Available online at www.sciencedirect.com

ScienceDirect

Procedia Technology 25 (2016) 232 – 239

Procedia Technology

Global Colloquium in Recent Advancement and Effectual Researches in Engineering, Science and Technology (RAEREST 2016)

A Novel Approach to Prevent Cache-Based Side-Channel Attack in the Cloud

Muhammed Sadique UK*, Divya James

Master of Technology, Rajagiri School of Engineering and Technology, Cochin, Kerala
Assistant Professor, Rajagiri School of Engineering and Technology, Cochin, Kerala

Abstract

Cloud computing means different things to different people. It is another way of outsourcing and aggregating any computational hardware service provided over the Internet. It can able to share resources among multiple mutually wariness clients. While there are scads of asset to this system, this sort of resource sharing permit new forms of information leakage across virtual machines. In this paper, we interrogate the time information parameter leakage through cache based side-channel attack and the usage of CPU-cache based side-channels in the cloud. Finally we developed a novel approach which is necessary to mitigate these kinds of attacks in the cloud environment, and correlated to traditional cloud technology.

Peer-review under responsibility of the organizing committee of RAEREST 2016

Keywords: CPU Cache; Side-Channel; Cloud Computing; Side-Channel Attack; Cache based Side-Channel; Sequential Side-Channel.

1. Introduction

Cloud computing is a new paradigm shift with service models in distributed computing and has acquired greater attention in the research communities. It is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (i.e., networks, servers, storage, applications and other services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. It can also provide on-demand services to users based on the distributed and virtualization technologies. Thus cloud has the concept of Mutually Distrusting co-resident Clients. Unlike most other standards, cloud computing permits potentially pernicious or inimical clients access to the same hardware. Specifically, access to the same hardware makes them chance to exploit the physical properties of the machine for communication or information leakage. Exploiting these properties is called using a hardware-based side-channel [3]. It typically involves correlating the

Peer-review under responsibility of the organizing committee of RAEREST 2016
doi:10.1016/j.protcy.2016.08.102

software's higher level functionality with the underlying hardware phenomena. The attacker exploits co-resident systems by presuming software functionality from observed hardware phenomena. With an established correlation, these phenomena can be analysed to infer what is occurring within the software program at a given time. While virtualization and other technologies are heavily used in cloud systems [4], they do not fully prevent information leakage along these channels. Currently cache-based side-channel attacks are considered to be the most dangerous, among side-channel attacks. They have recently been used to extract private keys in a Cloud environment [8] and precipitate high-bandwidth information leakage [12]. Since the potential for these attacks has been exposed, there have been several attempts to reduce such situations. Unfortunately, the solution like cache flushing [2] makes the cache useless in its functionality. And it also make huge data lose and expensive too. There are other solutions which either require the client to modify their software [5], [6], or the underlying hardware [7]. From our studies of cloud systems, we believe that either of these modifications disrupts the relationship between the cloud provider and their users, which we refer to in this paper as the "cloud model". We see the lack of an existing, non-intrusive solution to these attacks to be the motivation behind this work. Our goal in this paper is to provide solutions that prevent cache-based side-channels attack, more specifically preventing the time information parameter leakage in the cache in the cloud without affecting the cache functionality. Typically this paper concentrates in sequential side-channel. The major contributions of the paper are summarized as follows:

- A study of cache-based side-channels in a cloud environment. Mainly focus on Sequential type of side channel attack [13].
- Server-side defences against cache-based side channel. This includes a technique to prevent the side-channel's occurrence as well as an algorithm designed to implement the technique. The algorithm applies the solution in a minimalistic fashion to help minimize resulting overhead.

The rest of the paper is organized as follows: Section 2 explains the context in which these attacks occur and are mitigated. Section 3 describes our motivation, detailing the need for our solution, and comparing it to related work in the field. Section 4 specifies the details of how the attacks work. Section 5 details our approach to the problem. Section 6 concludes the paper.

2. Background

2.1. The Cloud Model

In this paper cloud model is referred as the specific relationship between cloud providers and their users. This model refers to two key attributes of cloud functionality that we aim to perpetuate while making the paradigm more secure. The first is that users tend to run canonical software in the cloud. Often a user will either not have the technical skill necessary to run their workload for security purposes, or will not have access to the code to make the modifications necessary. Hence it does not require any software modifications by the client or on the client-end of the interface. The second is that cloud systems are typically built on canonical hardware, and requiring them to do otherwise would be financially, and structurally costly. Hence it does not require any modifications to the underlying hardware. With these concepts in mind, we specify the conditions for holding with the cloud model.

