Cache-Based Side-Channel Attacks Detection through Intel Cache Monitoring Technology and Hardware Performance Counters

Mohammad-Mahdi Bazm*, Thibaut Sautereau*, Marc Lacoste*, Mario Sudholt†, and Jean-Marc Menaud†
*Orange Labs, 44 Avenue de la Republique, 92320 Chatillon, France
Email: {mohammadmahdi.bazm,thibaut.sautereau,marc.lacoste}@orange.com
†IMT Atlantique, 4 Rue Alfred Kastler, 44307 Nantes, France
Email:mario.sudholt@imt-atlantique.fr, jean-marc.menaud@imt-atlantique.fr

***Abstract*—Research on cache-based side-channel attacks shows the security impact of these attacks on cloud computing. Therefore, the detection of cache-based side-channel attacks has received more attention in IaaS cloud infrastructures because of improvements in the attack techniques. However, such detection requires high resolution information, and it is also a challenging task because of the fine-granularity of the attacks. In this paper, we present an approach to detect cross-VM cache-based side-channel attacks through using hardware fine-grained information provided by *Intel Cache Monitoring Technology (CMT)* and *Hardware Performance Counters (HPCs)* following the Gaussian anomaly detection method. The approach shows a high detection rate with 2% performance overhead on the computing platform.**

***Index Terms*—side-channel attacks detection, Intel cache monitoring technology, hardware performance counters, cloud security, isolation.**

I. Introduction

Cloud computing makes use of *virtualization* technology for dynamic resource allocation, consolidation and service provisioning. The virtualization layer schedules allocation of hardware resources among virtual instances such as virtual machines (VMs) with different operating systems. However, this layer has shown several security issues related to itself and virtualized instances. Although, the virtualization layer provides isolation, one new challenge is to guarantee strong isolation between virtualized instances running on top of the virtualization layer. OS and hypervisor traditionally implement resource isolation between processes or VMs using software/hardware techniques, such as memory management and access control to key shared physical resources. However, sharing hardware resources has an impact on isolation in virtualized environments. Furthermore, physical resources are often shared between independent users at very low-level through the virtualization layer. Such sharing allows bypassing security mechanisms implemented at virtualization layer, breaking isolation, and threatening the security of the overall virtualized infrastructure. *Side-channel attacks (SCA)* threaten isolation of virtualized environments. These attacks use a leaky channel to obtain secret information such as cryptographic keys. Such channels are either established through techniques or optimization mechanisms *e.g.*, memory deduplication used in the virtualization layer, or micro-architectural vulnerabilities in processors. The threat of side-channel attacks is becoming a security concern in cloud computing [1]. For instance, in IaaS platforms, a CPU cache which is shared between VMs leaks information on cache access patterns of running instances (cross-VM side-channel attacks). The Last-Level Cache (LLC) is one of resources shared among all cores of processor, and leaks cache access patterns of a VM running on another processor core than adversary [2] [3]. To obtain cache access patterns, an adversary does not need to any special privileged access. This is why these attacks gained attention in cloud infrastructures. Detection of side-channel attacks is hard because of relying on fine-grained information without exploiting special security barriers. These attacks may be detected at OS/hypervisor level through fine-grained information which are often provided by Hardware Performance Counters (HPCs): special platform hardware registers used to store statistics about various events of the processor (*e.g.*, number of *cache misses*). Existing approaches [4] [5] [6] use only HPCs as the source of raw information in their works to detect cache-based side-channel attacks. However, other hardware technologies such as *Intel Cache Monitoring Technology (CMT)* providing fine-grained information on the cache behavior of a virtual machine, might be useful to improve the precision of detection approaches.
In this work, we present a cross-VM side-channel attacks detection approach. It leverages Intel CMT and HPCs to detect side-channel attacks conducted between two VMs through the LLC. We made the following contributions: (1) we present a prototype to detect side-channel attacks using HPCs and Intel CMT for the first time; (2) we introduce the concept and implementation of the prototype; (3) we evaluate the prototype in terms of security and performance characteristics.

