need specific hardware support and sacrifice the advantages of cloud computing resource sharing, and therefore does not apply to the cloud computing environment. In addition, researchers have attempted to assign virtual machines to different cores to avoid sharing a single cores and cores' components[12-14], or assign it to a different server, avoid sharing any hardware[5]. However, cloud providers can not identify attackers and victims, taking into account the cross-processor side-channel attacks, so only the clients can use physical servers exclusively can achieve the goal of security which is undoubtedly greatly weakened the hardware sharing, increasing the burden on cloud providers. There is another way to achieve hard isolation. That is, when the system switches the virtual machine, reset all of the hardware status, such as the cache. However, studies have shown that resetting cache will slow the process of the execution speed more than 75-fold [15].

Except hard isolation, the researchers try to increase the channel noise by modifying the hardware. By removing or modifying the clock[16-18] can prevent an attacker from distinguishing events on microarchitecture, these attacks often extract information according the difference between cache hit and miss. Literature[19, 20] use special hardware to confuse the processor cache operation. Literature[8] allocate exclusive memory resources to sensitive process.

Last one is the soft isolation. Soft isolation method is a new research hotspot, its defense principle is to reduce the co-residence by scheduling. As Nomad reduce co-residence time by migration; Varadarajan[21] decreases the Xen's MRT to mitigate preempt frequency, but there is too much restrictive and can't cope with multi-core environment which is the trend of cloud computing.

2.2. Virtual Machine Scheduling

The essence of virtual machine scheduling is virtual CPU (VCPU) scheduling, once a VM runs out its time slice, scheduler has to choose physical CPU (PCPU) for next VM. In fact, the VCPU scheduling is similar to the process scheduling in operating system whose principle is to ensure efficiency while taking into account the fair. The difference lies mainly in the virtual machine is relatively static which means the running time is generally longer than the processes, while processes are created and closed often.

With the development of Xen virtualization technology, the VCPU scheduling algorithm is evolving, too. In order to make the VCPU scheduling algorithm be compatible, Xen applied the idea of separation of strategy and mechanism. So developers can design the algorithm meeting their own needs, which is the basis for the realization of this article.

The current research on scheduling algorithm is how to improve the efficiency of physical machine by balancing latency-sensitive virtual machine and the CPU-hungry virtual machine.

3. Problem Overview

3.1. Threat Model

Firstly, it is assumed that the adversary attack target is to obtain the target virtual machine's sensitive information. And the adversary has the following capabilities:

- Unknown adversary: The system doesn't know which virtual machine belongs to the adversary. Because all the virtual machine on the system has equal right to control shared resources, all the virtual machine is likely to establish a side-channel.
- Target identification: The adversary can identify the target virtual machine including the physical machine where the virtual machine is located, so we can't prevent the co-residency.
- Unknown side-channel: The way the adversary implements the side-channel is unknown, and all the shared resources are likely to become side-channels used by the adversary, such as CPU, cache, memory, and so on. Therefore, it is impossible to turn off the covert channel by fixing the hardware and software bugs. The adversary virtual machine begin to extract the sensitive information at the time target virtual machine and adversary machine are co-residence.
- Efficient information collation: Though the process of obtaining information through the side-channel may be intermittent, we maximize the information leakage caused by the side-channel by assuming the leakage information that adversary extracting in the intermittent procedure can be continuously accumulative.

To analysis the problem more clearly, We take some restrictions to the model. First of all, adversary can not control the virtual machine monitor (VMM). VMM is safe and reliable in this model. Because VMM is the basis of our security mechanism which is responsible for the allocation of physical resources. Then, each virtual machine uses only one VCPU in this paper. Although there are more and more the virtual machine uses multiple VCPUs, we think that this situation is similar to the situation where the user has a replication or the adversary can collaborate.

3.2. Disclosure Factor

Figure 1(a) shows virtual machine side-channel information leakage process under the model assumption: Only two virtual machine can running in this node at the same time, and after hypervisor allocate the VCPU to PCPU, the VCPU will queue in a PCPU queue and start running according to this queue in turn. Table<NR,NC> means the leakage between virtual machines. After the research of side-channel attacks, we identify four disclosure factor make a difference to the leakage. There are four table in figure 1(b): Table <NR,NC> means the virtual machine has no replication running in the same node and there is no collaboration between virtual machines; Table <NR,C> means the virtual machine has no replication running in the same node but there is collaboration between adversary virtual machines; Table <R,NC> means the virtual machine has replication running in the same node but there is no collaboration between virtual machines; Table <NR,NC> means the virtual machine has replication running in the same node and there is collaboration between adversary virtual machines.

