Side-Channels Beyond the Cloud Edge : New Isolation Threats and Solutions

Mohammad-Mahdi Bazm*, Marc Lacoste*, Mario Südholt†, Jean-Marc Menaud†

*Orange Labs, France

†IMT Atlantique, France

***Abstract*—Fog and edge computing leverage resources of end users and edge devices rather than centralized clouds. Isolation is a core security challenge for such paradigms: just like traditional clouds, fog and edge infrastructures are based on virtualization to share physical resources among several self-contained execution environments like virtual machines and containers. Yet, isolation may be threatened due to side-channels, created by the virtualization layer or due to the sharing of physical resources like the processor. Side-channel attacks (SCAs) exploit and use such leaky channels to obtain sensitive data. This paper aims to clarify the nature of this threat for fog and edge infrastructures. Current SCAs are local and exploit isolation challenges of virtualized environments to retrieve sensitive information. We introduce a new concept of distributed side-channel attack (DSCA) that is based on coordinating local attack techniques. We explore how such attacks can threaten isolation of any virtualized environments such as fog and edge computing. Finally, we study a set of different applicable countermeasures for attack mitigation.**

***Index Terms*—cloud security, isolation, side-channel attacks, distributed side-channel attacks, moving target defense, decentralized cloud infrastructures.**

I. Introduction

Decentralized cloud paradigms have been getting a lot of attention lately, notably because of the emergence of applications involving the Internet of Things (IoT) that generate high volumes of data whose processing requires significant resources. The corresponding novel architectures [1], [2] extend traditional cloud services beyond data center, migrating virtualization resources to the network edge into different types of micro data centers. Benefits include improved response time and cloud efficiency through geo-location-awareness, *e.g.*, load-balancing between core data centers and the edge, thus enabling multi-level context-aware adaptations.

Both traditional and decentralized clouds make pervasive use of *virtualization* for dynamic resource allocation, consolidation and service provisioning, reaching an extensive level of physical resource sharing between different operating systems. Virtualization technology has also evolved from virtual machines towards more lightweight solutions such as containers [3] or unikernels [4], concepts that are better manageable for users, and increase flexibility as well as reliability. Such technologies may be deployed in all elements of the infrastructure, especially on edge computing devices. They are, however, highly heterogeneous between cloud-facing and embedded-facing parts of the infrastructure.

Such features raise several security and privacy issues, some of which are also well-known for traditional virtualized systems. The most important new challenge is to guarantee strong isolation between virtualized execution environments. Resource isolation between processes or virtual machines is traditionally implemented by OS/hypervisor components using now well-established software techniques, such as access control to limit sharing of key resources. Unfortunately, in a virtualized environment the impact of co-residency on isolation reaches much deeper into the hardware: physical resources are often shared between independent customers at a fine-grained level including in terms of multi-tasking through the virtualization layer. This means software security mechanisms can be bypassed, threatening the overall security of the virtualized infrastructure and software systems using it.

Side-channels attacks (SCA) constitute a class of security threats to isolation that is particularly relevant for the emerging "beyond the edge of the cloud" virtualized systems, directly leveraging weaknesses of current virtualization technology. An SCA uses a hidden channel that leaks information on the execution time or on cache access patterns of computations such as cryptographic operations. Such channels can be established based on techniques used in the virtualization layer, the software implementation of cryptographic algorithms, or on micro-architectural vulnerabilities in processors. SCAs can be applied to a wide range of computing devices and can target particularly cryptographic systems such as AES without need for any special privileged access. For mitigation, countermeasures can be applied at the application, OS/hypervisor, and hardware levels. Each level of mitigation comes with its own challenges in terms of characteristics such as performance and compatibility with legacy platforms.

Despite having been already well investigated separately for cloud and for embedded systems, SCAs are still far from understood overall for decentralized virtualized systems. For instance, SCAs may be performed in a distributed and coordinated manner to attack several targets in a virtualized environment such as a fog infrastructure, giving rise to new forms of attacks: *Distributed SCAs* (DSCAs). However, no definition and systematic study of such DSCAs has been put forward yet.

This paper provides a first definition of DSCAs. It also aims to clarify the nature of this threat in terms of challenges and types of attacks, and presents possible counter-measures for mitigation. We provide the following contributions: (1) we identify isolation challenges related to virtualization, and show how they have been exploited in already known SCAs; (2) we introduce the concept of distributed SCAs which may exploit targets using several coordinated local SCAs; (3) we present three possible levels for mitigation, giving a brief overview of different proposed approaches.

This paper is organized as follows. In Section II, we give an overview of isolation challenges related to virtualization. In Section III, we define SCAs, and survey SCA types in virtualized environments, along with some typical attack and mitigation techniques. Then, we introduce DSCAs which are performed in a distributed manner by coordinating the attack techniques. In Section IV, we discuss possible approaches to mitigating attacks at different levels of application. Then, in Section V, we give an outlook of the approach Moving Target Defense (MTD) in dencetralized clouds infrastructures. Finally, we conclude in Section VI with some perspectives.