This paper is organized as follows. In Section II, we give an overview on existing approaches. Then, in Section III, we provide some background on the cache architecture in processors and on cache-based side-channel attacks. In Section IV, we give an overview on the design and implementation of the approach. In Section V, we evaluate the implemented prototype.

978-1-5386-5896-3/18/$31.00 ©2018 IEEE

Finally, we conclude with some perspectives in Section VI.

II. Related Work

Almost all the current detection approaches take advantage of Hardware Performance Counters (HPCs) in different ways to detect side-channel attacks. The reason for using HPCs is that they provide information by hardware to detect such fine-grained attacks. As a threshold-based detection approach; *Chiappetta et al.* [4] proposed a detection approach leveraging HPCs to detect the attacks. To analyze statistics collected through HPCs, they used machine learning and anomaly detection techniques to detect a side-channel attack between two processes (*i.e.*, spy and victim). As signature-based detection approaches; *Zhang et al.* [5] presented a real-time detection framework called *CloudRadar* to detect side-channel attacks between two VMs in the cloud. Their detection approach is based on the correlation between running a cryptographic application in the victim VM and then monitoring the malicious VM. They generate a signature for the sensitive application running in the victim VM, by calculating the *Fisher Score* [7] of different events provided by HPCs. The signature is then used by the detector to identify when the sensitive application is running. Once the application is identified, the detector starts to analyze performance events (*e.g.*, LLC-misses) of the adversary VM to detect any malicious behavior through anomaly detection. *Payer et al.* [6] also proposed *HexPADS* to detect inter-process cache-based side-channel attacks. *HexPADS* generates signatures for cache-based side-channel attacks, using HPCs. It then analyzes running processes in the system to detect which processes have the same signature as the generated signatures.

III. Background

A. Cache Architecture in Modern Processors

Caches of CPUs are specialized memory layers located between the CPU and the main memory. They are used to significantly improve the execution efficiency by reducing the speed mismatch between the CPU (operates in GHz clock cycles) and RAM (requires hundreds of clock cycles to be accessed). In modern processors, we can typically distinguish different levels of cache: L1, L2 and L3 (or the LLC). The L1 and L2 caches are usually private to each core and the LLC is shared among all cores of processor. The LLC is much larger (megabytes size) than the L1 and L2 caches (kilobytes size). Cache memories are divided into equal blocks called *ways* and consisting of *cache lines*. Cache lines with the same index in ways form a *set*. When the processor needs either to read or write data to a specific location in RAM, it first checks whether a copy of data is already present in one of the cache levels. If the data is found, a *cache hit* is triggered and the CPU performs operations on the cache quickly in a few clock cycles. Otherwise a *cache miss* occurred and the processor inevitably brings data from RAM at a much slower pace. There are two cache architectures in modern processors: *inclusive* and *exclusive*. Exclusive caches do not store redundant copies, hence enable higher cache capacity. For instance, exclusiveness is widely used in AMD Athlon micro-architectures. Otherwise, in *inclusive* architectures, all cache lines in higher levels are also available in the lower level. For instance, the architecture of LLC in Intel's Nehalem processors is inclusive to reduce snoop traffic between cores and the processor sockets [8]. Inclusiveness provides less cache capacity but offers higher performance to the processor.