2.2. Side-Channel

Side-Channel is a mode of bypassing virtual machine for gaining information from the physical implementation rather than brute force or theoretical weaknesses in the algorithm. Examples include verifying if two virtual machines are co-resident, and more dangerously, the extraction of cryptographic private keys from unwary hosts [8]. Similar research into side- channels yields additional attacks that can be migrated to the cloud [3, 9, 10]. When attention was first drawn to side-channels [11] there were several proposals for how to mitigate their potential [3], [7]. Various techniques to implement these solutions, such as altering how the algorithm uses the cache, or customizing the hardware channel would require either the modification of the source code, the hardware, or cause an unacceptable amount of overhead. Cache flushing [1] also cannot consider as the solution, for traditional cache-based side-channels because it generate large amounts of overhead and data lose.

2.3. Side-Channel Attack

Side-channel attack is a form of reverse engineering. Electronic circuits and Software programs are inherently leaky – they produce emissions or a mean of communication as by-products that make it possible for an attacker without access to the circuitry itself to deduce how the circuit works and what data it is processing. Time, heat and electromagnetic emissions are viable sources of information for an attacker. Because these leakages do not play a part in the operation of the circuit itself – they are simply side effects of it working – the use of them to perform reverse engineering has earned the term 'side-channel analysis' or 'side-channel attack'. For networked systems, time-based attacks are the most feasible and have been exploited. Systems that use memory caches are particularly vulnerable to timing-based attacks because of the significant difference in performance of a given section of code based on whether accesses to the cache hit or miss and force a slower read or write to main memory.

2.4. Cache-Based Side-Channel Attack

A cache is a small and fast storage area used by the CPU to reduce the average time to access main memory. It stores copies of the most frequently used data. When the processor needs to read a location in main memory, it first checks to see if the data is already in the cache. If the data is already in the cache (cache hit), the processor immediately uses this data instead of accessing the main memory, which has a longer latency than a cache. Otherwise (cache miss), the data is read from the memory and a copy of it is stored in the cache. The minimum amount of data that can be read from the main memory into the cache at once is called a cache line or a cache block, i.e., each cache miss causes a cache line to be retrieved from a higher level memory. Cache attacks exploit the cache hits and misses that occur. It is assumed to be able to observe the total execution time of the cipher, that is the number of cache hits and misses are observed. This attack is based on statistical inferences. The resulting differences in access times for cache hits and cache misses will serve as the communication means within the channel. Hence from the difference in time generates time information parameter leakage. We can term it as cache-based side channel attack.

3. Related Work

Earlier works have indicated that cache-based side-channels can be exploited in the Cloud to glean information. Such cases include determining whether two machines are co-resident [13], and the extraction of cryptographic private keys from incautious hosts [8]. The latter particularly validates the severity of a side-channel attack in the Cloud and the potential for similar attacks to migrate to a Cloud environment [10], [14]. The attention given to side-channels in non-Cloud environments since 70s [11], there have been many solutions to the problem proposed [3], [7]. These include: altering the functionality of the hardware channel, disabling the hardware channel, or modifying the victim code to break the correlation between the program's execution and the hardware phenomena. Implementing any of these defenses would require the customization of either all hardware intended for use in the Cloud, or else all software intended to be run in a Cloud environment. Both of these solutions conflict with the Cloud model, as they would either restrict the hardware requirements or the client skill level needed to use the Cloud. Kim *et al*. [5] have developed a solution for cache-based side-channels in Cloud systems. In their solution, they prevent cache-based side-channels by giving each VM exclusive access to a sectioned portion of the cache they call a *stealth page*. In order to have software applications access these hidden pages, their solution requires the user to make client-side modifications to the software being executed in the guest VM. However, we believe that requiring the client to modify their software violates the Cloud model and demonstrates the need for a solution transparent to the Client. For our solution, we implemented a purely server side defence for cache-based side-channels in the Cloud. To make it fully compatible with the Cloud model, we impress the constraints that it both prevent cache-based side-channels between co-resident VMs, and that it requires no modifications of the underlying hardware nor of the software being run on a VM. For non-cloud environments, it has been previously suggested to use cache flushing as a way to prevent side channels [3]. However, the idea was always deemed prohibitively expensive. We believe, however, that the discrete nature of the Cloud can be exploited to create a reasonable solution specific to Cloud environments. In recent existing methodology [1], [2], cloud paradigm uses Cache

