# Microarchitecture-aware Virtual Machine Placement Under Information Leakage Constraints

Arnaud Lefray*‡ and Eddy Caron*
*University of Lyon - LIP
CNRS - ENS Lyon - INRIA - UCB Lyon
{arnaud.lefray,eddy.caron}@ens-lyon.fr

Jonathan Rouzaud-Cornabas†
† Inria Beagle
INSA-Lyon, LIRIS, UMR 5205
jonathan.rouzaud-cornabas@inria.fr

Christian Toinard‡
‡LIFO
INSA Centre Val de Loire
christian.toinard@insa-cvl.fr

***Abstract*—One of the major concerns when moving to Clouds is data confidentiality. Nevertheless, more and more applications are outsourced to a public or private Cloud. In general, the usage of virtualization is acknowledged as an isolation mechanism between applications running on shared resources. But, as previously shown, virtualization does not ensure data security. Indeed, the isolation can be broken due to covert channels existing in both the software and the hardware (*e.g.,* improperly virtualized caches). Furthermore, even if a perfect control mechanism could be design, it would not protect against covert channels as they bypass control mechanism using legal means.**

**In this paper, we first describe how these attacks are working. Next, after presenting the existing mitigation mechanisms, we show that a good solution is to take into account security while allocating resources (*i.e.,* when placing the VMs). Furthermore, depending on which resources are shared, we demonstrate that the achievable bitrate of these attacks can change dramatically. We propose a new metric to quantify them and use it as an acceptable risk for isolation properties. Then, we show how to use them when allocating resources and the importance of a fine-grained resource allocation mechanism. Finally, we demonstrate that a security-oblivious placement algorithm breaks a fair amount of properties but taking into account the isolation impacts the acceptance rate (*i.e.,* the percentage of successfully placed VMs).**

***Keywords*—*Security; Cloud; Virtual Machine Placement; Isolation; Microarchitecture; Covert channels;***

## I. Introduction

Security is one of the biggest concern for the massive adoption of Clouds [6]. Indeed, multitenancy and shared resources facilitate attacks and thefts by altering the level of isolation between tenants, processes or Virtual Machines (VMs). While virtualization could be seen as an isolation mechanism between tenants, it is not the case [14]. Accordingly, it is required to have another control mechanism to enforce the isolation.

Furthermore, the security of a system is as strong as the weakest link. And, even with a perfect information flow (or access) control mechanism at the system level (*e.g.,* Hypervisor/Operating system), information can be silently leaked out (or accessed) by exploiting (unwillingly) unsecure design or implementation; this is called covert channels. The literature exhibits multiple covert channels attacks that have been successfully conducted in Cloud environments [14], [21], [22] (*e.g.,* EC2) thanks to the shared hardware components between an attacker and a victim. Previous works opt for different approaches to tackle this issue: detection of attacks, fine-grained tracking of resources usage and placement algorithms. Placement algorithms under collocation/anti-collocation constraints specify if 2 VMs can share the same Physical Machine (PM) or not. Accordingly, they do not take into account any scenarios where sharing resources is reasonable when measuring information leakage.

In this paper, we propose a new resource allocation mechanism under information leakage constraints. Our mechanism takes into account microarchitectural components as they are one of the main reasons of covert channels. Moreover, it allows to have a more fine-grained allocation and thus reduces the quantity of resources wasted due to security constraints. But first, we need a way to present the quantity of information that can be leaked between 2 applications. Our allocation algorithm can be based on any covert-channel metric but due to the lack of proper metric, we propose a new information leakage metric that can be easily used by a tenant and is both application and hardware independent. This metric quantifies information leakage through microarchitectural covert channels based on the achievable bitrate between 2 applications. Using this metric, a tenant can express its isolation properties and specify the acceptable leakage that fits his security needs. For example, banks, governments and medical establishments usually require more secure setups than research institutes running experiments. Using the application model (a set of VMs) and our metric, we propose our new resource allocation mechanism. Furthermore, we present how we have tackled the specificity of NUMA (Non-Uniform Memory Access) allocation policies to encompass both traditional and modern architectures, and proposed a model for our algorithm.

The paper is organized as follows. Section II motivates our work by presenting related works on covert channel metrics and security-aware placement algorithms. We explain our new information leakage metric in Section III. Section IV details our placement algorithm including our NUMA allocation model. In section V, after presenting our evaluation environment and scenarios, we evaluate our security-aware algorithm against a security-oblivious one. Section VI concludes this paper and presents our future work.

## II. Background

In this section, we present the background and motivation of our work. First, we do a quick introduction of microarchitecture of multi-core processors and of the security and isolation issues in Clouds. Then we explain how to break isolation and

the mitigation techniques against these attacks. Finally, we present existing metrics that quantify the quality of isolation.

### A. Microarchitectural components

Since the last few years, the complexity of physical machines' hardware topology has increased dramatically [2]. The number of cores, shared caches, and memory nodes have completely changed the microarchitecture of computers. With the increasing number of cores, scalability becomes a major issue. Modern processors use non-uniform interconnects to address it. This technology is named Non-Uniform Memory Access (NUMA) architectures. For example, Figure 1 shows the inner architectural components of 3 modern platforms. As NUMA architectures impacts the memory hierarchy model, a traditional x86 architecture can be viewed as a single NUMA with local accesses. Therefore, we encompass most legacy and modern systems by considering the NUMA model.