B. Cache-based Side-channel Attacks

The virtualization layer enables running multiple VMs on a physical machine. Main physical resources of the host such as CPU and RAM are shared between VMs through the virtualization layer, which has a strong security impact on running VMs. Furthermore, such sharing is strongly at odds with isolation, both at the hardware level and at the software level. To make better use of shared resources, some optimization techniques such as *memory page deduplication* are used in the virtualization layer. These techniques may have an impact on isolation directly or indirectly. In such environment, processor's caches and especially the LLC are shared among running VMs or virtualized instances. These caches are widely exploited through cache-based side-channel attacks. Because the LLC is shared among all cores of processor, it is more exploited in the case of side-channel attacks between two VMs. The main idea of side-channel attacks rests on the eviction of cache lines from cache and measuring fetching time of memory lines from RAM to know which cache lines are re-accessed by the victim during an operation, to obtain the victim's cache access pattern. Furthermore, if a memory line was flushed by the victim from the cache, it takes more CPU cycles to fetch the line from the main memory when it is re-accessed. Otherwise, the line is already in the cache, thus requiring fewer CPU cycles. This timing difference is exploited to profile cache access patterns of the victim. Generally, any cache-based side channel attack consists of three principal steps: (1) the attacker evicts cache lines of victim VM from processor's cache by *e.g.*, filling cache sets (2) the attacker then waits for a given period of time (several CPU cycles) while the victim is doing some operations (3) finally, the attacker re-accesses the same cache lines which are evicted from caches, to know which ones are used by the victim during the execution of the victim's operations *e.g.*, encryption.

There are three classes of the attacks: *time-driven*, *trace-driven*, and *access-driven*. The main idea behind *time-driven* attacks is to measure the execution time of *e.g.*, an encryption operation, and derive a secret key from the gathered samples by finding any correlation between measured time and the operation. In *trace-driven* attacks [9], [10], the attacker inspects cache activities (*i.e.*, memory lines which are accessed) of the victim during its execution in order to obtain a sequence of cache hits and cache misses. For instance, monitoring which memory accesses to a lookup table lead to cache hits, allows finding indices of tables entries. In *access-driven* attacks [3], [11], the attacker tries to find any relation between *e.g.*, an encryption process and accessed cache lines, to exploit the pattern of cache accesses of the victim. Technically, we may also distinguish different techniques to perform a cache-based side-

channel attack: *Prime+Probe*, *Flush+Reload*, *Flush+Flush*, and *Evict+Time*. *Prime+Probe* is the first and common technique for profiling the victim VM through a shared cache. The attacker needs to observe which cache lines are accessed by the victim during the execution of sensitive operations using timing information (provided by instructions such as `rdtsc`). The *Flush+Reload* technique has more resolution than other attack techniques because of leveraging memory deduplication feature and inclusive caches.

IV. Design And Implementation

The main idea behind our detection approach is to analyze periodically the behavior of running VMs, to identify malicious activities which may be detected as a side-channel attack. We need to have several different fine-grained (provided by the hardware) information gathered along the execution of VMs. To date, there are two hardware sources providing high resolution cache-related information on running VMs that might be utilized for analysis purposes: Intel CMT and HPCs.

A. Threat Model

We assume a IaaS cloud and that the underlying CPU is based on x86 architecture equipped with Intel CMT. We also assume that victim and attacker VMs are pinned to different cores of processor. It means only the LLC is shared between two VMs. Thus private resources of a core such as L1 cache and the unified L2 cache are **not** shared between two VMs. The detection module aims to detect side-channel attacks conducted between two VMs through the LLC. We further assume that the attacker has full control on his VM and is capable of exploiting the LLC through certain low-level instructions such as `rdtsc` and `clflush` which are accessible to unprivileged users in x86 architecture. Design and implementation objectives for our detection are as follows: (1) it should detect side-channel attacks between two VMs through the LLC (2) it should not impose modifications to the hypervisor to be generic and compatible with other hypervisors (3) it should induce small performance overhead.