II. Virtualization: Breaking Isolation?

Virtualization is in some respects strongly at odds with isolation, both at the hardware level and at the software level. On the one hand, the virtualization layer enables several virtual machines to run concurrently on a physical machine. VMs are distributed between processor physical cores, where they are scheduled and run. The low-level physical resources of the host are broadly shared between VMs through the virtualization layer, which has a strong security impact on running VMs. On the other hand, a number of techniques in the virtualization layer make the use of some physical resources (the main memory and caches, in particular) more efficient. Yet, they may also strongly threaten isolation. These techniques may have an impact on isolation directly or indirectly (Table. I).

A. Hardware Isolation Challenges

The processor is the primary resource shared among VMs/containers in different ways. Several features of current micro-architectures may cause isolation violations.

1) Cache Architecture and Multi-Cores: Because of rising performance needs for processing resources, multi-core processors are widely used in virtualized environments. Cache memory is a fast and small amount of memory speeding up access times between the processor and, the main memory.

We may distinguish 3 levels of cache (see Fig. 1). L1 and L2 levels are usually private to each processor core. L3 (Last-Level Cache, LLC) is shared among all cores, having a much larger capacity (MBs) than L1 and L2 (KBs). A cache is organized into *sets* containing several *cache lines*, or data blocks of fixed size which are the units of data transfer between the CPU and the RAM. When the processor needs to access a specific location in the RAM, it checks whether a copy of the data is already present in L1, L2 or LLC. If so (*cache hit*), the CPU performs operations on the cache in a few clock cycles. Otherwise (*cache miss*), the CPU fetches data from the RAM into the cache taking more clock cycles. Such timing differences (or more generally access patterns) may be observed by an attacker. The cache hierarchy, especially the shared LLC, can thus be exploited as a channel leaking information on running processes *e.g.*, between different cores.

Layer	Mechanism / Technique	Direct	Indirect
Hardware	Cache architecture in multi-cores CPU	✓	
	Exclusive/inclusive cache		✓
	Simultaneous Multi-Threading (SMT)	✓	
Virtualization	Memory deduplication	✓	
	Large-Page memory managment		✓
	Non-privileged access to hardware instructions		✓

Table I: Impact of the mechanisms/techniques on the isolation in virtualized environments. If a technique is used as helper in attacks steps, it has an indirect impact on the isolation.

2) Exclusive/Inclusive Caches: Two types of cache architecture can be distinguished in modern processors. In *exclusive* caches, each cache line is not replicated in the different cache levels, thus obtaining higher capacity. In the case of *inclusive* cache architectures, all cache lines at a given cache level (*e.g.*, L1 and L2) are also available at the lower-levels of cache (*e.g.*, LLC). Inclusive architectures provide less cache capacity but offer higher performance. Unfortunately, they are widely exploited in cache-based SCAs: when a cache line is evicted from the LLC, it is also evicted from the L2 and L1 caches. An attacker can then simply evict cache lines from the LLC, and measure fetching time of a memory line from RAM to obtain the victim's cache access pattern to perform an SCA.

3) Simultaneous Multi-Threading (SMT): Simultaneous multi-threading such as Intel Hyper-Threading (HT), enables multiple threads to be run in parallel on a single core of a processor in order to improve execution speed. SMT allows sharing of dedicated resources of a core, mainly its L1 cache, between multiple threads running on the same core. This sharing can leak information between threads (of a core) and may be used by a spy thread to gather information from other threads. SMT can potentially ease cache-based SCAs [5], [6].

B. *Virtualization Layer Isolation Challenges*

The virtualization layer provides important isolation services between VMs. Unfortunately, some mechanisms in this layer can also be abused for isolation violations.

1) Memory Deduplication: This mechanism is widely used in OSes and hypervisors to optimize memory usage [7]. Duplicated physical pages are merged, and mapped into virtual address spaces of the processes using them. This technique breaks isolation between two processes that may thus access the same physical memory pages [8].

2) Large-Page Memory Management: Large pages enable more efficient memory management in the hypervisor: access to large pages boosts Translation Lookaside Buffer (TLB) page walks during address translation by reducing the number of page table entries. Unfortunately, this technique has a security impact as in large-page memory management, more bits of a virtual address remain unchanged during the page mapping to a physical address than small-page memory management [9].

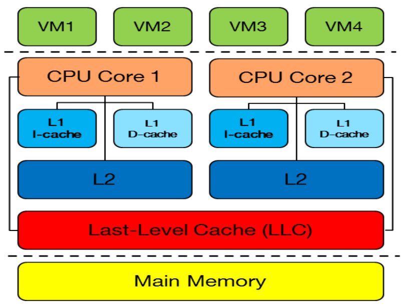


Figure 1: Common cache architecture in virtualized systems.

3) Non-Privileged Access to Hardware Instructions: Many hypervisors do not perform any access control to execute instructions on the host. Almost all SCA techniques exploit specific hardware instructions such as `rdtsc` or `clflush`, respectively to measure time, and to evict memory lines from the cache. Such instructions may be used without any special privileges by an attacker to manipulate the state of the cache.

III. Distributed Side-Channel Attacks in Virtualized Platforms

Recently, the idea of a *distributed timing attack* (DTA) was proposed in the context of network-on-chips (NoC) [10]. This environment is not virtualized, but as we will show, SCAs may be performed in a distributed way on virtualized platforms. In what follows, we first present SCAs and attack techniques through an example. We then introduce a new concept of SCA, distributed side-channel attacks, that combine such techniques.

