Catch Me if You Can: A Closer Look at Malicious Co-Residency on the Cloud

Ahmed Osama Fathy Atya*, *Member, IEEE,* Zhiyun Qian*, *Member, IEEE,*
Srikanth V. Krishnamurthy*, *Fellow, IEEE* Thomas La Porta† *Fellow, IEEE,*
Patrick McDaniel† *Fellow, IEEE,* and Lisa Marvel‡ *Senior Member, IEEE,*
*University of California, Riverside, †Pennsylvania State University, ‡U.S. Army Research Laboratory

Abstract—VM migration is an effective countermeasure against attempts at malicious co-residency. In this paper, our overarching objectives are to (a) get an in-depth understanding of the ways and effectiveness with which an attacker can launch attacks towards achieving co-residency and (b) to design migration policies that are very effective in thwarting malicious co-residency, but are thrifty in terms of the bandwidth and downtime costs that are incurred with live migration.

Towards achieving our goals, we first undertake an experimental study on Amazon EC2 to obtain an in-depth understanding of the side-channels an attacker can use to ascertain co-residency with a victim. Here, we identify a new set of stealthy side-channel attacks which, we show to be more effective than currently available attacks towards verifying co-residency. We also build a simple model that can be used for estimating co-residency times based on very few measurements on a given cloud platform, to account for varying attacker capabilities. Based on the study, we develop a set of guidelines to determine under what conditions victim VM migrations should be triggered given performance costs in terms of bandwidth and downtime, that a user is willing to bear. Via extensive experiments on our private in-house cloud, we show that migrations using our guidelines can limit the fraction of the time that an attacker VM co-resides with a victim VM to about 1 % of the time with bandwidth costs of a few MB and downtimes of a few seconds, per day per VM migrated.

1 Introduction

Infrastructure-as-a-Service (IaaS) providers allow VMs that belong to different users, to share the same physical infrastructure. Thus, the risk of sharing a physical machine with a potential malicious VM is very real [1], [2], [3]. Once an attack VM is able to co-reside with a victim VM on the same physical machine, it can launch arbitrary attacks (e.g., using side channels to achieve information leakage) to compromise the security of the victim VM. Although providers continuously make improvements to better isolate resources across VMs, new vulnerabilities are expected to emerge as hardware architectures and hypervisor technologies evolve [4].

Prior to launching such attacks however, the attacker typically needs to place a malicious VM on the physical machine that houses the victim VM (co-reside with the victim). The process of performing co-residency today requires the attacker to launch its VMs, use side-channels to ascertain co-residency, and upon failures terminate and repeat the process. Once co-residency is achieved, the longer a victim VM resides on the same physical machine occupied by the attacker VM, the higher is its risk of being compromised.

In this paper, we have two main objectives. First, we seek to get an in depth understanding of the ways and the effectiveness with which an attacker can achieve co-residency with a victim VM in practice. Towards this, we undertake an extensive experimental effort on Amazon's EC2 cloud infrastructure, to understand the side channels that an attacker can use to ascertain co-residency with a victim VM. En route, we discover a new set of very stealthy and highly effective timing based side channels that can be used today to ascertain co-residency.

Migrating a VM is a way of mitigating long periods of co-residency with an attacker VM [5]. As our second objective, we seek to determine under what conditions a victim VM should be migrated to minimize its co-residency time with an attacker, given a bandwidth/downtime cost the user of the VM is willing to bear. Towards this (based on the above experimental studies) we formulate a set of guidelines, which are based on (a) the time that a victim VM has resided on a host machine and (b) monitoring the side channel that the attacker could have used to ascertain co-residency. We perform extensive experiments on our in house cloud (built using CloudStack [6]) to demonstrate that our guidelines can drastically reduce the times for which a victim VM co-resides with an attack VM with low costs in terms of downtimes and bandwidth.

To summarize, our contributions are as follows:

- We carry out extensive experiments on Amazon EC2, arguably the most popular cloud provider, to develop a comprehensive understanding of the efficacy of an adversary in successfully co-residing its VM with a victim's VM.
- We build a simple model that can provide us with rough estimates of how long it takes for an attacker with varying capabilities, to successfully co-reside with a victim. The model requires very few measurements and can provide guidelines on how often VMs should be migrated in different scenarios.
- We discover a set of new highly effective timing based side channels that can be used by an attacker to determine if any of its VMs co-resides with a targeted victim VM. Our side channel provide the highest accuracy in ascertaining co-residency as compared to other previously proposed side channel tests (≈ 86 %), but with lower false positive rates. In addition, we believe that they are much harder to detect than the latter since they do

not create explicit congestion on a shared resource.