B. Components of the Detection Module

In our design, we take advantage of following techniques/technologies to detect the attacks:

a) Intel Cache Monitoring Technology (CMT): Intel has introduced two features in processors to improve cache utilization: *Cache Allocation Technology (CAT)* and *Cache Monitoring Technology (CMT)*. CAT is basically intended to provide Quality of Service (QoS) to running workloads (*i.e.*, VMs, applications) in the allocation of LLC. CMT aims for monitoring the use of shared resources. Recent processors have several cores allowing to run simultaneously diverse workloads on CPU cores. This capability may potentially cause run-time resource contention on shared processor resources such as the LLC. Consequently, such resource contention may result in reduced performance of the overall system. This issue gets more importance in real-time and virtualized systems, where performance is critical. For instance, in a virtualized environment such as the cloud, CMT allows monitoring LLC occupancy of VMs running on the same physical host in a data center, finding aggressive VMs and then reacting to such events by applying resource usage limitation through CAT. Regarding cache-based side-channel attacks, as the attacker VM evicts several cache lines many times from the cache, this aggressive behavior can be detected by CMT. Hence cache occupancy is provided at the byte-level granularity. Therefore, we can use the cache occupancy of a VM as an effective information in the detection of cache-based side-channel attacks through the LLC. In our detection module, we thus leverage CMT as a means to provide useful information to detect these attacks targeting the LLC. To date, our detection approach is the first to leverage Intel CMT to detect side-channel attacks.

Name	Event Description
`LLC-misses`	Last-level cache misses
`LLC-references`	Last-level cache references
`LLC-r-accesses`	Last-level cache read accesses
`LLC-w-accesses`	Last-level cache write accesses
`L1-dcache-misses`	L1-data cache misses
`L1-icache-misses`	L1-instruction cache misses
`iTLB-r-accesses`	Instruction TLB read accesses
`iTLB-cache-misses`	Instruction TLB read misses

Table I: Specific hardware performance events related to cache

b) Hardware Performance Counters (HPCs): These are special platform hardware registers used to store statistics about various CPU events such as *clock cycles* and *context-switches*. Those events are widely used to profile a program's behavior to analyze and optimize it. Collected performance data may also be analyzed to detect any situation of performance degradation. For instance, in the Intel micro-architecture, the events are provided by *Processor Monitoring Units (PMU)* and are accessible through the `perf_events` subsystem provided by the Linux kernel. For system security, HPCs are widely used in the detection of different attacks such as Rowhammer [12] and cache-based side-channels [4]. Our detection module takes advantage of HPCs to detect malicious VMs in a virtualized environment. Table I gives an overview of several hardware cache events that might be utilized in the detection of cache-based side-channel attacks in the system. For real-time attack detection, the detection module should simultaneously get access to events. Therefore, we select the most effective events among those available for the attack detection. The detection module aims to detect any malicious behavior induced by a virtual machine, in the cache. We thus focus on the events which are necessarily related to the LLC. Previous works [4] [6] [13] proposed to focus on `LLC-misses` and `LLC-references` as the important events to monitor. After our observation during experimentation, in our approach, we use `LLC-misses`, `LLC-references`, `iTLB-cache-misses`, and `iTLB-r-accesses` as the most effective performance counters to detect the attack.

c) Anomaly Detection: As an application class of Machine Learning, the Gaussian anomaly detection approach detects data points in a data set which do not fit well with

the rest of the data set. This approach is widely used in monitoring and diagnostic applications. Given a data set X with m data points. Each data point $x^{(i)}$ in the X is represented by n features. Among data points in the data set, there might be some anomalous data points to detect. The first step in the Gaussian method is to fit a Gaussian distribution to data points of data set. We thus calculate the mean (μ_j) and the variance ($\sigma^2{}_j$) of each feature $j = 1, \ldots, n$ for $x_j^{(1)}, \ldots, x_j^{(m)}$, to estimate Gaussian distribution of features.

Finally, according to a defined threshold value based on a cross-validation, we can find anomalous data points through calculating the probability of each data point ($p(x^{(i)})$) and finding those with lower probability value than threshold. To find an optimal threshold, we use a cross-validation set and calculate the F_1-score [14] for each example in the set. We calculate the F_1-score using the precision (*prec*) and recall (*rec*), where *tp* is the number of *true positives*, *fp* is the number of *false positives*, and *fn* is the number of *false negatives*.