A. *Side-Channels*

A *side-channel* is a hidden information channel that differs from traditional "main" channels (*e.g.*, files, network) in that security violations may *not* be prevented by placing protection mechanisms directly around data. They usually come from indirect effects of computations and are thus much more elusive. Such channels may be created through hardware, *e.g.*, in some components of the micro-architecture, such as the processor cache hierarchy. They may also be created through software, *e.g.*, by techniques in the virtualization layer to optimize resource usage such as memory deduplication. Table. II shows different leaky channels in the virtualization technologies.

B. Side-Channel Attacks

A *side-channel attack* (SCA) exploits a side-channel to obtain important information. SCAs may be classified according to the type of exploited channel. *Timing attacks* and *cache-based attacks* are two main classes of SCAs, where the processor cache memory is often exploited by adversaries to obtain sensitive information such as cryptographic secret keys.

In timing attacks, *time* is the leaky channel used to retrieve sensitive information such as secret keys. The attacker attempts to analyze the time taken to execute cryptographic algorithms. Cache-based timing attacks [11], [12] leverage cache memory as a means to attack a cryptosystem. As an example, consider an attack implemented on DES encryption by inferring *S-box* inputs [13]: the encryption time is measured for different plaintexts. The execution time of an encryption algorithm may also vary due to its memory activity. Cache misses notably increase the execution time of an application at run-time.

In cache-based side-channel attacks, the shared cache itself leaks information. Two broad classes of attacks should be distinguished. In *access-driven attacks* [14], [15], the attacker tries to find any relation between an encryption process and accessed cache lines to exploit the pattern of cache accesses of the victim. To derive a profile of cache activities, the attacker manipulates the cache by evicting memory lines of the victim process. In *trace-driven attacks* [16], [17], the attacker observes cache activities *i.e.*, memory lines which are accessed by an encryption process during its execution in order to obtain a sequence of cache hits and cache misses. For example, observing which memory accesses to a lookup table lead to cache hits allows finding indices of entries in the table.

Different techniques have been proposed to perform a cache-based SCA. For instance: *Prime+Probe* [18], *Flush+Reload* [15], *Flush+Flush* [14] and *Evict+Time* [19]. *Prime+Probe* (see Fig. 2) is the most common technique to obtain a profile of a victim VM or process through a shared cache. The attacker needs to monitor which cache lines are accessed by the victim during the execution of sensitive operations such as encryption using timing information (usually provided by hardware instructions such as `rdtsc`). The attack is performed in three main steps: (1) the attacker fills one or several selected cache sets – he selects more likely cache sets to be accessed by the victim; (2) the attacker waits for a defined period while the cache is used by the victim process; (3) the attacker refills memory sets with the same data used in step (1) to verify which cache lines are accessed by the victim. Indeed, if a memory line was flushed by the victim from the cache, it takes more CPU cycles to fetch the line from 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.

Instance Type	**Leaky Channel**		**Exploited Architecture**
	Software	**Hardware**	
VM	*Page deduplication*	*LLC*	*Multi-cores*
Linux Container	*Page sharing*	*LLC, L1*	*Multi-cores, SMT*

Table II: Side-channels in different virtualization technologies.

To defend against such attacks, a distributed computing technique such as *Secure Multi-Party Computation* [20] may be considered as a mitigation approach to SCAs by dividing computations among a certain number of computing resources, and securing them by *e.g.*, a

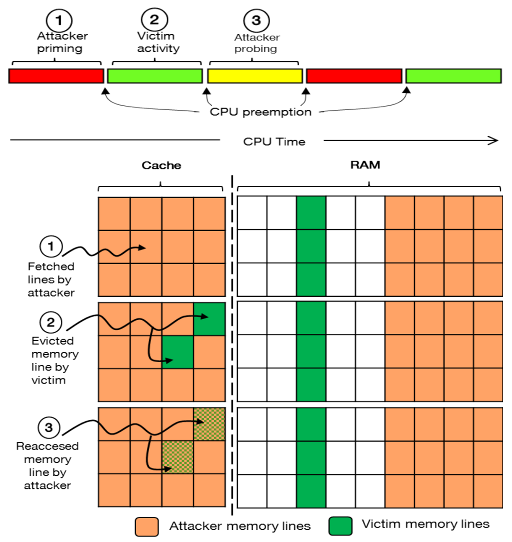


Figure 2: Prime+Probe side-channel attack: main three steps of the technique.

shared-key cryptographic scheme. Here, a *Distributed Side-Channel Attack (DSCA)* comes into play to retrieve sub-keys used in such distributed system.

C. Distributed Side-Channel Attacks (DSCA)

A DSCA aims to retrieve sensitive information, or a subset of it, from a number of computing resources of a distributed system, where each computing resource performs processing of a part of the overall system activity. The retrieved information may, for instance, be a set of encryption keys that can be exploited to compromise the functionality of the whole distributed system (see Fig. 3). In the following, we provide the (first) definition of distributed side-channel attacks applicable to such a class of key recovery schemes.