- We consider VM migration as a countermeasure to thwart malicious co-residency and develop a set of guidelines on when to invoke victim VM migration given a cost budget in terms of the bandwidth expenses and downtimes that the user is willing to tolerate. We perform extensive experiments on an in house KVM-based private cloud (users cannot invoke live migrations on commercial clouds today) to evaluate our guidelines. The results show they can drastically reduce the times for which a victim VM co-resides with an attack VM. Specifically, with very reasonable performance costs (of the order of MB of bandwidth and seconds of downtime per day, per VM migrated), the fraction of time that the victim VM co-resides with an attack VM can be limited to about 1 %.

Roadmap: In §2 and in §3 we present related work and our threat model, respectively. We present an experimental study on co-residency on Amazon EC2 and showcase our new attacks in §4. We present a model to estimate the time taken to achieve co-residency in §5. Our mitigation guidelines are derived in §6. We present our evaluations in §7. A discussion on how the migration guidelines influence costs while mitigating side channels targeting information leakage is provided in §8. We conclude in §9.

2 Related work

Side channel attacks targeting information leakage: Side channel attacks exploit physical information leakage such as timing information, cache hits/misses, power consumption etc. This information is typically obtained based on the usage of shared resources (e.g., cache). There are several side channel attacks on cloud tenants that have been previously studied [1], [2], [7], [8], [9], [10], [11]. Side channel attacks can target different shared resources; examples include the cache, shared storage etc. For example, the FLUSH+RELOAD attack [12] achieves information leakage by flushing the L3 cache and observing the times taken for reloading specific memory blocks previously in the cache; a short time indicates that the memory blocks were reloaded by the victim. Based on the attack, the time taken to successfully extract information ranges from the order of minutes to hours. We point out here that almost all such work on information leakage attacks however, assume that the victim has already co-resided with the victim process. In contrast, we focus on side-channels used by an attacker to ascertain co-residency.

Co-residency with a victim process: For almost all side-channel attacks reported, an attack process (VM) will need to co-reside with the victim process (VM) on the same physical machine. The attacker will need to launch its VM and use some kind of side channel to ascertain if has co-resided with a victim. Side channels have also been proposed for enabling co-residency checks (e.g.,[13]). However, these prior efforts do not provide a comprehensive understanding of how effective these are in terms of their accuracy and the time it takes for an attack VM to successfully co-reside with a victim VM.

Reverse engineering the algorithm for determining the placement of VMs empirically, as in [13] and [14], although hard, might be useful in the short term. However, placement algorithms are likely to dynamically change over time [14]. Because of this, one can largely consider the placement algorithms to be opaque (and possibly customized to users); instead of trying to reverse engineer the process, we develop measures to determine co-residency, and construct a model which provides rough estimates of the average time taken for acheiving co-residency (regardless of the nuances of placement). Further, cloud providers have ensured that many co-residency checks proposed much earlier (e.g., [2], [15]) are no longer feasible[1]. To the best of our knowledge, we are the first to propose network timing based side channels for ascertaining co-residency. Note that there are other network timing based attacks previously studied (e.g., [16], [17], [18], [19]) but they are quite different. For example, the authors in [16]) look at the times between transmitted packets to infer the keystrokes of a user. [18] and [17] use timing towards inferring which nodes are communicating on a Tor network. [19] uses timing channels towards website fingerprinting. While the idea of using time as a side channel is common with these efforts, we look at the time series in contacting two processes on a physical machine on the cloud from external vantage observation points, to ascertain co-residency. Thus, our solution is quite different (and thus novel) from what has been previously considered.

Defending side-channel attacks: Cloud providers as well as the research community is continuously looking for ways to improve resource isolation which can help defend against side channel attacks. Efforts such as [20] and [21] introduce random delays while accessing a resource to thwart timing based side channel attacks. [22] and [23] employ software level defense mechanisms as countermeasures against cache based side channel attacks; for example, the idea in [22] is to obfuscate the program at the source code level to provide the illusion that many extraneous program paths are executed. If an attacker conducts a prime and probe attack (where he primes the cache and probes for determining changes to cache sets) his observations will be skewed. Although the current methods can defend against known side channel attacks, it is unclear if there exist other type of attacks that are unknown to the research community. Other vulnerabilities could appear in the future due to advancements in computer architecture and hypervisor technologies. Mitigating malicious co-residency can significantly alleviate the attacker's ability to launch such attacks.

VM migration to mitigate side-channel attacks: VM migration has recently been considered to counter cloud-based side-channel attacks targeting information leakage in [5]. In brief, the authors of Nomad [5], model information leakage from side channel attacks over time and determine how often migration is needed. However, they assume that the attacker has successfully co-resided with the victim. Nomad also assumes that such decisions on migration to cope with side channels, are made by the provider and not the users of the VMs. The users may not be willing to accept the performance penalties (downtimes) that inevitably occur when VMs are migrated for such purposes. We make no assumptions on what the provider will do in terms of placement of VMs.