We identify four key factor that affect information leakage that we discuss below.

Pair times. We define pair times means the frequency of two VMs running in the same PCPU. Because the mainstream of cross-VM side-channel leads from two VM running in the same PCPU, by the way that the latter VM extracts information from the former VM. We don't considerate the noise in the channel means that if there are VMs between two VMs who use an channel, the information would not be influenced. We assume that every time the sender VM can make full use of its time slice and send K bits information to the recipient VM. In figure 1(b), VM0 can send K bits information to VM2 no matter there are VM1 and VM2 running between them.

Across time. As researchers have developed cross-core and cross-processor side-channel construction methods, so isolation achieved by CPU is becoming the past. When VMs are running in the same server but

in different CPU, it's possible that there exist a side-channel. We assume that two virtual machines run co-run a time slice can transfer L-bits information.

Number of sender replication. It's obviously that if there are more replications in the same server, the adversary can get the information more easily. In figure 1(b), table <NR,C> is the situation that VM2 is collaborate with VM3.

Recipient's collaboration. As we assumed in section 3.1, adversary can find the server which target VM running, so the adversary can deploy more VM on the same server to get the information. In figure 1(b), table <R,NC> is the situation that VM1 is VM0's replication.

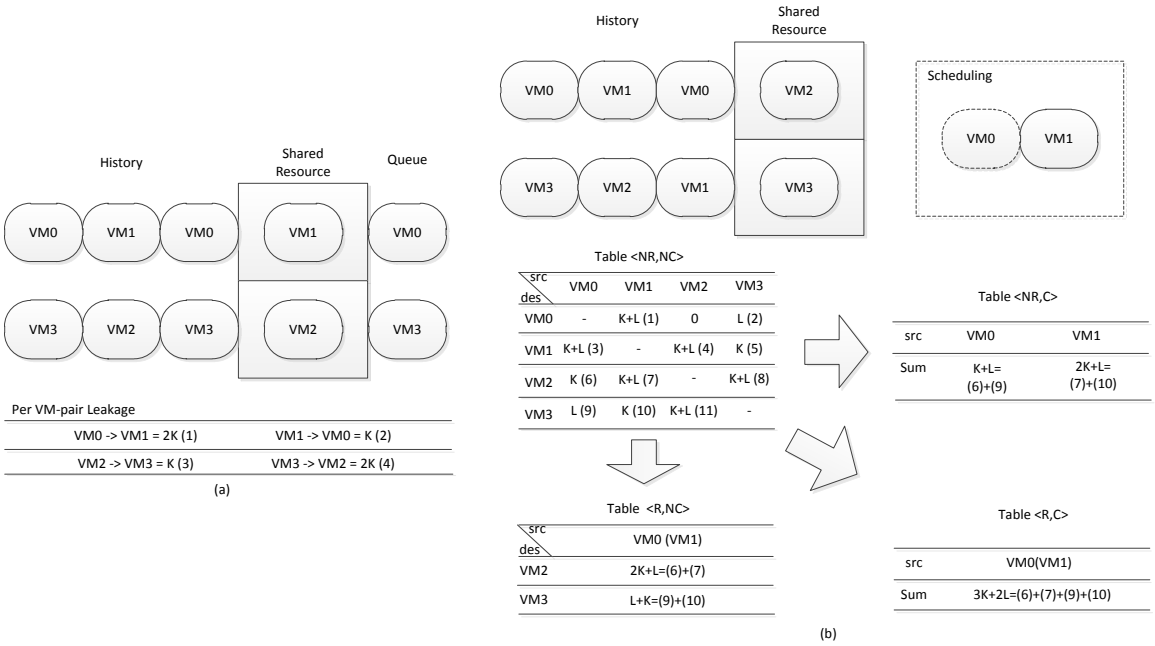

Fig. 1: Cross-VM Leakage Process: (a) traditional scheduler (b) anti-side-channel scheduler