Definition. A *distributed side-channel attack* (DSCA) over a set M_{vic} of VMs running in a distributed system S is defined as the tuple $DSCA = (S, M_{vic}, D, K, M_{mal}, A, CP, EP)$ where:

- S is a distributed system;
- M_{vic} are the VMs/containers that are targeted by the attack;
- D is the distributed dataset to be compromised (partially or totally);
- K is the distributed cryptographic secret to retrieve;
- M_{mal} are malicious VMs, co-located with the victim VMs;
- A is a set of local SCA techniques;
- CP is a protocol to coordinate the attacker VMs in M_{mal};
- EP is a protocol to exfiltrate data.

We consider $D = d_1, ..., d_n$ a dataset to be processed by the distributed system $S = s_1, ..., s_n$ implemented on a set of virtual machines $M_{vic} = m_{vic_1}, ..., m_{vic_n}$ on a virtualized platform. Each component s_i of S processes data d_i locally and runs in its own virtual machine m_{vic_i}. We assume data protection is based on a shared cryptographic scheme with secret-key $K = k_1, ..., k_n$: each sub-key k_i is used by component s_i to encrypt/decrypt data d_i. To perform the distributed attack, the adversary sets up a number of malicious VMs (at least equal to the number of M_{vic}) $M_{mal} = m_{mal_1}, ..., m_{mal_n}$, co-located with the victim VMs M_{vic}. The adversary also masters a set $A = a_1, ..., a_m$ of side-channel attack techniques *i.e.*, Flush+Reload, etc.

The objective of a DSCA is to first attack in a decentralized manner each component of the system s_i running on m_{vic_i} through m_{mal_i} running local attack technique a_i to retrieve the sub-keys k_i. The adversary may then coordinate attacking VMs to retrieve the overall secret K of the distributed system. Such synchronization may be performed using a coordination protocol CP. A protocol EP may be used to control attacking VMs remotely, and to send collected information to a remote server to exfiltrate sensitive data.

1) Attack Platform: To perform a DSCA, an adversary needs a set of virtual machines/containers distributed across a virtualized environment such as a cloud, and co-located on the same physical machines as victim VMs/containers (see Fig. 4). Each node of the set may then apply on its co-located victim VM/container a side-channel attack using one of the techniques described in Section III. After a number of attacking nodes obtain their target sub-keys, results may be sent using an exfiltration protocol to a centralized attack server that will rebuild the overall secret key from received results.

2) Attack Prerequisites: Performing any side-channel attack requires a number of preparation steps which are detailed in the following:

a) Finding Physical Hosts Running Victim Instances: To perform any co-resident attacks including side-channel ones, the attacker needs several VM launching strategies to achieve co-residency with the victim instance. A pre-condition for any side-channel attack is that the malicious VMs/containers reside on the same physical host as victim VMs/containers, as side-channel attacks are performed locally. The first and main step is thus to find the location of physical hosts running victim VMs/containers. There are several placement variables

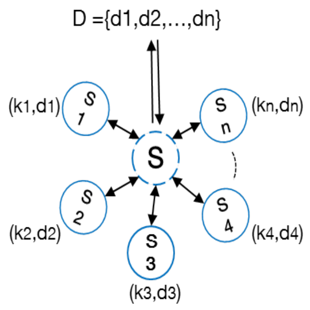


Figure 3: Overall view of the distributed system. Each component of the system has a sub-key to protect data.

such as *datacenter region*, *time interval*, and *instance type* which are important in achieving co-residency. Although, these variables are different in a IaaS cloud (VMs) than PaaS clouds (containers). For instance, the *application type* can be considered as an effective factor in the placement strategy in container-based clouds [21]. Given $P(m_{mal_i})$ the chances of instance m_{mal_i} to be co-resident with instance victim m_{vic_i}. The value of P will be raised by increasing the number of launched attack instances. To make sure that both attacker and victim VMs achieve co-resident placement, the adversary can perform co-residency detection techniques such as *network probing*. The attacker can also use data mining techniques to detect the type and characteristics of a running application in the victim VM by analyzing interferences introduced into different cache levels, to increase the accuracy of co-residency detection [22]. Co-residency detection is easier in container-based environments than in VM-based ones because of more existing logical side-channels in such environment [23].

On the other hand, this preparation phase of attacks *i.e.*, achieving co-residency gets more importance in the case of decentralized cloud infrastructures composed of several self-managed data centers with different placement and security policies [23]. More precisely, if victim instances are spread in different sites, achieving co-residency comes more complicated to handle because the attacker needs to launch a large number of malicious instances and plays several different scenarios to bypass placement algorithms (*i.e.*, different cloud providers result in playing different VM launching scenarios).

b) Synchronization of Attacker and Victim Instances: The spy application running one of the local attack techniques must be well synchronized with the victim application to trace its activities in the cache.

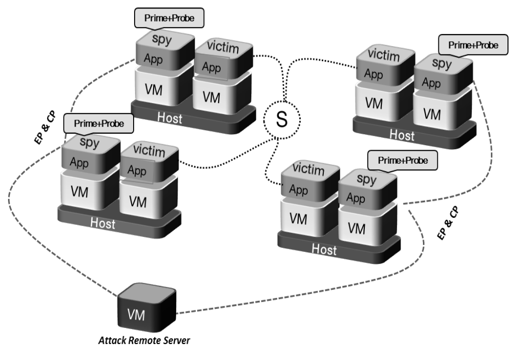


Figure 4: Malicious VMs co-located with victim VMs. The attack coordination is performed by a botnet under adversary control.