flushing in much more effective technique by exploiting knowledge of Cloud using scheduling architecture. The common inefficiency of the technique can be summarized in two side-effects: The reduced usefulness of the cache and the increased cost due to flushing the cache. Existing solution, shown in Fig. 1, is designed such that both parties view the cache as flushed (invalid) upon gaining access to the cache. The existing solution attempts to mitigate the data loss due to frequent cache flushing by only flushing the cache when the CPU switches domains. If they have no data in common, then flushing the data will not happen. Flushing the cache only when a context switch occurs between domains will significantly reduce the amount of flushes necessary. In addition, the scheduler can enforce specific rules for the frequency of context switching between domains. Thus the solution should prevent sequentially scheduled VMs from gaining any information about the previous VMs cache usage, thereby blocking the prospective side-channel.

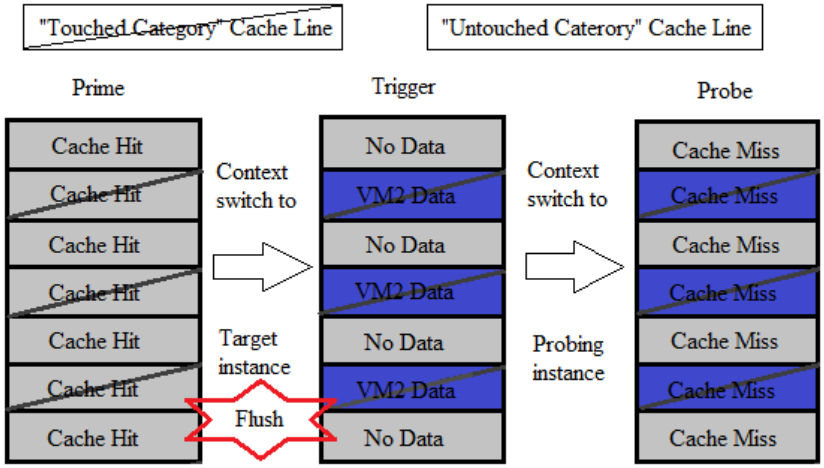


Fig. 1. Existing solution's effect on the PTP technique.

The technique includes two new functions within the hypervisor: One is a server run process to revert the cache to a blank slate (flush the cache), and the other is a tainting algorithm in the scheduler for deciding when flushing the cache is necessary. Algorithm was implemented to record which VMs own data currently in the cache and to determine when it is necessary to flush those data. The outcome of Algorithm is that a cache flush occurs only when a context switch changes from one domain to another where the second domain has the ability to establish a side-channel with the first. Switching to the idle domain, or to the same domain, will not invoke a cache flush. Still the existing solution will reduce usefulness of the cache and increased cost due to flushing the cache other than the above conditions. Therefore cache flush will happens, in case of context switch between two virtual machines which fetch different data's. And if first virtual machine came again to access the same files accessed before. Then it will face cache miss. This is the drawback of existing methodology which leads to reduced usefulness of cache and the increased cost.

4. Technique

Cache-based side-channel attacks can be categorized into two types: Sequential and Parallel. These attacks vary in whether they employ concurrent access to the CPU-cache or not. This paper discusses possible mitigations for a cache-based side-channel attack of sequential type. The first documented form of a cache-based side-channel attack explored in the cloud was by Ristenpart et al. [13] who demonstrated the use of side-channels to verify virtual machine co-residence in Amazon's EC2. As part of their work, they explored the use of a previously identified cache-based side-channel technique, which they refer to as the Prime+Probe technique, in a cloud environment. The result of such work is the prime+trigger+probe (PTP) technique, illustrated in Fig. 2, a variation designed to work in

cloud environments. The purpose of their experiment, using the PTP technique, was to see if a cache-based side-channel could be established between two guest domains in the cloud. The channel was established such that the first VM (referred to as the probing instance) could receive a message that the second VM (the target instance) encodes in its usage of the cache. The basic version of the PTP technique shown in Fig. 2 is an example of a sequential side-channel.