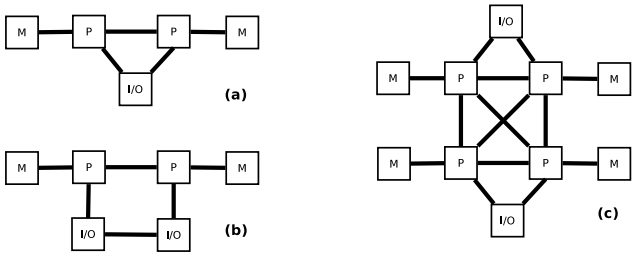


Fig. 1. Interconnection of processors (P), memory (M) and I/O chipsets in some modern architectures: (a) and (b) 2 Intel Westmere-EP processors; (c) 4 Intel Nehalem-EX processors.

### B. Security and Isolation in Clouds

Security is a major issue in Clouds [6], in particular public Clouds. Ristenpart *et al.* [14] discuss multiple approaches to exploit this co-residency in Clouds such as Cross-VM performance degradation, Denial of Service (DoS) attacks or stealing cryptographic keys, thus demonstrating the reality of security threats. They have shown the feasibility of collocating their VMs on the same physical machine as business targets in EC2. From this work, it is clear that virtualization is far from being sufficient to have a secure Cloud.

The major problem of virtualization is a weak isolation. In security, *isolation* is defined as the composition of the *confidentiality* and *integrity* properties where:

- Confidentiality is the absence of unauthorized disclosure of information.
- Integrity is the absence of unauthorized system alteration.

As a result, isolation is the absence of unauthorized disclosure and alteration of information.

In this paper, we focus on the *isolation problem* of VMs sharing the same hardware and hypervisor. Furthermore, we will not consider performance interference. Indeed, it does not allow information modification or leak even if it is due to improper isolation [18].

### C. Micro-Architectural Timing Covert channels

Cloud providers can provide a large range of security mechanisms to prevent unauthorized information flows. But, despite all the effort, there is still a potential risk of data leakage in the Cloud, that is covert channels.

A covert channel is an attack which bypasses the control mechanism using legal means to leak information to unauthorized neighbors. A covert channel breaks the confidentiality property and thus, the isolation property. Therefore, even a perfect control mechanism is useless against covert channels.

Covert channels should not be confused with side channels. A covert channel exists when 2 cooperative entities, let say *Trojan* and *Spy*, use a common protocol to communicate or exfiltrate information. For a side channel, only one process, the Spy, collects unwillingly disclosed information. Because the leaking process can be considered as an unintentional Trojan, we argue that side channels are special cases of covert channels. Recent exploits (*e.g.,* Heartblead, Ghost) have shown the easiness of remotly uploading and executing a malicious code.

Covert channels are categorized in *covert storage channels* and *covert timing channels*.

A covert storage channel exploits a standard data channel to encodes secret information. This problem has been described in 1983 by G. Simmons [15] as the prisoner's problem. This attack is popular for network protocols, for example by using the reserved or unused bits of a frame to convey information. As covert-storage channels are not due to hardware components but software design, we will focus mainly on *covert timing channels*.

A covert timing channel exploits access timings of shared resources. In his work [12], C. Percival took advantage of cache hit and miss to convey "0" and "1" respectively. With collocated VMs sharing multiple resources (hardware and system), the Cloud is an environment conducive for such exploits. Building a covert timing channel in a public commercial Cloud (*e.g.,* EC2) has been proven feasible [14], [21], [22]. A reliable bandwidth of just a hundred bits per second is enough to silently extract hundreds of 1024-bytes private keys, or tens of thousands of credit cards in a day. The DoD guideline [17] (TCSEC / Orange Book) characterizes a covert channel by its bitrate and error rate. It suggests that covert channels exceeding a threshold of 0.1 bit per second should be audited and be of concern to security if it performs over 1 bit per second.

In a Cloud environment, covert timing channels can be conducted using several hardware or system components. In Table I, we draw a summary of previous Cloud covert timing channels works. For the same component, reported bitrates can differ up to 6 orders of magnitude (*e.g.,* 0.2 bps and 190.4 kbps for L2 cache covert channel) depending on the experimental environment (*i.e.,* setup column). This gap cannot be explained by hardware differences alone (*e.g.,* CPU clock speed, cache size). How the attack is implemented also has a major impact on the bitrate.

### D. Mitigation techniques

Covert channels in multitenant environments pose a real threat with bitrates largely over the 1 bit per second standard

TABLE I. COVERT-TIMING CHANNELS SUMMARY FOR CLOUD ENVIRONMENTS

| Paper | Component | Bitrate | Setup |
|---|---|---|---|
| [12] | L1 | 3.2 Mbps | Lab |
| | L2 | 800 kbps | Lab |
| [19] | SMT/FU | 500 kbps | Lab |
| | Speculation | 200 kbps | Lab |
| [22] | L2 | 262.47 bps | Lab |
| | L2 | 3.75 bps | EC2 |
| [21] | Memory bus | 746.8 bps | Lab |
| | Memory bus | 107.9 bps | EC2 |
| | L2 | 190.4 kbps | Lab |
| [11] | CPU | 0.49 bps | Lab |
| [20] | Xen Shared-Mem | 174.98 bps | Lab |
| [14] | Memory bus | 0.006 bps | EC2 |
| | Hard disk | 0.0005 bps | EC2 |
| | L2 | 0.2 bps | EC2 |

threshold. A covert channel is by definition a reliable data channel, therefore reducing the bandwidth, preventing the channel from being reliable (*i.e.,* increasing the error rate) or removing the channel are the basic ideas for mitigation techniques.