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (1)$$

$$F_1 = 2.\frac{prec.rec}{prec + rec} \quad prec = \frac{tp}{tp+fp} \quad rec = \frac{tp}{tp+fn} \quad (2)$$

In our detection approach, each virtual machine is represented as a data point in the data set, and performance information such as `LLC-misses` and `llc-occupancy` are the features of each data point. We apply Gaussian anomaly detection on the data set to detect any anomalous VM. As there is a limited number of running VMs on a physical machine, the data set size remains small and will efficiently reduce the computation cost of the anomaly detection module.

C. Implementation

Fig. 1 gives an overview of the architecture of the detection module. Our detection prototype is composed of three threads: The main thread is the core of module and probes periodically the system to collect statistics information (*i.e.*, `LLC-occupancy`, `LLC-misses`, etc.) of all running VMs. A second thread scans the host machine to provide a live list of running VMs. A detection thread applies Gaussian anomaly detection on collected information to identify any malicious behavior. The detection module is implemented as a user space program written in C, and operates on the KVM/QEMU hypervisor.

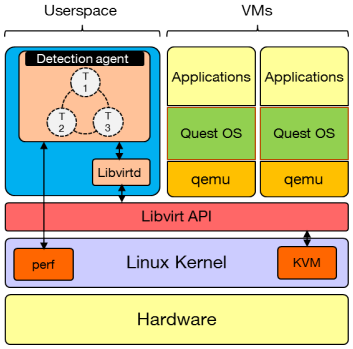


Figure 1: Detection module architecture

V. Evaluation

To validate our detection approach, we ran 6 VMs on the following experimental platform: `Intel(R) Xeon CPU D-1541 2.10GHz, 16GB RAM, L1 32KB, L2 256KB, LLC 12MB`.

A. Evaluation Process

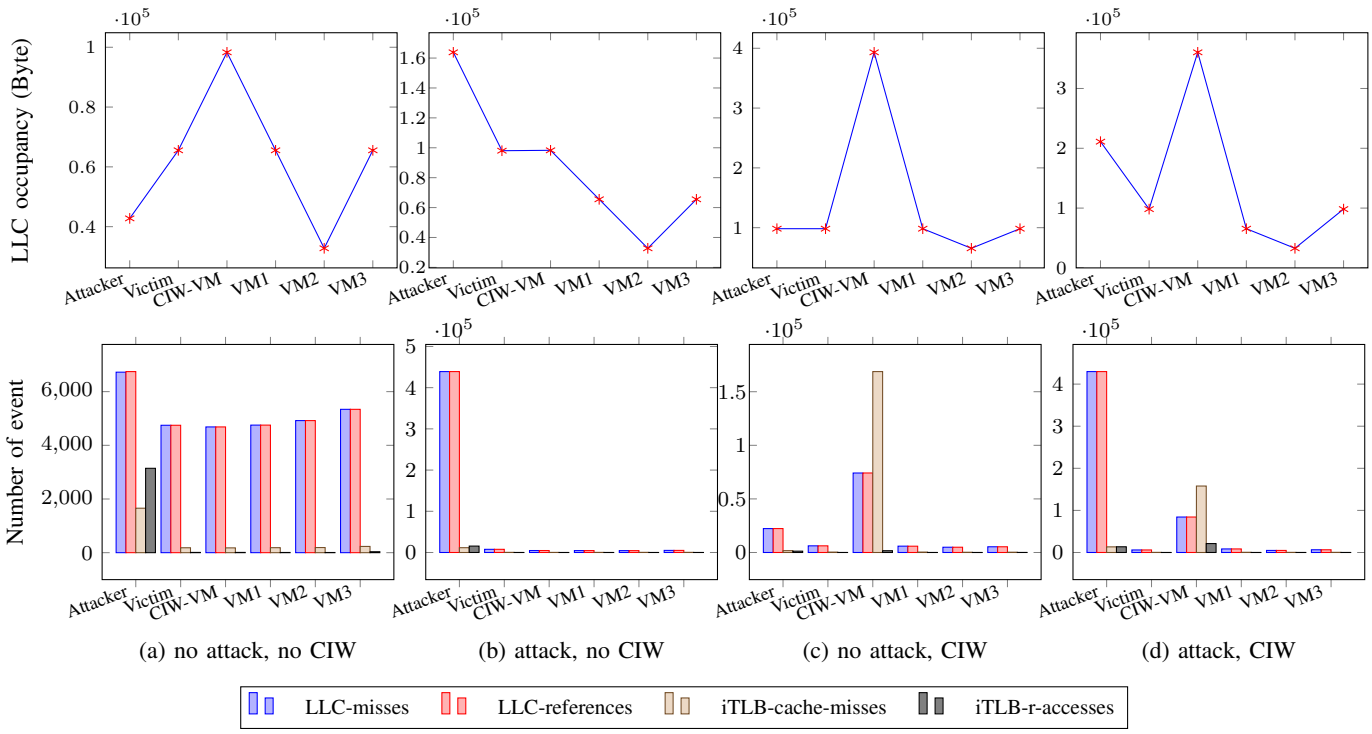
1) Attack Scenario: We use the code of Prime+Probe attack provided by [13] [15]. We consider one attacker VM and one victim VM. We assume that other VMs perform normal operations and among them there is a VM with a compute-intensive workload. We need a *compute-intensive workload (CIW)* to determine if the detection module is able to distinguish between a malicious VM and a honest compute-intensive VM (CIW-VM).