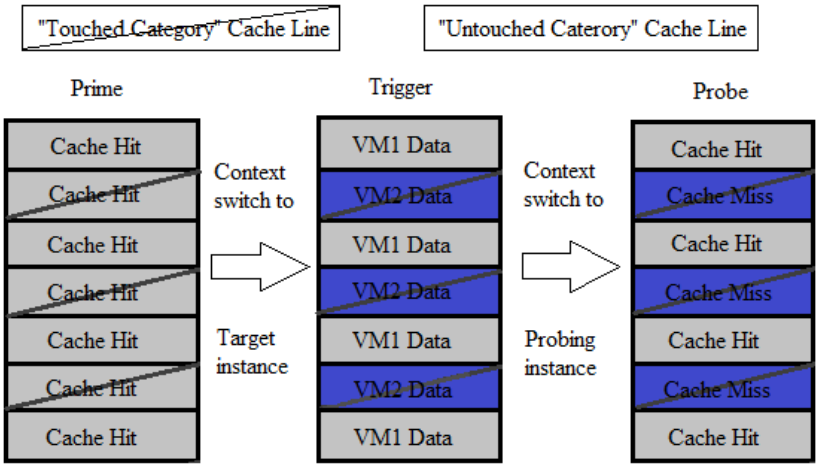


Fig. 2. The prime+trigger+probe technique.

In the PTP technique, the probing instance first separates the cache lines into two categories: the Touched category and the Untouched category. Once the categories have been established, the probing instance primes the cache by filling as many cache lines as it can. It then establishes an access time baseline by reading from each line in both categories. Having just been primed, each cache line should yield a cache hit, regardless of category, thus keeping the baseline access times low. This process is highlighted in the “Prime” step of Fig. 2. Having done this, the probing instance now has a series of values which represent how long it took to access each cache line with a primed cache. Once the cache has been primed, and the baseline established, the probing instance must trigger (context switch to) the target instance by busy looping, or otherwise giving up its time slice. When the target instance begins its time slice, it heavily accesses the cache lines in one of the pre-defined categories of the cache (the Touched category), but not the other (the Untouched category). The effect of this switch on the data can be seen in the Trigger step of Fig. 2. In this figure, the target instance is accessing every second cache line in an access pattern we have pre-defined. The resulting set of accessed lines we define as the Touched category of the cache. By contrast, cache lines in the untouched category were not accessed by the target instance. When the CPU core context switches back to the probing instance, the instance probes the cache by re-measuring the access times for each line. This is illustrated in the Probe step of Fig. 2. If there is a significant increase in the access times for cache lines in the touched category compared to the untouched category, then the probing instance can assume that the target instance was trying to communicate.

5. Our Approach

5.1. Over View

The existing solution to all sequential cache-based side-channels is to avoid both the probing and the target instances from different cache access time. Since access to cache cannot be denied to either instances. Our solution focuses on disabling the difference in access time other than through cache flush, which reduces usefulness of the cache and increase cost and data loss. Thus our solution is to use cache wait function. The objective of our solution

is to prevent time information parameter leakage in the cache of the cloud. More specifically, if virtual machines cannot generate differences in access time, that is, time between the cache hit and the cache miss. Then the problem can be solved.

5.2. Expected Issues and Mitigations

The common expected issues for inefficiency can be summarized in three side-effects of the implementation: The reduced usefulness of the cache, the increased cost and the data lose due to flushing the cache. When the cache is flushed, all data held within is expunged, making the cache useless until it once again has relevant data stored within. The more often these flushes occur the less likely there is to be relevant data in the cache when it is accessed. When cache-based side-channels were being investigated in a non-virtualized environment, flushing the cache was deemed too expensive for general use [3]. Our solution attempts to mitigate the data loss due to cache flushing, is by waiting the cache execution when the CPU switches domains under single condition. When proper isolation between VMs is enforced, one domain should not have any cache data in common with another. The second issue relies on implementing a solution with a minimal amount of overhead, which will usually rely on the hardware. Waiting the cache execution only when a context switch occurs between domains under single condition will significantly increase the usefulness and reduce cost and data loss of cache.

5.3. Implementation Technique

Our solution to sequential side-channels involves the incorporation of our cache wait technique into a canonical cloud system. The technique includes two new functions within the hypervisor: One is a server run process to make the cache execution to a wait stage (cache-wait), and the other is a tainting algorithm in the scheduler for deciding when waiting the cache execution is necessary.