We discuss below the mitigation possibilities given 3 perspectives: the tenant, the Cloud provider and the hardware manufacturer.

*1) Tenant:* The tenant has limited possibilities against covert channels. HomeAlone [24] is a detection approach to detect unusual L2 cache usage without relying on hardware support nor hypervisor modification. But HomeAlone's oracle is not perfect and the Spy can try to evade the detection. Moreover, according to Wu *et al.* [21], for memory bus covert timing channels, this approach would be subject to high performance overhead and high false positive rate due to non-determinism and higher access latencies. To sum up, a detection approach at the tenant level cannot be generalized and do not apply to yet to discover covert timing channels.

*2) Cloud provider:* On the other side, the Cloud provider has more latitude to mitigate covert channels. He can use detection techniques at the hypervisor level with lower overhead. Moreover, he can modify the scheduling policy to implement cache partitioning or page coloring to isolate cache/memory accesses [13], [16] at the cost of performance. Others existing preventive approaches are dedicated instances or collocation/anti-collocation placement. In this case, the isolation properties can be modelized as collocation and anti-collocation constraints [3], [7]. Nevertheless, they require to express the list of tenants with whom each VM is allowed to share resources (or not). Moreover such measures are costly for the tenant as he pays extra charges for dedicated physical machines as these approaches use coarse grain resource allocation model.

On modern public Clouds such as Amazon EC2 and Microsoft Azure, hardware multithreading called simultaneous multithreading (SMT) is disabled as cache-based covert channels attacks are easy to build. For example, L1 caches that are dedicated to one core are the easiest way to create covert channels [12] between VMs. The NoHype concept [8] consists in removing the virtualization layer while retaining the key features enabled by virtualization. They limit covert channel by enabling one VM per core but as we have shown, other covert channels exist within the microarchitecture components.

*3) Hardware manufacturer:* A covert channel is based on a faulty implementation. Thus, the hardware design is the initial reason of most covert channels. The solution would be to integrate security concerns when designing hardware components. Needless to say, it will not fix the issues on hardware already in production. Nevertheless, hardware design is out of the scope of this paper.

*4) Summary:* Because of the lack of near-future improvement in hardware design and the specificity of detection techniques, we propose a covert channel aware placement solution. Outsourced applications do not have the same level of criticality. For example, a private individual's web site is less threatened by covert channels than a banking or government application. Therefore, we consider the tenant to be responsible for specifying an acceptable information leakage risk and then, the Cloud provider has automated procedures to decide whether or not the tenant's VM should share L1/L2/L3 caches, memory bus, etc. with other VMs. Such solution can be applied to any Cloud and easily enriched with newly discovered covert channel attacks. The placement algorithm has to use a general metric to deduce a value from a given placement/hardware configuration and compare it with the tenant's risk requirement. The following part discusses existing metrics and devises a new metric to tackle practicability issues.

### E. Covert channel aware metric

Due to the recentness of Cloud (and virtualization) covert channels works, the literature is quite poor in covert channel metric propositions. Published in 2012 and after, the reference works are the metrics proposed by Demme J. *et al.* [5] and Zhang T. *et al.* [23] with respectively the Side channel Vulnerability Factor (SVF) and the Cache Side channel Vulnerability (CSV). Firstly, both metrics are design for side channels and not covert channels, but because side channels and covert channels are intrinsically related, theses metrics cannot be discard based on this sole argument to motive our new information leakage metric.

SVF and CSV are float values between 0 and 1 reflecting the degree of information leakage an observer can see from an execution. They correlate the oracle execution traces with the leaked execution traces viewed by another process in terms of cache accesses, cpu loads, etc. The approach is thus hardware-independent but the obtained value is application-specific. Indeed, the evaluation is based on the execution of a cryptographic library and it is hard to deduce if a given value would be roughly the same for an arbitrary application.

One problematic is to allow a tenant to specify a value as an acceptable risk. Because SVF and CSV are correlation factors, they can be easily used for comparison but except for limit values (0 and 1) it is quite a challenge to give a practical meaning to an intermediate value (*e.g.,* 0.467), even for a knowledgable tenant.

Therefore, both SVF and CSV are unspecifiable in practice, and thus we must come up with an alternative *covert channel aware metric* presented thereafter.

## III. Information Leakage Metric : A Micro-Architectural Security Metric

We have shown in Section II that both SVF and CSV metrics are not satisfying. Because the DoD guideline [17] (TCSEC / Orange Book) indicates that a covert channel (characterized by its bitrate and error rate) over 1 bit per second (bps) should be of concern to security, we use it as a reference value for our metric.

Therefore, we define the Information Leakage Metric as follows :

***Definition** 1 (Information Leakage Metric):* Ratio between the covert channel theoretical bitrate in a noiseless environment and the reference value (1 bps).

Unlike SVF and CSV, our metric characterizes the *worst case scenario*. In a security context, such scenario corresponds to the *maximal bitrate* a covert channel can theoretically achieve. Thus, to compute this value, we consider *a noiseless environment* and *a parallel setup* (the Trojan and the Spy run on distinct cores). Because our Information Leakage approach is based on a core characteristic (*i.e.,* the bitrate), a value exists for any timing covert channel though it can be complex to compute it.