3.3. Formalizing Leakage Model

In order to analysing the above process more clearly, we formalize the above process. First, let VM represent the virtual machine set running on the physical machine. The copy of the sender virtual machine i and its replications are represented by I. The consonant virtual machine of the receiver virtual machine j and machine j are denoted by J. T means the time that system is running. $PairTime_{i,j}(t)$ means the frequency of (i,j)-pair from system start to time t. $CoTime_{i,j}(t)$ means the time VM i and VM j co-run from system start to time t. So the amount of leakage information as follow formula:

$$Leakage_{i\to j}(T) = PairTime_{(i,j)}(T)\times K + CoTime_{(i,j)}(T)\times L \quad (1)$$

The above formula only considers the information leakage between two VMs, in the actual environment, a physical machine is often running a number of virtual machines. In this paper, whether the sender has a replication run, or the receiver has a conspirator is unknown so the maximum of leakage is the key to judge whether a system is safe or not. and we set four cases and theirs leakage formula as follow.

(NR, NC), which is the case where the sender does not have replication and the recipient does not have the collusion capability.

$$SysLeak_{i,NR,NC}(T) = \max_{j\in VM-i}(PairTime_{(i,j)}(T)\times K + CoTime_{(i,j)}(T)\times L) \quad (2)$$

(R, NC) is the case where the receiver can collaborate but the recipient does not have replication,

$$SysLeak_{i,NR,C}(T) = \sum_{k\in I}\left(PairTime_{(i,k)}(T)\times K + CoTime_{(i,k)}(T)\times L\right) \quad (3)$$

Followed by the sender has replications but the recipient can't collaborate as (NR, C):

$$SysLeak_{i,R,NC}(T) = \max_{j\in VM-i}\sum_{k\in I}\left(PairTime_{(k,j)}(T)\times K + CoTime_{(k,j)}(T)\times L\right) \quad (4)$$

Finally, the sender has replications and the recipient can collaborate which is (R, C):

$$\mathrm{SysLeak}_{i,R,C}(\mathrm{T}) = \sum_{l \in J} \sum_{k \in I} \left(\mathrm{PairTime}_{(\mathrm{k,l})}(\mathrm{T}) \times \mathrm{K} + \mathrm{CoTime}_{(\mathrm{k,l})}(\mathrm{T}) \times \mathrm{L} \right) \quad (5)$$

4. Schedule Algorithm

In order to improve the ability of anti-side-channel attack of IaaS, we design two virtual machine scheduling algorithm for different workload need in IaaS.

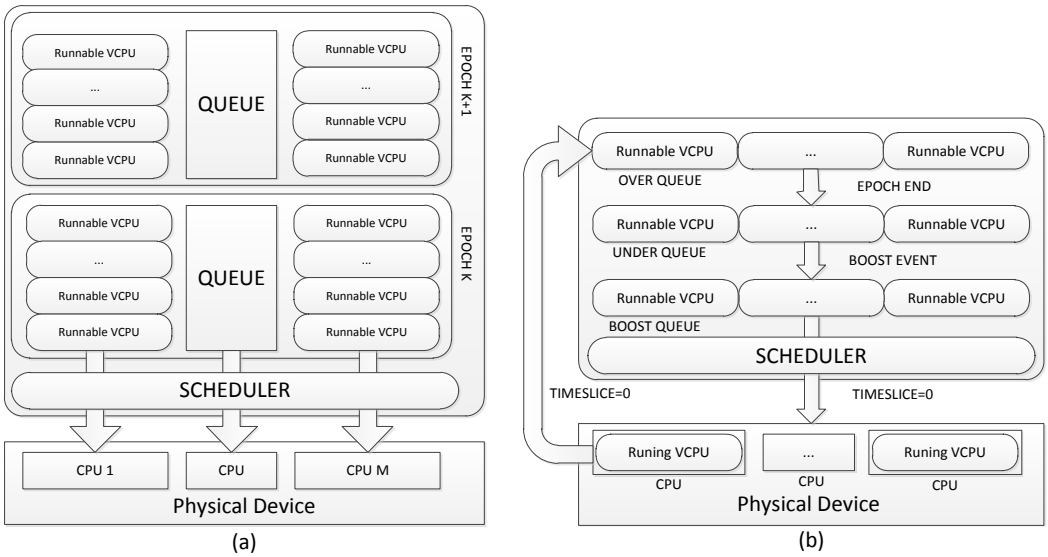

Fig. 2: Workflow of algorithm: (a) Latency-sensitive algorithm (b) CPU-hungry algorithm