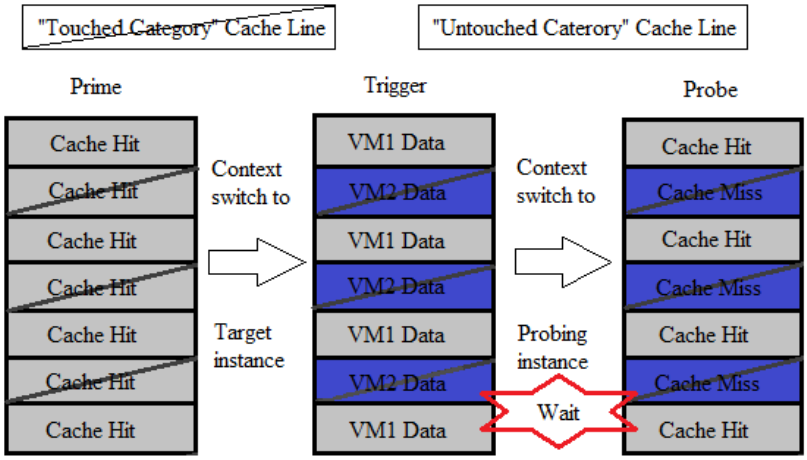


Fig. 3. Our solution's effect on the PTP technique.

5.3.1. Cache-Wait Function

Cache-Wait function waits the cache execution for the specific time. Cache-Wait operates only in one condition. When the context switch happens and the current domain requires data, both from the cache memory (cache hit) and the main memory (cache miss). And if the total time required for accessing the main memory is higher than the cache memory. Only at this particular instant, the Cache-Wait function operates Fig 3. That is, when context switch happens and the current VM performs cache hit and cache miss. And if the time taken for the cache miss is greater than the cache hit, at this instant Cache-Wait operates. The function is such that Cache-Wait will hold the cache

execution process for the specific time. The specific time is determined from the difference in the accessing time required for fetching data from the main memory and the cache memory. That is, the difference in accessing time required between cache miss and cache hit.

5.3.2. Decision Algorithm

Flushing a high level cache on a modern machine can be a time consuming process. As mentioned in the above section, it is better to wait the cache and also only when it is necessary. When context switch happens and the current VM performs cache hit and cache miss. And if the time taken for the cache miss is greater than the cache hit, at this instant Cache-Wait operates. Algorithm 1, was implemented to record which VM's data currently reside in the cache and to determine when a wait is necessary.

```
Function contextSwitch(DomX,DomY) { // from DomX to DomY
If Main_T  > Cache_T
waitCache(); return;
} EndFunction
```

Algorithm 1. Pseudo-code for tainting domain with cache data.

The outcome of Algorithm 1, is that a cache wait occurs only when a context switch changes from one domain to another where the second domain has the ability to establish a side-channel with the first. Allowing a wait only in a single condition ensures the usefulness of cache, decrease the cost and prevent the data lose. At present Algorithm 1 is quite conservative and does not distinguish between the Prime and Probe steps of a side-channel attack. In practice, a wait would only be necessary before the Probe step, which could be recorded in the scheduler by further tainting. Theoretically, this would further reduce the overhead of this algorithm by at least 75 percentages but is left by the authors for future work.

5.4. Statistical Analysis

Table 1. Flush and Wait function required with respect to the type of data accessed.

Data Type	Flush()	Wait()
Same	N	N
Different	Y	N
Both	Y	Y
Idle	N	N

The type of data accessed by the VM's in every cycle can be any of the four in Table 1. Hence it is clear that out of four cases, Cache-Wait function needed only in a single case where as Cache-Flush function needed in two cases.

Fig. 4. Frequency of type of data accessed by VM's.

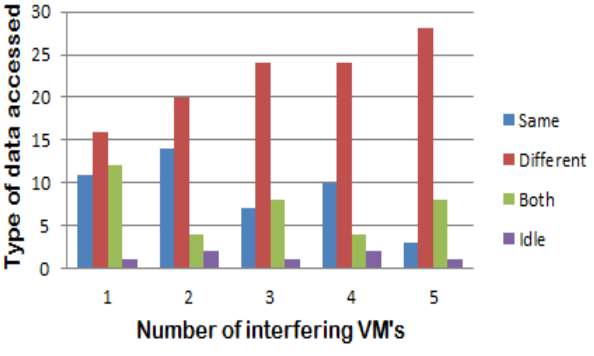
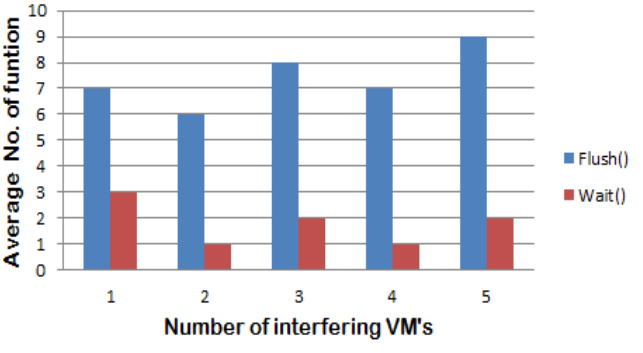


Fig. 5. Comparison of use of flush() and Wait().