In the following, we exemplify the approach with a cache-based timing covert channel.

### A. Cache-based Timing Covert channels bitrate in Virtualized Environment

In this subsection, we detail how a cache-based timing covert channels works using Percival *et al.* technique [12].

The idea is to time the latency of accessing memory addresses. We distinguish two cases, that is, if the accessed data are already in the cache, then the latency is small; else it must be retrieved from the main memory, then the latency is larger. As illustrated in Figure 2, the Spy accesses twice a set of cache lines (by accessing memory addresses mapped to it). When the data is in the cache, the latency is under a threshold and so the Spy reads bit “0”, otherwise it reads bit “1”. Its data are now in the cache.

Similarly, to transmit bit “1”, the Trojan accesses the same set of cache lines (by accessing its own memory addresses mapped to it) *i.e.,* he flushes the Spy’s data off the cache. To transmit bit “0”, the Trojan lets the cache in the same state by doing nothing.

A set-associative cache is divided in sets of cache lines. As shown in Figure 3, a memory address can be splitted into {TAG, SET, OFFSET} bits where OFFSET is the offset in the cache line, SET the set number and TAG the stored value to compare addresses. In a $w$-way set-associative cache, $w$ addresses mapped to the same cache line can be stored at the same time, generally based on a LRU replacement policy.

As a result, $w$ addresses must be accessed to fully flush one $w$-way cache line.

Furthermore, virtual to physical address translation rises the problem of addressing uncertainty. Nonetheless, in modern operating systems, the memory space is divided in 4 KB pages and the translation mechanism maps a virtual page to a physical page. Thus, as illustrated in Figure 3, the last 12 bits of a memory address remain identical after the translation, leaving $unknownsets$ possible cache sets for one virtual memory address where :

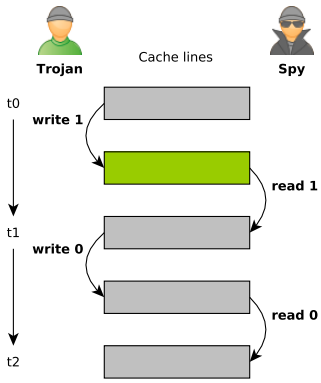


Fig. 2. Cached-based timing covert channel transmitting ”10”.

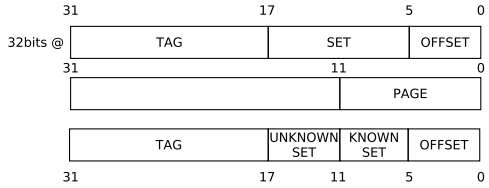


Fig. 3. Memory address cache line and page mapping.

$$unknownsets = \frac{nbsets \times linesize}{pagesize} \quad (1)$$

Therefore, to evict a Spy’s cache line, the Trojan accesses at least $minlines$ addresses where:

$$minlines = w \times unknownsets \quad (2)$$

Given the total cache size is $ttsize = w \times nbsets \times linesize$, to read or transmit a single bit the trojan accesses at least:

$$minlines = \frac{ttsize}{pagesize} \quad (3)$$

Finally, we suppose the same probability to transmit bit ”0” or bit ”1” *i.e.,* $P(0) = P(1) = 1/2$. Thus, the final time to read or write a bit is:

$$\frac{(latency_{cached} + latency_{flushed})}{2} \times \frac{ttsize}{pagesize} \quad (4)$$

### B. Discussion

Previous works on covert timing channels are mainly based on experimental results. To our knowledge, our paper is the first attempt to generalize the computation of cache-based covert timing channel bitrate. For this generalization, we consider the time between writing and reading to be null whereas experimental approaches must implement a *synchronization mechanism*. Indeed, there is a vast range of possibilities for this synchronization between the Trojan and the Spy, and choosing one would alter the quality of our metric. Moreover, this synchronization can be achieved independently from the attack itself. For example, in side channels works, the question is to detect when a cache activity corresponds to a cryptographic computation. If the victim is a web server, a simple solution is to *trigger* it by accessing a SSL-encrypted web page.

Secondly, our information leakage metric can only be based on *known* covert timing channel attacks, thus it encompasses neither unknown existing attacks nor yet-to-discover attack schemes. However, this metric does not rely on execution traces and as so, it applies to any application and, if properly adapted, any hardware. For example, atomic memory operations are improperly emulated in x86 virtualized environment [21]. These atomic operations have been implemented with a big system lock (a lock that freezes all other activities of the system) to keep the memory consistent. Consequently, a memory contention generated by one VM can be seen by another one running on the same physical machine. By observing a contention or its absence, the Trojan and the Spy can transmit respectively bit "1" or "0".

## IV. Information Leakage Aware Placement

This paper focuses on the placement of VMs with information leakage constraints specified as isolation properties. What we want to demonstrate is the effectiveness of our approach to ensure isolation and what to consider when doing placement-based security *i.e.,* how to enhance state of the art algorithms to take into account these constraints. In this section we present how isolation properties are satisfied (*i.e.,* enforced) during the placement routine. Moreover, we describe the problem of NUMA allocation and show how we tackle this issue. Finally, we prove the NP-completeness of our placement problem.