Otherwise, the attacker will not be able to profile the cache activity of the victim application. How to perform effectively but stealthy synchronization is very important in the implementation of side-channel attacks.

We consider t_{victim}: start time of a sensitive operation *e.g.*, cryptographic, t_{attack}: start time of spy operation (tracing cache activities of victim) , and t_{wait}: waiting period in the phase 2 of any side-channel attack. In order to detect the first access of sensitive operation in sensitive application running in s_i, t_{victim} must be : $t_{attack} < t_{victim} < t_{attack} + t_{wait}$. The attacker may probe cache periodically to detect any cache access by the sensitive operation. However, getting synchronized with the victim depends on the attacks techniques. For instance, synchronization in Flush+Reload is less difficult because of high resolution of technique that uses page sharing. In a real attack scenario, the spy application (running in malicious VM/container) must be able to recognize the start of the victim application (running in the victim VM/container). Achieving synchronization between two containers running on the same CPU core, in the case of Prime+Probe attack on L1 cache is simpler because of the reduced level of noise in L1.

IV. Countermeasures

Although, DSCAs are performed across the networking infrastructure, relevant countermeasures may already be applied locally on physical machines. However, the design and implementation of a federative mitigation framework leveraging approaches such as *Moving Target Defense* (MTD) [24] looks quite promising in terms of avenue for research. In addition to overcoming the underlying sources of heterogeneity, this type of autonomic security framework may decide that selected countermeasures *i.e.*, dynamic ones may be applied on different hosts, for instance, according to available resources or performance, thus enabling trade-offs between protection and other non-functional properties.

Actually, existing mitigation approaches may be divided into three main broad classes according to the enforcement layer in the infrastructure: enforcement on the *application*, *system*, and *hardware levels*. All approaches impose significant performance overheads. The best counter-measure is the one that offers robust security with minimal performance degradation while being scalable and applicable to existing infrastructures with little modifications. Application-level solutions may be applied to achieve secure-by-design protection of applications running in a virtualized environment such as cloud while they are purely static. Hardware approaches are mainly static and focus on applying security in the design of hardware such as processors. They can influence the design of CPUs such as those from ARM that are widespread in fog/edge computing devices. Research on system (i.e., OS and hypervisor) approaches are currently getting more attention because of their benefits in terms of dynamicity and compatibility with legacy systems. Furthermore, they are applicable both in classic clouds and decentralized ones such as fog computing.

A. Application-Level Approaches

This class of approaches focuses on improving the security of sensitive applications by secure design and implementation. Design-oriented solutions aim to improve side-channel attack resistance of applications. Development-oriented and code optimization solutions propose new programming methods for the secure implementation of cryptographic algorithms.

For timing attacks, information leakage comes from variations in execution time of applications, either due to instructions or operations with variable execution time or to data-dependent branches in the implementation. Using instructions with a constant execution time [25] and eliminating conditional branches and loops dependent on secret input data [26], [27] can help to improve SCA resistance of an application. Code transformation techniques for software protection [28] is another approach to mitigate timing SCAs, notably through compiler-based techniques [29], [30]. Key-dependent control flow and data flow are the main causes of timing variations and correlation between secret data and execution time. Therefore, the compiler may be modified to eliminate branches to cut any such correlation using an *if-conversion* transformation [30]. For cache attacks, *obfuscation* of cache access patterns from secret data is a possible mitigation technique [27], [31]. Unfortunately, the overhead of such techniques remains high.

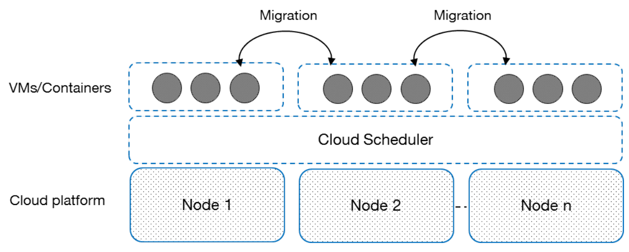
B. OS-Level Approaches

SCA mitigation is also possible at the OS-level through different mechanisms. This level of mitigation might be more interesting than application-layer solutions, as fewer modifications are needed to the infrastructure – although the guest OS needs to be updated. *Time padding*, *cache cleansing*, *dynamic partitioning* [32], *cache locking and multiplexing cache lines* [33] are a number of possible countermeasures. Time-padding ensures that the execution time of a protected application is independent of its input data. Thus, an attacker is unable to observe any variation in the execution time of a function. Cache cleansing flushes the cache to prevent from obtaining cache state information after execution of a function. Cache partitioning, cache locking, and multiplexing of cache lines protect resources of a trusted process from an untrusted process during its execution. *Safe scheduling* is also an efficient technique against timing side-channels [34].