2) Experimental Scenarios: In our evaluation, we experiment different scenarios for the virtualized platform.

a) ***No-attack, No-CIW****:* In this scenario, two VMs are in idle status *i.e.*, the attacker does not perform any side-channel attack and there is not any compute-intensive workload. As shown in Fig. 2.a, almost all VMs have roughly the same values for the perf events. We can also see that the attacker VM, victim VM, and other VMs have normal values for `llc-occupancy` (comparing to the attack situation). All VMs have the same `LLC-misses` and `iTLB-cache-misses` rates.

b) ***Attack, No-CIW****:* In this scenario, the attacker performs the side-channel attack and there is not any running compute-intensive workload. As we can see on Fig. 2.b, the attacker has a lot of `LLC-misses` and occupies more LLC space than other VMs. The attacker also encounters fewer `iTLB-cache-misses` than other VMs. Indeed, as seen before, the attacker VM re-accesses the same memory lines several thousands times during the attack period. Consequently, read access to TLB cache results in cache hit.

c) ***No-attack, CIW****:* In this scenario, the attacker VM does not perform any attack. Among other VMs, one VM performs compute-intensive operations. As shown in Fig. 2.c, the compute-intensive VM has very different perf and `llc-occupancy` values. This VM has a higher value for `llc-occupancy` than other VMs and a large number of `LLC-misses`. This behavior is not abnormal for a compute-intensive workload. However, it has larger `iTLB-cache-misses` (also iTLB-cache-misses ratio) than the attacker VM in the attack scenario. In fact, the compute-intensive VM has a larger instruction footprint that results in increasing instruction cache misses. This difference may be considered as an important feature to distinguish between attacker and compute-intensive VMs.



(a) no attack, no CIW (b) attack, no CIW (c) no attack, CIW (d) attack, CIW

Figure 2: Perf information of VMs in different scenarios

*d) **Attack, CIW**:* In this scenario, the attacker VM performs a side-channel attack. Another VM performs compute-intensive operations as well. As we can see on Fig. 2d, the attacker and compute-intensive VMs have a larger number of `LLC-misses` and `llc-occupancy` than other VMs. Although, two VMs have a large value of `LLC-misses`, the attacker VM has a higher distinct value than the compute-intensive VM. On the other hand, the compute-intensive VM has a larger value for `llc-occupancy` than the attacker VM. We also observe that the attacker VM encounters fewer `iTLB-cache-misses` than the compute-intensive VM. This significant difference may be used to distinguish the attacker VM from the compute-intensive VM. This scenario may thus potentially generate *false positives* alerts.