### A. Isolation Properties Satisfaction

A placement is the association between a VM and a configuration. We define a *configuration* as a set of NUMAs and cores. By *instantiated VMs*, we denote the VMs placed on a physical machine (on a set of NUMAs and cores).

A candidate VM's configuration is valid if both following conditions are met:

1) All properties of the candidate VM are satisfied regarding instantiated VMs.
2) All properties of instantiated VMs are satisfied regarding the candidate VM.

An isolation property of a first VM $vm_1$ is satisfied regarding a second VM $vm_2$ if one of the 2 following conditions is met:

1) $vm_2$ is allowed to transfer information with $vm_1$
2) the *information leakage* between their configurations is lower than the one specified by the property (as an acceptable risk).

These 2 requirements on top of the hardware ones must be fulfilled to find a suitable configuration. Furthermore, to be able to ensure that all the VMs are respecting their isolation properties (and the ones of the others) during their whole lifetime, it is important to use CPU pinning *i.e.,* to statically associate VMs with cores. Indeed, by not doing so (*i.e.,* if we let a VM changes core during its lifetime or randomly selects one at startup), one of the isolation property could be broken and so the security of the application running inside it as the configuration of the VM would have changed.

Accordingly, our security-aware algorithm must test all configurations for a VM on the physical machines and place it as soon as a configuration is valid. Moreover the placement routine must update the available resources of the platform. Actually, this update is more complex than it seems due to the way NUMA allocation works. In the following part, we present the NUMA allocation problem and the solution we propose.

### B. The NUMA allocation problem

We must consider real-world microarchitectural allocation schemes to counter real-world microarchitectural attacks (covert channels). Therefore, our proposal is to have a fine-grained control over which (microarchitectural) component is shared amongst multiple VMs to mitigate potential covert channels. With *libvirt*[1], in addition to the selection of specific cores, we can choose a set of NUMA nodes and one of the 3 available memory allocation policies:

- *interleave* that allocates memory on a given set of NUMA nodes in a round-robin fashion but falls back to other nodes if the allocation is not possible. In the worst case, any NUMA node can be allocated.
- *strict* that only allocates memory on a given set of NUMA nodes (or it fails).
- *preferred* that allocates memory on a given preferred node but falls back to other nodes if the allocation is not possible. In the worst case, any NUMA node can be allocated.

We use the *strict* policy to have a fine-grained control over memory allocation as we have for CPU cores.

Figure 4 illustrates how the strict NUMA allocation works. We consider a physical machine composed of 2 NUMA nodes with 2 Gb memory and 2 cores each. A first VM (1 core, 1 Gb memory) is bound to the core $c_1$ and the closest NUMA node (*i.e.,* the memory banks are directly connected to the core), $Numa_1$. A second VM requiring 3 cores is bound to $c_2, c_3, c_4$ which are spread on the 2 NUMA nodes. If this VM requires 2 Gb memory, allocating memory from $Numa_2$ is enough. But if 3 Gb memory are required, both $Numa_1$ and $Numa_2$ are chosen.

In terms of microarchitectural attacks, the information leakage between the 2 VMs is simply the bandwidth leakage between their bound cores and NUMAs.

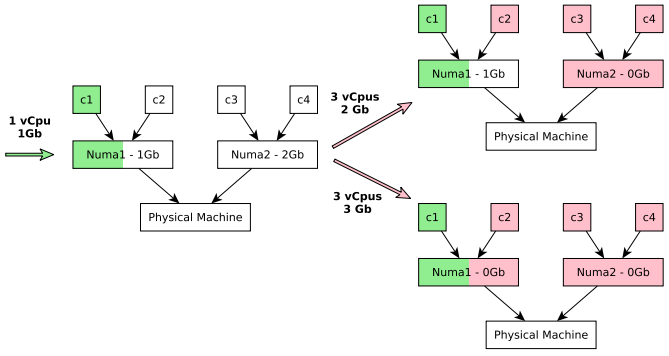


Fig. 4. Strict memory allocation policy for 2 VMs (1 core then 3 cores).

[1]The API for managing the virtualization layer on Linux: http://libvirt.org/

Figure 5 illustrates our proposal to deal with the memory segmentation. The first VM (3 cores and 3 Gb of memory) is bound on both $Numa_1$ and $Numa_2$ because it requires more than 2 Gb. The problem is that there is no indication on how much memory will be left on $Numa_1$ and $Numa_2$, a total of 1 Gb memory being free though. As a result, the second VM requiring 1 core and 1 Gb memory is also bound to $Numa_1$ and $Numa_2$. Our solution is to virtually consider the merging of $Numa_1$ and $Numa_2$, that is $Numa_{1,2}$ which has a total memory of 4 Gb, 1 Gb being free after the placement of the first VM.

Then, in terms of microarchitectural attacks, the information leakage between the 2 VMs is the bandwidth leakage between $\{c_1, c_2, c_3, Numa_1, Numa_2\}$ and $\{c_4, Numa_1, Numa_2\}$.

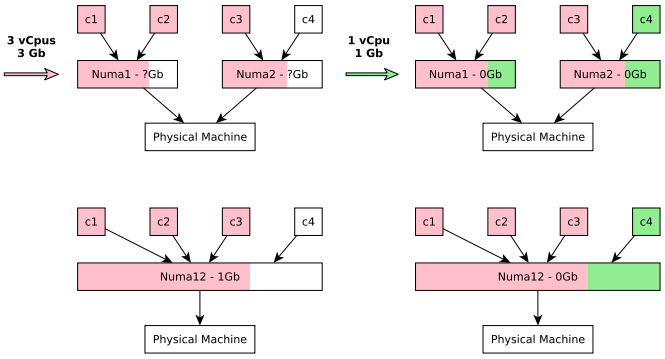


Fig. 5. Strict memory allocation policy for 2 VMs (3 cores then 1 core).