4.1. Latency-Sensitive Algorithm

Algorithm 1 Latency-sensitive algorithm

```
1: function LatencyAlgorithm(VMs)
2:  Plan[CPU_number][VM_number/CPU_number]
3:  while there are VMs not in plan
4:       position=choosePosition(VM, Plan)
5:       Plan[position%CPU_number][position/CPU_number]=VM
6:  end while
7:  return Plan
8: function choosePosition(VM, Plan)
9:      while there are position in plan not calculate the score
10:          score=evaluate(VM, position)
11:     end while
12: return position with smallest score.
```

Algorithm 1 shows our latency-sensitive algorithm suits the latency-sensitive workload. Figure shows the execution of latency-sensitive algorithm. When most of VMs are latency-sensitive workload, to make sure every VMs can get PCPU to run, we set a matrix called plan to achieve that. To have a better understanding of this algorithm, I'd like to take a introduction to some important data structure:

- Plan: The plan is a (the number of PCPU)× (running VCPU/the number of PCPU) size matrix. The scheduler will assign VCPU to PCPU according to the plan.
- Epoch: an epoch is a plan's running time.
- Position: The position in plan means which PCPU and the time the VCPU is assign to run during one epoch.
- Score: The score means the influence making by choose one position. If it improve the value of SysLeak, then this choice will get a big score and would not be chosen.

In this algorithm, we create a plan for each epoch. Before a epoch finish, the scheduler calculates a new plan. During the process of making plan, we choose VM according to its leakage maximum. For each VM, leakage from large to small, we enumerate the feasible position and choose the position make the least influence to SysLeak.

4.2. CPU-Hungry Algorithm

Algorithm 2 CPU-hungry algorithm

```
1: function CPUhungryAlgorithm(CPUID)
//this algorithm will activate after VCPU running out the time slice and the CPUID is which the VCPU ran
2:  if there are VMs in Under Queue or Boost Queue
3:        VMID=ChooseVM(CPUID)
4:  else
5:  take VMs in Over Queue to Under Queue, goto 2
6: function chooseVM(CPUID)
7:       while there are VCPUs in Over Queue or Under Queue not calculate the score
8:         score=evaluate(VM, CPUID)
9:     end while
10:return position with smallest score.
```

Algorithm 2 shows our CPU-hungry algorithm suits the CPU-hungry or general workload. Figure shows the execution of CPU-hungry algorithm. In this algorithm, it is more like a traditional efficiency-first scheduling algorithm. Let's introduce some data structure:

- Over Queue: Over Queue stores the time out VCPU. After one VCPU is running out its time slice, it will go to the Over Queue. When the Under Queue and Boost Queue is clear, the VCPU in Over Queue will go to Under Queue and Boost Queue.
- Under Queue: Under Queue store the VCPUs which still have the time slice to run. If Boost Queue is empty, next running CPU will choose from the first VCPU in Under Queue.
- Boost Queue: When VCPU in Under Queue receives an event from event channel, this VCPU will go to Boost Queue. VCPU in the Boost Queue is considered first after any VCPU go to Over Queue.
- Score: The score means the influence making by choosing one CPU.

In this algorithm, scheduler's main work is to choose next VCPU for the PCPU in idle state. So we need to update the leakage once a CPU running out its time slice and calculate the score of different VCPU in Boost Queue for choosing the next CPU.

5. Information Leakage Simulation

We use a simulation to test and verify the validity and efficiency of the virtual machine schedule algorithm as a defense against side-channel. We focus on information leakage and consuming time.

Experimental setup. We assume that there is 20 virtual machines running in a physical server. Each virtual machine uses one virtual CPU. The server has 4 physical CPU which means 4 virtual machines running in the server at most at a time. Each Virtual machine comes and leaves randomly and independently. Each experiment will take 10 times of simulation with different average virtual machine running time, because virtual machine runs randomly. And we get average leakage and total running time which means all virtual machine running out.