3) Origin of High Perf Values: To justify that the high number of *e.g.*, `cache-misses` measured for attacker VM, are generated by the spy program running in the attacker VM, we use `perf-kvm`. This is another extension of `perf` that allows tracing/measuring a guest OS running on top of KVM. As shown in Table. II, `clear_page_c_e` is the first kernel function that generates a lot of `LLC-misses` during the execution of spy program. After verification on the guest OS, we observe that this function is called many times during the spy program execution. This shows that a process (in the attacker VM) is spending its time to clear memory. Such a situation particularly occurs during a cache-based side-channel attack, especially in the first phase of the attack *i.e.*, evicting memory lines from cache. We also verified the rate of `LLC-misses` when the spy program was not executing and it was $\backsimeq 1.5\%$ (see Fig. 3). We can thus confirm that the high

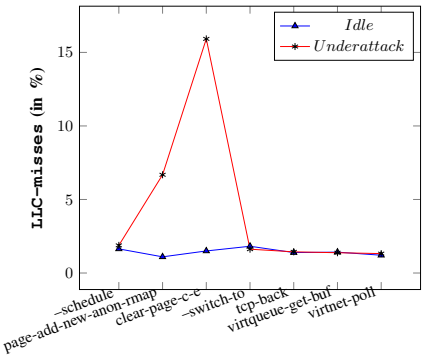


Figure 3: LLC cache misses vs. system functions for attacker VM

rate of `LLC-misses` is induced by the spy program.

Additionally, as shown in Fig. 4, during a side-channel attack, the rate of `llc-occupancy` increases abnormally because of evicting cache lines from LLC by the spy program, early in the first phase ($t = 3s$) of the attack. This event is more notable in the case of the attacks based on Prime+Probe or Evict+Reload techniques when the attacker uses an eviction set to evict totally a cache set.

B. Evaluation Results

In this Section, we present detection accuracy and performance overhead results.

1) Detection Accuracy: To evaluate the accuracy of cache-based side-channel attack detection, we measure the true positive rate and false positive rate generated by our detection module in the different scenarios described in

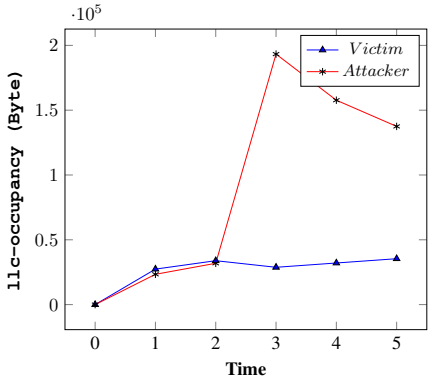


Figure 4: LLC occupancy of attacker and victim VMs during the attack

Overhead	shared Object	Symbol
15.92%	[guest.kernel]	[g] clear_page_c_e
6.68%	[guest.kernel]	[g] page_add_new_anon_rmap
1.88%	[guest.kernel]	[g] __schedule
1.62%	[guest.kernel]	[g] __switch_to
1.44%	[guest.kernel]	[g] tcp_back
1.38%	[virtio_ring]	[g] virtqueue_get_buf
1.32%	[virtio_net]	[g] virtnet_poll

Table II: Output of perf-kvm for the attacker VM: top functions generating `cache misses` in the attacker VM