The type of data accessed by VM's varies with respect to the number of interfering VM's. The analysis was conducted by considering 40 cycles of data access from each Virtual Machine. From Fig 4., it can be concluded that the number of access for different data is increased with the increase in VM's interfering. Hence the function Cache-Flush will increase accordingly as shown in Fig 5. The Cache-Wait function which applied only in a single case can effectively overcome the drawbacks of the Cache-Flush, like reduced usefulness of the cache, increased cost and data lose.

6. Conclusion and Future Work

Two unique security attributes of the cloud motivated this research. First, the cloud's architecture is particularly susceptible to cache-based side-channel attacks. Second, such attacks in the cloud cannot be solved by conventional means without interfering with the cloud model. To address these problems, we have developed a new techniques designed to prevent cache-based side-channels which is dealing with sequential side-channels. This technique can be implemented without interfering the cloud methods of operation. Our solution is unique in that it both addresses cache-based side-channels in the Cloud and does not interfere with the Cloud model (requires no changes to the client-side code, nor to the underlying hardware). The time information parameter leakage is one of the biggest threads in cloud environment. Using PTP technique we have investigated this information leakage through cache-based side channel attack and explored the drawbacks in existing solution. Later we proposed effective methodology which can overcome the drawbacks of existing solution. Statistical analysis presented in this paper demonstrated that the suggested algorithm is efficient. The future plan is to implement this approach in real- time environment and in the Docker.

Acknowledgements

It worth mentioning the contributions of M. Godfrey and M. Zulkernine through their publishes [1, 2].

References

[1] M. Godfrey and M. Zulkernine. A server-side solution to cache based side-channels in the cloud. IEEE 6th Int. Conf. Cloud Comput., 2013, pp. 163–170.
[2] M. Godfrey and M. Zulkernine. Preventing Cache-Based Side-Channel Attack in a Cloud Environment. IEEE transactions on cloud computing, 2014, vol. 2, No 4, pp. 395-408.
[3] D. A. Osvik, A. Shamir, E. Tromer. Cache attacks and countermeasures: The case of AES, in Proc. Cryptographers' Track RSA Conf. Topics Cryptol., 2006, pp. 1–20.
[4] S. Pearson. Privacy, security and trust in cloud computing, in Privacy and Security for Cloud Computing, S. Pearson and G. Yee, Eds. London, U.K.: Springer, 2013, pp. 3–42.
[5] T. Kim, M. Peinado, G. Mainar-Ruiz, STEAL THMEM: System-level protection against cache-based side channel attacks in the cloud, in Proc. 21st USENIX Conf. Security Symp., Berkeley, CA, USA, 2012, p. 11.
[6] J. Shi, X. Song, H. Chen, B. Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring, in Proc. IEEE/IFIP 41st Int. Conf. Dependable System Network Workshops, 2011, pp. 194–199.
[7] D. Page. Defending Against Cache-Based Side-Channel Attacks, Inf. Security Tech. Rep., Vol. 8, Issue 1, 2003, pp. 30- 44.
[8] Y. Zhang, A. Juels, M. K. Reiter, T. Ristenpart. Cross-VM side channels and their use to extract private keys, in Proc. ACM Conf. Comput. Commun. Security, 2012, pp. 305–316.
[9] D. X. Song, D. Wagner, X. Tian. Timing analysis of keystrokes and timing attacks on SSH, in Proc. 10th Conf. USENIX Security Symp., 2001, vol. 10, p. 25.
[10] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri. Cryptanalysis of DES implemented on computers with cache, in Proc. 5th Int. Workshop Cryptograph. Hardware Embedded Syst., 2003, pp. 62–76.
[11] B. Lampson. A Note on the Confinement Problem, In Commun. ACM, vol 16, Issue 10, 1973, pp. 613-615.
[12] Z. Wu, Z. Xu, H. Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud, in Proc. 21st USENIX Conf. Security Symp., 2012, p. 9.
[13] T. Ristenpart, E. Tromer, H. Shacham, S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds, in Proc. 16th ACM Conf. Comput. Commun. Security, 2009, pp. 199–212.
[14] S. Xi, J. Wilson, C. Lu, C. Gill. RT-Xen: Towards Real-time Hypervisor Scheduling in Xen, In Proc. of the Ninth ACM Int. Conf. on EMSOFT, 2011, pp. 39-48.