We compare the leakage and time achieve by our algorithms with two strawman solutions: random schedule algorithm and efficiency-first schedule algorithm.

Figure 3 shows the information leakage in 4 different situations. Figure 2 shows the consuming time in 4 different situations. Table 1 and Table 2 shows a more accurate comparation result in information leakage and consuming time. The table's value means the influence made by the algorithms. Table 1 descripts the

comparation between our schedule algorithms and random schedule. Table 2 compares with efficiency-first schedule. According the tables, our algorithms can mitigate the threats effectively.

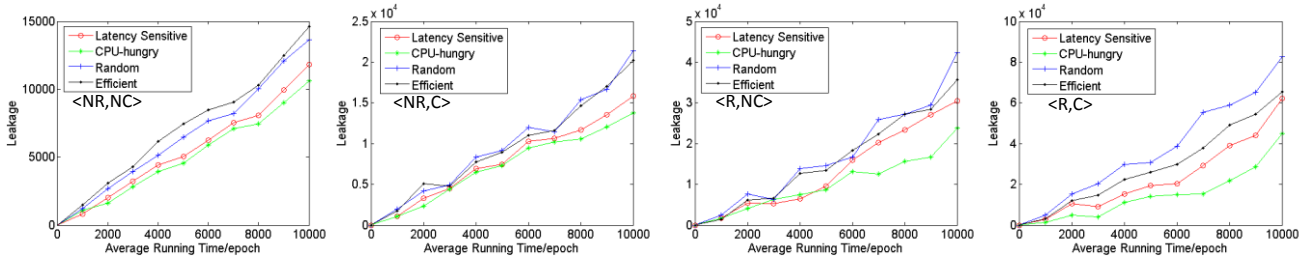

Fig. 3: Information leakage in different situation

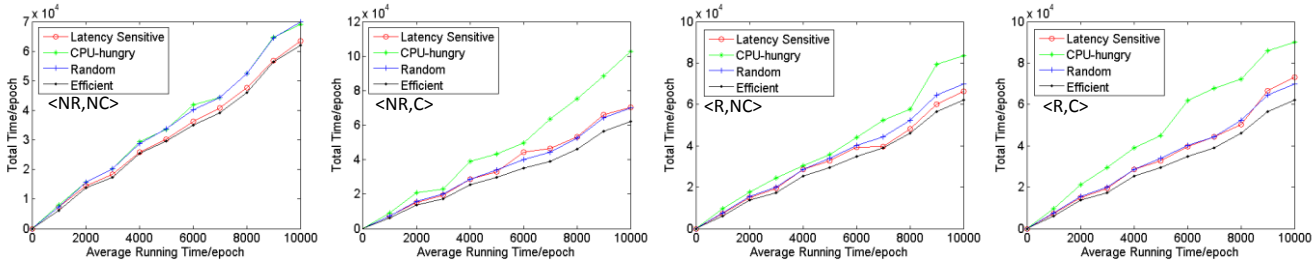

Fig. 4: Consuming time in different situation

Table 1: Information leakage and consuming time comparing with random schedule

		<NR,NC>	<NR,C>	<R,NC>	<R,C>
Latency-sensitive	Information leakage	-18.66%	-19.93%	-23.90%	-39.86%
	Consuming time	-8.58%	0.93%	-4.35%	-0.87%
CPU-hungry	Information leakage	-24.23%	-26.99%	-36.37%	-63.60%
	Consuming time	1.17%	32.89%	15.57%	38.33%

Table 2: Information leakage and consuming time comparing with efficiency-first schedule

		<NR,NC>	<NR,C>	<R,NC>	<R,C>
Latency-sensitive	Information leakage	-26.66%	-18.31%	-14.95%	-21.73%
	Consuming time	4.93%	15.78%	9.76%	13.72%
CPU-hungry	Information leakage	-31.84%	-25.15%	-28.56%	-53.00%
	Consuming time	16.11%	52.31%	32.68%	58.69%

6. Discussion

6.1. Conclusion

Based on the analysis of the problem of side-channel, we proposed a new virtual CPU scheduling algorithm prevent cloud from side channel attack. After the analysis of the factors that affect the efficiency of side channel attacks, we formalized the threat of side channel attacks to the system. Two scheduling algorithms for reducing the threat of virtual machines are designed according to the requirements of different workloads. The simulation results show that through software layer of virtual machine operation management, the algorithm can mitigate the side channel threat in IaaS cloud.