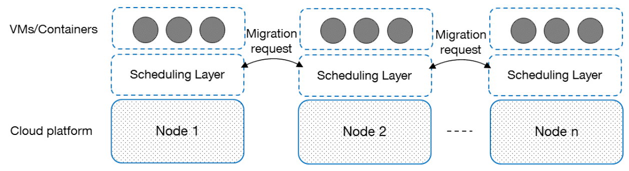
C. Hypervisor-Level Approaches

This level of mitigation could be more effective than the OS layer as cloud providers do not need to modify the guest OS. All OS-level mechanisms may also be implemented at the hypervisor level. Safe scheduling-related approaches include configuring the scheduler to prevent cross-VM SCAs [35]: malicious VMs tracing victim VMs may be interrupted using the `ratelimit_us` parameter in Xen and KVM that determines the minimum run-time of a virtual CPU (VCPU) allocated to a physical CPU. A related approach, applicable in a cloud environment is based on *VM migration* [24]: the scheduler prevents a malicious VM from spying on a victim VM by moving it away to other physical hosts in the cloud.

Page coloring is another technique that assigns a specific color to the memory pages of each VM. It then maps pages with the same color to a fixed set of cache lines, which are only accessible to related VMs. This technique is applicable both statically [36] and dynamically [37]. Static page coloring degrades the



(a) Centralized



(b) Fully-distributed

Figure 5: Fully-distributed scheduling vs centralized one.

performance of a virtualized environment as well as the number of running VMs. In dynamic page coloring, cache protection mechanisms are only active during the execution of sensitive operations to improve the system performance.

Hypervisor may also be used as interceptor between VMs and hardware to control some hardware instructions used in SCAs such as `rdtsc`. *Vattikonda et al.* [38] proposed an approach called *fuzzy time* based on degrading the timing resolution of `rdtsc`: without fine-grained timing information, it becomes hard for the attacker to detect cache hits/misses.

D. Hardware-Level Approaches

Hardware-based mitigation for instance explores integrating cryptographic security in processor design. Solutions also include enhancing cache architectures with new security mechanisms such as *locking of cache lines* [39], *random permutation* techniques [39], and *cache partitioning* [40] to provide strong isolation between processing units *i.e.*, threads and VMs. Recently, Intel introduced a new technology called *Cache Allocation Technology (CAT)* [41] in its processors to improve performance of latency-sensitive applications by guaranteeing cache capacity to priority applications. This technology enables dynamic LLC partitioning between processor cores, from OS/hypervisor layer. Unfortunately, while very efficient and providing strong isolation, hardware mitigation approaches suffer from compatibility issues with mainstream platforms.

E. Moving Target Defense Approach Using Cloud Scheduler

Moving Target Defense (MTD) [42] is based on the changing of the system configurations using techniques such as randomization, in order to make attack surface dynamic and consequently harder to exploit by attackers. This approach may be applied to mitigate DDoS attacks [43] or co-residency attacks such as side-channel ones [44]. To perform any side-channel attacks, an attacker must co-localize a malicious instance *e.g.*, an attacker VM with the victim VM, by reverse engineering of the VM placement algorithm in IaaS cloud. The MTD approach aims to leverage the cloud scheduler in order to migrate a malicious/victim virtualized instance to another host in the cloud. Generally, there are two classes of VM placement algorithms: *centralized* and *fully-distributed* (see Fig. 5). Centralized algorithms [45] may be leveraged to mitigate side-channel attacks in a uniform cloud *i.e.*, a cloud operated by a service provider such as Amazon, and fully-distributed algorithms [46] for multi-cloud and decentralized cloud infrastructures such as Fog computing. There are different types of placement algorithm in cloud such as *Round Robin, Greedy, Genetic algorithms*. Each algorithm tries to guarantee some particular properties such as *load balancing among* *servers, minimum number of active hosts, least response time,* to avoid SLA violation in the cloud platform. Furthermore, a cloud scheduler takes two placement decisions [47]: *Instance initialization* (when an instance is created, the scheduler decides to allocate the instance to a host in the infrastructure, according to the instance placement policy), and *Instance migration* (when an instance is migrated between hosts in infrastructure to guarantee SLA). For instance, in side-channel attacks, the instance initialization decision plays a preventive role in co-residency attacks by avoiding a malicious instance to be co-localized with a targeted victim instance on the same physical machine. This phase of VM placement algorithm is undocumented for security reasons by cloud service providers. The migration decision may be leveraged to mitigate co-residency attacks such as side-channel ones when the attack is detected. Thus, this part of the placement algorithm gets attention in a moving target defense approach. Such mitigation approach may be extended by a detection module to increase efficiency. Thus, an MTD approach may be defined as following:

Definition. A *MTD* mitigation approach over a set N of nodes that run a set of VMs/containers is defined as the tuple $MTD = (CI, N, PLC, M_{vic}, M_{mal}, DTC, A)$ where:

- CI is the cloud infrastructure;
- N are nodes/hosts in the cloud infrastructure;
- PLC is a cloud placement algorithm;
- M_{vic} are the VMs/containers that are targeted by the attack;
- M_{mal} are malicious VM(s)/container(s), co-located with the victim VM(s)/container(s);
- DTC is a detection technique;
- A is a side-channel attack.