Section V. Results show that the detection module works very well in experimental scenarios **(a)** and **(b)**, and detects malicious VM correctly. However, our detection module may generate false positives in scenario **(c)** and especially in scenario **(d)** *i.e.*, when a compute-intensive workload is present. In Fig. 2.c and Fig. 2.d, we also notice that there is a clear difference between the `iTLB-cache-misses` rate ($\frac{\texttt{iTLB-cache-misses}}{\texttt{ITLB-r-accesses}}$) of the compute-intensive VM and the attacker VM. Therefore, this can be used as an important feature for distinguishing between the attacker VM and compute-intensive VM. We also observe that a malicious VM has a lower `iTLB-cache-misses` rate (*e.g.*, 1.2%) than a benign VM (*e.g.*, 30%). Consequently, by using this feature, we can distinguish compute-intensive workload and malicious VMs, that results in a false positive reduction. We also carried out other experimentation to assess the efficiency of Intel CMT in detection accuracy on two different scenarios; when Intel CMT is enabled, and when it is not. We obtained 0.67 as $F_1 - score$ when Intel CMT was enabled than 0.5 without Intel CMT. It means, we effectively obtained more precision in our detection approach through leveraging Intel CMT (Fig. 5).

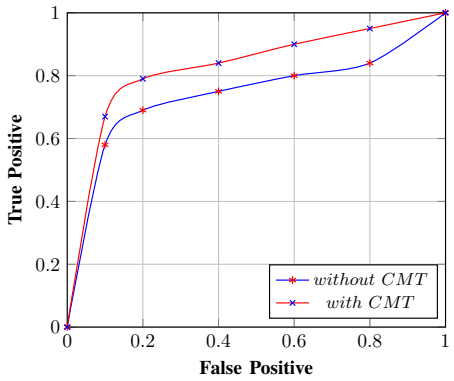


Figure 5: Roc curve without & with Intel CMT

2) Performance Overhead: The detection module imposes about 2% performance overhead to the hypervisor, which is negligible. However, we measured this value for 6 running VMs on the platform. This value may increase when extending the number of running VMs or during a long period of probing to collect perf information.

VI. Conclusion

In this paper, we presented a new approach of cross-VM side-channel attack detection leveraging Intel Cache Monitoring Technology (CMT), Hardware Performance Counters (HPCs), and the Gaussian anomaly detection method to detect malicious VMs in IaaS platforms. This work has demonstrated the promising performance of Intel CMT and HPCs in the detection of side-channel attacks in a virtualized environment. As future work, we will explore improving our approach, also comparing it with emerging approaches for side-channel attacks detection.

References

[1] M.-M. Bazm, M. Lacoste, M. Südholt, and J.-M. Menaud, "Side-Channels Beyond the Cloud Edge : New Isolation Threats and Solutions," in *1st IEEE International Conference on Cyber Security in Networking (CSNet) 2017*, (Rio de Janeiro, Brazil), Oct. 2017.

[2] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$a: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes," in *2015 IEEE Symposium on Security and Privacy*, pp. 591–604, IEEE, 2015.

[3] Y. Yarom and K. Falkner, "Flush+Reload: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *23rd USENIX Security Symposium*, pp. 719–732, 2014.

[4] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016.

[5] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 118–140, Springer, 2016.

[6] M. Payer, "Hexpads: a platform to detect stealth attacks," in *International Symposium on Engineering Secure Software and Systems*, pp. 138–154, Springer, 2016.

[7] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[8] M. E. Thomadakis, "The architecture of the nehalem processor and nehalem-ep smp platforms," *Resource*, vol. 3, p. 2, 2011.

[9] O. Acıiçmez and Ç. K. Koç, "Trace-driven cache attacks on aes (short paper)," in *International Conference on Information and Communications Security*, pp. 112–121, Springer, 2006.

[10] J.-F. Gallais, I. Kizhvatov, and M. Tunstall, "Improved trace-driven cache-collision attacks against embedded aes implementations," in *International Workshop on Information Security Applications*, pp. 243–257, Springer, 2010.

[11] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ Flush: A Fast and Stealthy Cache Attack," *arXiv preprint 1511.04594*, 2015.

[12] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, 2015.

[13] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: a fast and stealthy cache attack," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 279–299, Springer, 2016.

[14] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," 2011.

[15] "Flush+Flush side-channel attack github repository, https://github.com/iaik/flush_flush."