Our NUMA allocation procedure works as follows. Upon a new memory allocation request, we select a set of NUMAs $numas$ in the list of available NUMAs such as there is enough memory to allocate. If $numas$ contains more than 1 NUMA, we merge them and create a new virtual NUMA $n_{new}$, set $n_{new}$ available memory properly, and finally we replace $numas$ from the list available NUMAs with $n_{new}$. Otherwise ($numas$ is a singleton), we just update $numas$ available memory.

In the desallocation procedure, we delete a virtual NUMA if and only if there is no more VM allocated on it. When it is the case, we put the merged NUMAs back in the list of available NUMAs.

Our security-aware allocation is not rendering the placement problem less complex. Indeed, VM placement (and any placement problem) is know to be NP-hard and can be abstracted as bin-packing problem. In the next subsection, we prove that our extension actually increase the complexity of solving the placement problem.

### C. Complexity of our allocation problem

Let an infinity of cores with the VM deployment capacity $C$. Let a list of virtual machine $\{vm_{1,1}, vm_{1,2}, \ldots, vm_{n,n}\}$. We can split this set of virtual machines, in different sub-set: $\{vm_{1,1}, vm_{1,2}, \ldots, vm_{1,n}\}$ with a size $c_n$, $\{vm_{2,1}, vm_{2,2}\}$ with a size $c_2$, and so on. We want pack objects of different volumes (here a sub-set of VMs) into a finite number of bins (here cores) each of volume $C$ in a way that minimizes the number of bins (/cores) used.

We use the binary code to describe the solution and indicate if a VM is on a core or not. Variable $x_{ij}$ is egal to 1 if the VM $i$ is deployed on the core $j$, and 0 otherwise. Boolean variable $y_j$ is egal to 1 if the core is used, 0 otherwise. We try to minimize the number of core used.

$$min \sum_{j=1}^{n} y_j \tag{5}$$

This problem is known as the NP-hard problem called the Bin Packing Problem (BPP). Nevertheless, in our case we have additional constraints due to the security rules. We can consider these constraints like conflicts. The Bin-Packing Problem with Conflicts is called BPPC. It consists in determining the minimal number of identically bins (/cores) needed to store a set of items (VMs) with height is less than the capacity of bins (/cores), where some of this items are incompatible with each other (isolation rules), therefore cannot be packed together in the same bin (/core). The BPPC is a variation of the classical one dimensional BPP which is a combinatorial problem and known also to be NP-hard [4], [9]. Finally, our problem is even harder as we do not have 1 dimension (core) but 2 (memory and core). To keep the proof of concept simple, we rely on a *first-fit* heuristic to solve this BPPC problem.

## V. Evaluation

In this section, we evaluate our security-aware framework on a 3-tier use case using the information leakage metric previously introduced. A 3-tier application delivers a service (most of the time, a website) that is composed of 3 main parts: a frontend server (*e.g.,* Apache), an application server (*e.g.,* Tomcat) and a database (*e.g.,* MySQL). In this section, we consider a 3-tier application composed of 1 frontend VM, 2 application VMs and 1 database VM. Each of these VMs requires 1 core. Firstly, we present our testbed platform. Then, we detail the scenarios we use in our evaluation. Finally, we demonstrate that a security-oblivious placement algorithm breaks a fair amount of isolation properties but taking them into account impacts the acceptance rate (*i.e.,* the percentage of successfully placed submissions).

### A. Testbed platform

We consider an heterogeneous platform of a hundred physical machines. We define 2 types of existing physical machines, namely Taurus and Genepi from the Grid'5000 experimental platform [1]. In order to compute the cache-based timing covert channel bandwidth, we use the hwloc [2] tool to obtain Taurus and Genepi hardware topology.

We observe that Taurus has a shared L3 cache of 15 MB and Genepi has a shared L2 cache of 6144 KB. Then we compute the cache latencies using lat_mem_rd from the lmbench [10] benchmarks suite presented in Figure 6.

From Taurus latency measures, we can identify 4 steps, that is 1.43ns from 1 B to 32 KB (L1 cache), 4.3ns from 32 KB to 256 KB (L2 cache), 17.0ns from 256 KB to 15 MB (L3 cache). Taurus RAM latency is 108ns on local NUMA and 184.5ns on remote NUMA.

From Genepi latency measures, we can identify 3 steps, that is 1.2ns from 1 B to 32 KB (L1 cache), 6.3ns from 32 KB to 6144 KB (L2 cache), and 130.0ns above (RAM).