We consider a cloud infrastructure CI composed of j nodes $N = N_1, N_2, ..., N_j$, which are distributed in different sites *i.e.*, large or small data centers. In such infrastructure, as a preventive countermeasure, the goal of the initial step of the placement algorithm is to prevent m_{mal_i} and m_{vic_i} to be co-localized on node N_i to avoid meeting any co-residency conditions to perform side-channel attack A between m_{mal_i} and m_{vic_i}. As a reactive countermeasure, the MTD approach aims to react to the detection of side-channel attack A between m_{mal_i} and m_{vic_i} which are co-resident in the same node N_i, by migrating m_{mal_i} to another node N_k in the infrastructure.

V. Discussion

Either the attacks are done in a coordinated way or in a centralized manner, in any cloud architecture *i.e.*, centralized or decentralized. Design and implementation of a moving target defense mitigation framework may be different according to the cloud architecture. Handling a migration event is completely dependent on the cloud scheduler architecture *i.e.*, distributed or centralized. Furthermore, the efficiency of the MTD platform is strongly based on the performance of the placement algorithm. This topic gets more importance in large-scale infrastructures such as Fog that are composed of several thousands of nodes distributed in different locations. Thus, as much as the placement algorithm is scalable to large infrastructures, it should also be reactive in order to guarantee SLA (*e.g.*, VM resource requirements) rapidly, to reduce the trade-off between security and performance in the cloud infrastructure.

As a solution to improve the performance of the MTD framework in the case of co-residency attacks such as side-channel ones, it may be extended to a local MTD on a physical host with *NUMA* (see Fig. 6) architecture with several processors, by leveraging hypervisor/OS scheduler. For instance, if a host has two processors, once a side-channel attack is detected between two instances on one of the processors, local MTD will be applied to the malicious instance by moving it to the second processor. In such scenario, the efficiency of the overall system will be raised because of none migrating the instance between physical machines in the virtualized infrastructure.

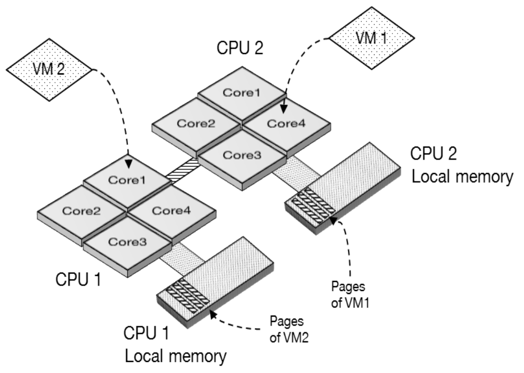


Figure 6: A NUMA architecture with two CPUs in a VM-based environment.

VI. Conclusion

In this paper, we surveyed a number of virtualization issues, related to violating isolation in cloud computing infrastructures. We identified distributed side-channel attacks as an important challenge for cloud infrastructures including novel multi-domain cloud architectures. We presented the first definition of DSCAs that can be used to exploit isolation violations: DSCAs are coordinated attacks involving several local SCAs that can be triggered to exfiltrate sensitive information from different parts of a distributed system. Finally, we sketched an approach for the mitigation of side-channel attacks using an autonomic system, for instance enforcing a moving target defense strategy. As future work, we focus on the design and implementation of an autonomic mitigation framework for multiple SCA classes in decentralized cloud infrastructures.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and its Role in the Internet of Things,” in *First ACM Workshop on Mobile Cloud Computing (MCC)*, pp. 13–16, ACM, 2012.

[2] A. Lebre, J. Pastor, and the DISCOVERY Consortium, “The DISCOVERY Initiative - Overcoming Major Limitations of Traditional Server-Centric Clouds by Operating Massively Distributed IaaS Facilities,” Research Report RR-8779, Inria, Sept. 2015.

[3] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[4] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, “Unikernels: Library Operating Systems for the Cloud,” *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 461–472, 2013.

[5] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, “Side-Channel Vulnerability Factor: a Metric for Measuring Information Leakage,” *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 106–117, 2012.

[6] C. Percival, “Cache Missing for Fun and Profit,” in *Technical BSD Conference (BSDCan)*, 2005.

[7] A. Arcangeli, I. Eidus, and C. Wright, “Increasing Memory Density by using KSM,” in *Proceedings of the Linux Symposium*, pp. 19–28, 2009.

[8] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, “Memory Deduplication as a Threat to the Guest OS,” in *Proceedings of the Fourth European Workshop on System Security*, p. 1, ACM, 2011.

[9] G. I. Apecechea, T. Eisenbarth, and B. Sunar, “Jackpot Stealing Information from Large Caches via Huge Pages,” *Cryptology ePrint Archive 2014/970*, 2014.

[10] M. J. Sepúlveda, J.-P. Diguet, M. Strum, and G. Gogniat, “NoC-Based Protection for SoC Time-Driven Attacks,” *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 7–10, 2015.

[11] M. Weiß, B. Heinz, and F. Stumpf, “A cache timing attack on aes in virtualization environments,” in *International Conference on Financial Cryptography and Data Security*, pp. 314–328, Springer, 2012.

[12] O. Acıiçmez, W. Schindler, and Ç. K. Koç, “Cache based remote timing attack on the aes,” in *Cryptographers’ Track at the RSA Conference*, pp. 271–286, Springer, 2007.

[13] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, “Cryptanalysis of des implemented on computers with cache,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 62–76, Springer, 2003.

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

[15] 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.

[16] 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.