6.2. Limitation

Our algorithms have some limitations. First, we assume that all information leakage in system will not decrease as time goes by and some other virtual machine use these shared resource. Second, different side-channel extracts different information and cause different threats to clients. Side-channel threat classification will help improve the effectiveness of the algorithm.

7. Acknowledgements

Thanks for the help from Soo-Jin who propose Nomad system.

8. References

[1] Moon, S.J., V. Sekar, and M.K. Reiter. Nomad:Mitigating Arbitrary Cloud Side Channels via Provider-Assisted Migration. in Proceedings of the 22nd acm sigsac conference on computer and communications security. 2015. ACM.

[2] Irazoqui, G., et al., Wait a Minute! A fast, Cross-VM Attack on AES. Lecture Notes in Computer Science, 2014. 8688: p. 299-319.

[3] Zhang, Y., et al. Cross-VM side channels and their use to extract private keys. in ACM Conference on Computer and Communications Security. 2012.

[4] Yarom, Y. and K. Falkner. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. in Usenix Conference on Security Symposium. 2014.

[5] Ristenpart, T., et al. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. in ACM Conference on Computer and Communications Security. 2009.

[6] Gullasch, D., E. Bangerter, and S. Krenn. Cache Games -- Bringing Access-Based Cache Attacks on AES to Practice. in IEEE Symposium on Security and Privacy. 2011.

[7] Irazoqui, G., T. Eisenbarth, and B. Sunar, Cross Processor Cache Attacks. Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. ACM, 2016: p. 353-364.

[8] Kim, T., M. Peinado, and G. Mainar-Ruiz. STEALTHMEM: system-level protection against cache-based side channel attacks in the cloud. in Usenix Conference on Security Symposium. 2012.

[9] Wang, Z. and R.B. Lee. New cache designs for thwarting software cache-based side channel attacks. in International Symposium on Computer Architecture. 2007.

[10] Raj, H., et al. Resource management for isolation enhanced cloud services. in ACM Cloud Computing Security Workshop, Ccsw 2009, Chicago, Il, Usa, November. 2009.

[11] Shi, J., et al. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. in Ieee/ifip International Conference on Dependable Systems and Networks Workshops. 2011.

[12] Mars, J., et al. Bubble-Up:increasing utilization in modern warehouse scale computers via sensible co-locations. in Ieee/acm International Symposium on Microarchitecture. 2011.

[13] Keller, E., et al., NoHype:virtualized cloud infrastructure without the virtualization. Acm Sigarch Computer Architecture News, 2010. 38(3): p. 350-361.

[14] Tang, L., J. Mars, and M.L. Soffa. Contentiousness vs. sensitivity:improving contention aware runtime systems on multicore architectures. in International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era. 2011.

[15] Crane, S., et al. Thwarting Cache Side-Channel Attacks Through Dynamic Software Diversity. in NDSS Symposium. 2015.

[16] Li, P., D. Gao, and M.K. Reiter. Mitigating access-driven timing channels in clouds using StopWatch. in Ieee/ifip International Conference on Dependable Systems and Networks. 2013.

[17] Martin, R., J. Demme, and S. Sethumadhavan. TimeWarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks. in International Symposium on Computer Architecture. 2012.

[18] Vattikonda, B.C., S. Das, and H. Shacham. Eliminating fine grained timers in Xen. in ACM Workshop on Cloud Computing Security Workshop. 2011.

[19] Kong, J., et al. Hardware-software integrated approaches to defend against software cache-based side channel attacks. in IEEE International Symposium on High PERFORMANCE Computer Architecture. 2009.

[20] Wang, Z. and R.B. Lee. A novel cache architecture with enhanced performance and security. in Ieee/acm International Symposium on Microarchitecture. 2008.

[21] Varadarajan, V., T. Ristenpart, and M. Swift. Scheduler-based defenses against cross-VM side-channels. in Usenix Conference on Security Symposium. 2014.