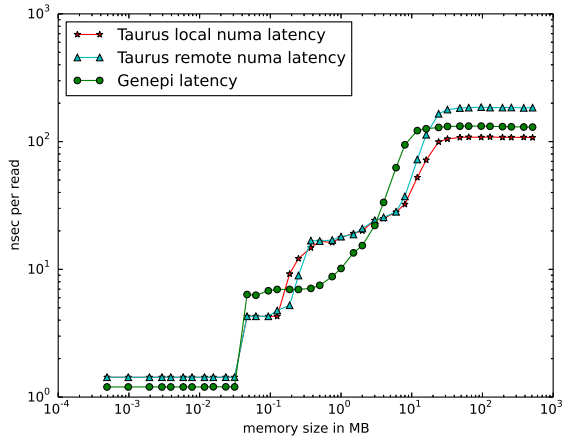


Fig. 6. Taurus (with NUMAs) and Genepi memory read latencies.

Finally, using Equation 4, Taurus L3 cache-based timing covert channel can transmit one bit in $0.5 \times (17ns + 108ns) \times 15MB/4KB = 240.0\mu s$ *i.e.,* **4.167 Kbps** bitrate on local NUMA, and $0.5 \times (17ns + 184.5ns) \times 15MB/4KB = 386.9\mu s$ *i.e.,* **2.585 Kbps** bitrate on remote NUMA.

Identically, Genepi L2 cache-based timing covert channel can transmit one bit in $0.5 \times (6.3ns + 130ns) \times 6144KB/4KB = 104.7\mu s$ *i.e.,* **9.551 Kbps** bitrate.

### B. Scenarios

For our placement evaluation, we define 4 security levels, namely *Strong*, *Medium*, *Weak* and *None*; their corresponding bitrates are given in Table II. As we want to observe the impact of bitrates on the placement (*i.e.,* acceptance rate), these levels are based on our platform microarchitecture (*i.e.,* Taurus and Genepi). Determining which levels are relevant to security requires a thorough study, and thus this question is out of the scope of this paper. Considering our platform, a VM with a *Strong* isolation property cannot share any cache with any other VM. Similarly, a VM with a *Medium* Isolation can only share Taurus's L3 if memory is allocated on a remote NUMA. Either a local or a remote allocation is possible with a *Weak* Isolation. Finally, only the *None* security level allows VMs to share Genepi's L2 cache.

TABLE II. SECURITY LEVEL AND ASSOCIATED BITRATES

| **Security Level** | **bitrate** ($B$) |
|---|---|
| Strong | $B < 2.585Kbps$ |
| Medium | $2.585Kbps \leq B < 4.167Kbps$ |
| Weak | $4.167Kbps \leq B < 9.717Kbps$ |
| None | $B > 9.717Kbps$ |

It is reasonable to admit that a tenant considers other tenant's VMs as a bigger threat than its own VMs. Therefore, we define a two-fold policy where *internal* refers to the security level between the 4 VMs of an application, and *external* refers to the security level with other VMs in the Cloud.

### C. Placement Results

Our objective is to analyze (1) the risk, in terms of unsatisfied security rules, of a security-agnostic algorithm as implemented in current Cloud platforms, and (2) the acceptance rate with our security-aware algorithm which guarantee the satisfaction of all rules.

In this evaluation, we consider all internal/external security level combinations with the internal level always weaker (or equal) than the external one.

We simulate a platform that is composed of 50 Genepi physical machines and 50 Taurus physical machines for a total of 1000 cores. We evaluate the submission of 250 3-tier applications requiring a total of 1,000 cores (*i.e.,* 4 VMs $\times$ 250 apps.) to saturate the platform.

We evaluate 2 First-Fit algorithms:

- *security-agnostic* (sec-agnostic FF). It does not try to verify any security rules and so will always have a 100% acceptance rate.
- *security-aware* (sec-aware FF). It must satisfy all rules and so will always have 0% of unsatisfied rules.

The results presented in Table III show that not considering security (*i.e.,* sec-agnostic FF) exposes a non-negligible number of instantiated applications to covert channels attacks. Even with light constraints like *Medium*/*None* some VMs are still insecure (15% of unsatisfied rules) whereas sec-aware FF is able to place 100% of the submissions.

On the other hand, strict security policies come at great expenses. For example the *Strong*/*Strong* scenario can only achieve 30% acceptance rate. A future work will be to budget an application according to its security constraints. These first results gives us an overlook of the trade-offs induced by security levels.

TABLE III. FIRSTFIT PLACEMENT RESULTS

| **Scenario** | | **sec-agnostic FF** | **sec-aware FF** |
|---|---|---|---|
| **External** | **Internal** | (% of unsatisfied rules) | (acceptance rate in %) |
| Strong | Strong | 100 | 30 |
| Strong | Medium | 65 | 60 |
| Strong | Weak | 50 | 60 |
| Strong | None | 30 | 80 |
| Medium | Medium | 50 | 80 |
| Medium | Weak | 35 | 80 |
| Medium | None | 15 | 100 |
| Weak | Weak | 20 | 80 |
| Weak | None | 0 | 100 |
| None | None | 0 | 100 |

Due to our simulation setup (platform and submissions), all acceptance rates obtained with sec-aware FF are *optimal*. It would not be the case for heterogeneous inputs though. The proof is quite simple. In short, with the *Medium* and *Weak* levels, 100% of Taurus cores can be allocated if using a remote NUMA, that is 12 cores. For Genepi physical machines, the *Medium*/*Weak* levels only allow one core per L2 to be allocated out of 2. As a result, only half of the cores can be allocated on Genepi physical machines *i.e.,* 4 cores. We obtain (12 cores + 4 cores) $\times$ 50 physical machines = 800 cores for 1000 available cores, and thus the 80% acceptance rate obtained with a First Fit is *optimal* for *Medium* and *Weak* levels. The proof is similar for other levels, and thus the design of other security-aware algorithms would be relevant only for a more heterogeneous setup where First Fit is not optimal. As we want to focus on a proof-of-concept in this paper, the study of other algorithms is part of our future work.