[17] 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.

[18] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds,” in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 199–212, ACM, 2009.

[19] R. Spreitzer and T. Plos, “Cache-access pattern attack on disaligned aes t-tables,” in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 200–214, Springer, 2013.

[20] O. Goldreich, “Secure multi-party computation,” *Manuscript. Preliminary version*, pp. 86–97, 1998.

[21] W. Zhang, X. Jia, C. Wang, S. Zhang, Q. Huang, M. Wang, and P. Liu, “A comprehensive study of co-residence threat in multi-tenant public paas clouds,” in *Information and Communications Security*, pp. 361–375, Springer, 2016.

[22] C. Delimitrou and C. E. Kozyrakis, “Bolt: I know what you did last summer... in the cloud,” in *ASPLOS*, 2017.

[23] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. M. Swift, “A Placement Vulnerability Study in Multi-Tenant Public Clouds,” in *24th USENIX Security Symposium*, pp. 913–928, 2015.

[24] Y. Zhang, M. Li, K. Bai, M. Yu, and W. Zang, “Incentive compatible moving target defense against vm-colocation attacks in clouds,” in *IFIP International Information Security Conference*, pp. 388–399, Springer, 2012.

[25] V. Gopal, J. Guilford, E. Ozturk, W. Feghali, G. Wolrich, and M. Dixon, “Fast and constant-time implementation of modular exponentiation,” in *28th International Symposium on Reliable Distributed Systems. Niagara Falls, New York, USA*, 2009.

[26] M. Rivain and E. Prouff, “Provably secure higher-order masking of aes,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 413–427, Springer, 2010.

[27] D. A. Osvik, A. Shamir, and E. Tromer, “Cache attacks and countermeasures: the case of aes,” in *Cryptographers’ Track at the RSA Conference*, pp. 1–20, Springer, 2006.

[28] G. Barthe, T. Rezk, and M. Warnier, “Preventing timing leaks through transactional branching instructions,” *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 2, pp. 33–55, 2006.

[29] J. V. Cleemput, B. Coppens, and B. De Sutter, “Compiler mitigations for time attacks on modern x86 processors,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, p. 23, 2012.

[30] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, “Practical mitigations for timing-based side-channel attacks on modern x86 processors,” in *30th IEEE Symposium on Security and Privacy*, pp. 45–60, IEEE, 2009.

[31] D. J. Bernstein, “Cache-timing attacks on aes,” 2005.

[32] B. A. Braun, S. Jana, and D. Boneh, “Robust and efficient elimination of cache and timing side channels,” *CoRR*, vol. abs/1506.00189, 2015.

[33] T. Kim, M. Peinado, and G. Mainar-Ruiz, “STEALTHMEM: System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud,” in *21st USENIX Security Symposium*, pp. 189–204, 2012.

[34] Y. Zhang and M. K. Reiter, “Duppel: Retrofitting Commodity Operating Systems to Mitigate Cache Side Channels in the Cloud,” in *ACM Conference on Computer and Communications Security (CCS)*, pp. 827–838, 2013.

[35] V. Varadarajan, T. Ristenpart, and M. Swift, “Scheduler-based defenses against cross-vm side-channels,” in *23rd USENIX Security Symposium*, pp. 687–702, 2014.

[36] J. Xinxin et al., “A simple cache partitioning approach in a virtualized environment,” in *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 519–524, IEEE, 2009.

[37] J. Shi, X. Song, H. Chen, and B. Zang, “Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring,” in *IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 194–199, IEEE, 2011.

[38] B. Vattikonda, S. Das, and H. Shacham, “Eliminating Fine-Grained Timers in Xen,” in *Proceedings of the 3rd ACM Workshop on Cloud Computing Security (CCSW)*, pp. 41–46, ACM, 2011.

[39] J. Kong et al., “Deconstructing new cache designs for thwarting software cache-based side channel attacks,” in *Proceedings of the 2nd ACM Workshop on Computer Security Architectures*, pp. 25–34, ACM, 2008.

[40] F. Liu et al., “Catalyst: Defeating last-level cache side channel attacks in cloud computing,” in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 406–418, IEEE, 2016.

[41] Intel, “Improve Real-Time Performance Utilizing Cache Allocation Technology,” tech. rep., April 2015.

[42] R. Zhuang, S. A. DeLoach, and X. Ou, “Towards a theory of moving target defense,” in *Proceedings of the First ACM Workshop on Moving Target Defense*, pp. 31–40, ACM, 2014.

[43] M. Wright, S. Venkatesan, M. Albanese, and M. P. Wellman, “Moving target defense against ddos attacks: An empirical game-theoretic analysis,” in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*, pp. 93–104, ACM, 2016.

[44] S.-J. Moon, 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*, pp. 1595–1606, ACM, 2015.

[45] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, “Entropy: a consolidation manager for clusters,” in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pp. 41–50, ACM, 2009.

[46] F. Quesnel, A. Lèbre, and M. Südholt, “Cooperative and reactive scheduling in large-scale virtualized platforms with dvms,” *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1643–1655, 2013.

[47] K. Mills, J. Filliben, and C. Dabrowski, “Comparing vm-placement algorithms for on-demand clouds,” in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pp. 91–98, IEEE, 2011.