## VI. Conclusion

Even with perfect information flow control mechanisms, virtualized environments are still sensitive to silent information leakage, that is covert channels, due to shared hardware ressources. This paper proposes a fine-grained placement based on the tenant's isolation properties to tackle this issue. The tenant submits an application *i.e.,* a set of VMs, and he defines foreach VM the acceptable risk in isolation properties, with respect to other VMs. Due to the lack of usable covert channel metric to qualify an acceptable risk, this paper proposes a new information leakage metric and demonstrates how to compute it with the example of a cache-based timing covert channel. As covert channels exploit microarchitecture flaws, we integrate the specificity of NUMA allocation schemes in our placement algorithm. Furthermore, this paper outlines the risk-factor of traditional placement algorithms while demonstrating the feasability of taking into account information leakage constraints.

This work is a proof-of-concept for placement-based security. This paper covers VM isolation constraints on microarchitectural components. A next step will be to apply our approach to networking. Combining computing and networking resources, designing efficient provisioning algorithms for arbitrary Cloud applications is part of our future work. Independently, an open issue is how to define what is an acceptable risk for real-world tenant's specifications. Finally, we have shown the trade-off between infrastructure consolidation and security requirements. A perspective is to propose a model for the Cloud provider to charge these security requirements.

## Acknowledgment

This work was done thanks to the financial support of the Celtic+ project SEED4C (Eureka Celtic+ CPP2011/2-6).

## References

[1] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[2] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst. hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 180 –186, feb. 2010.

[3] E. Caron and J. Rouzaud-Cornabas. Improving users' isolation in iaas: Virtual machine placement with security constraints. In *IEEE CLOUD 2014. 7th IEEE International Conference on Cloud Computing*, Anchorage, USA, June 27-July 2 2014. IEEE Computer Society.

[4] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for np-hard problems. chapter Approximation Algorithms for Bin Packing: A Survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[5] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan. Side-channel vulnerability factor: A metric for measuring information leakage. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 106–117, June 2012.

[6] T. Jaeger and J. Schiffman. Outlook: Cloudy with a Chance of Security Challenges and Improvements. *IEEE Security & Privacy Magazine*, 8(1):77–80, Jan. 2010.

[7] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible. Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-aware Virtual Machine Placement. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 72–79. IEEE, 2011.

[8] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. NoHype: Virtualized Cloud Infrastructure without the Virtualization. *SIGARCH Comput. Archit. News*, 38(3):350–361, June 2010.

[9] M. Maiza and M. S. Radjef. Heuristics for solving the bin-packing problem with conflicts. *Applied Mathematical Sciences*, 5(35):1739–1752, 2011.

[10] L. W. McVoy, C. Staelin, et al. lmbench: Portable tools for performance analysis. In *USENIX annual technical conference*, pages 279–294. San Diego, CA, USA, 1996.

[11] K. Okamura and Y. Oyama. Load-based covert channels between xen virtual machines. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, page 173180, New York, NY, USA, 2010. ACM.

[12] C. Percival. Cache missing for fun and profit. 2005.

[13] H. Raj, R. Nathuji, and A. Singh. Resource management for isolation enhanced cloud services. *CCSW '09 Proceedings of the 2009 ACM workshop on Cloud computing security*, page 77, 2009.

[14] 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*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[15] G. J. Simmons. The prisoners' problem and the subliminal channel. In *Advances in Cryptology: Proceedings of CRYPTO '83*, pages 51–67. Plenum, 1983.

[16] K. Taesoo, M. Peinado, and G. Mainar-Ruiz. System-Level Protection Against Cache-based Side Channel Attacks in the Cloud. In *Proceedings of the 21st Usenix Security Symposium*, USENIX Security'12, pages 1–16, Berkeley, CA, USA, 2012. USENIX Association.

[17] TCSEC. Trusted Computer System Evaluation Criteria. Technical Report DoD 5200.28-STD, Department of Defense, 1985.

[18] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift. Resource-Freeing Attacks: Improve your Cloud Performance (At your Neighbor's Expense). In *Proceedings of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 281–292, New York, NY, USA, 2012. ACM.

[19] Z. Wang and R. Lee. Covert and side channels due to processor architecture. In *Computer Security Applications Conference, 2006. ACSAC '06. 22nd Annual*, pages 473–482, Dec. 2006.

[20] J. Wu, L. Ding, Y. Wang, and W. Han. Identification and evaluation of sharing memory covert timing channel in xen virtual machines. In *2011 IEEE International Conference on Cloud Computing (CLOUD)*, pages 283–291, July 2011.

[21] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[22] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, page 2940, New York, NY, USA, 2011. ACM.

[23] T. Zhang, F. Liu, S. Chen, and R. B. Lee. Side channel vulnerability metrics: The promise and the pitfalls. In *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, pages 2:1–2:8, New York, NY, USA, 2013. ACM.

[24] Y. Zhang, A. Juels, A. Oprea, and M. Reiter. HomeAlone: Co-residency detection in the cloud via side-channel analysis. In *2011 IEEE Symposium on Security and Privacy (SP)*, pages 313–328, May 2011.