Unlike in Nomad, we try to minimize the occurrence and periods of successful malicious co-residency. We account for the time that an attacker takes in order to successfully co-reside with a victim; this can influence the costs associated with migrations (reduce the frequency). The users can choose when to migrate their VMs based on their risk averseness and the costs they are willing to bear.

In a realistic scenario, where users are likely to configure their own VM migration policies (e.g., enable, disable, choose

1. We have also experimentally verified that this is the case. For example, the cloud cartography approach to locate victim processes in the cloud proposed in [2] is no longer viable.

periodicity etc.), not accounting for the fact that the attacker takes time to first co-reside with the victim (as with Nomad) in addition to the time taken for an information leakage attack, for driving migrations will increase migration frequencies and thus bandwidth/downtime costs (discussed later). We later show that very frequent migrations could also adversely affect security when the attacker simply stays put on a physical machine (since the victim VM can potentially return to the very same machine).

3 Threat model and Roadmap

3.1 Threat Model

We assume that an attacker seeks to co-reside its VM on the same physical machine as a certain targeted victim VM and co-reside for as long as possible. First we consider a case where the attacker launches a set of VMs, repeatedly if needed, and attempts to have one of these attack VMs co-reside with a victim's VM. We assume that it has no knowledge or control over the policies followed by the cloud provider for VM placement (as with Amazon's EC2). We call such an attacker an "reactive attacker"; this scenario is reflective of what the attacker can do on today's commercial clouds.

Next, we consider the cases where (like any user) an attacker can choose to migrate its VM (or stay fixed on a single physical machine), if user driven migrations are allowed. We assume that the provider does not unilaterally perform migrations (without user requests) like in [5], since this may cause downtimes without user consent (which some users may not want to experience). An attacker VM could simply choose to stay put on its initial physical machine assuming that the target VM (due to migration) will be placed on the same (physical) machine eventually. We call such an attacker a "static attacker". Finally, we also consider a possibility that an attacker may choose to migrate periodically itself. We call such an attacker, a "periodic" attacker. In both of the above cases, we assume that an attacker continuously checks for co-residency (using one of the approaches discussed in Section 4), since the victim could now at any point, migrate to the machine on which its VM resides. Note that these attack strategies cannot be implemented and tested today on Amazon's EC2 (migrations are not viable as of today); we test them on our in house cloud in Section 7.

Once the attacker is able to verify with high accuracy that one of his attack VMs has successfully co-resided with the victim's VM on the same physical machine, an attempt is made to launch a previously proposed side-channel attack (described in Section 8) to successfully create a leakage of information from the victim. However, we do not explicitly focus on such side-channel attacks themselves in this work; we provide a discussion on the impact of our work on such attacks in Section 8. For simpilicity, we assume that the number of virtual machines owned by the victim remains unchanged, i.e. the number of virtual machines does not vary over small time scales (hours or days). We also assume that after a migration occurs, the attacker does not know "where the victim process has been migrated." We assume that it is not interested in triggering other attacks (e.g., causing a DoS attack by inducing repeated migrations).

3.2 Roadmap

The remainder of the paper is organized as follows. First, we seek to showcase and understand attacks that target malicious co-residency via an extensive measurement study on Amazon EC2 (Section 4). This leads to results on how long it takes for an attacker to co-reside with a victim and subsequently launch any information leakage attack. In addition, we believe that it naturally leads to the following two questions: (a) Given that one cannot perform exhaustive sets of experiments, can we develop a model to roughly characterize these co-residency times?, and (b) Given the co-residency times and certain patterns that manifest themselves due to the attack, how can one guide migration decisions that are cost-effective ? We answer the first question by developing a simple rough, yet reasonable model in Section 5. The second question is then addressed via assessing risk and developing cost-effective migration guidelines in Section 6. Finally, we evaluate our guidelines in Section 7. We point out here that while the measurement study in Section 4 were done on EC2, the evaluations in Section 7 are done on an in house cloud since they cannot be implemented today on a commercial cloud.

Model	Virtual CPU	CPU Credits / hour	Mem (GiB)	Storage
t2.micro	1	6	1	EBS
t2.small	1	12	2	EBS
t2.medium	2	24	4	EBS
t2.large	2	36	8	EBS

TABLE 1: Instance Type Comparison

4 Characterizing co-residency via experiments

We perform extensive experiments on Amazon's EC2 over a period of 5 months, to obtain an understanding of (a) the accuracy and (b) the time taken by an attacker to successfully (we define what mean by success below) co-reside its VM with a targeted victim VM (T_{CR}), while using different types of side-channels to verify co-residency. Specifically, we implement and test, multiple previously proposed ways of verifying co-residency (co-residency tests). In addition, we design new timing based co-residency tests that are stealthy and yet, very effective. We reiterate here that some of the previously proposed side-channel based tests to verify co-residency (e.g., [2], [15]) do not work anymore (as verified by our experiments) since cloud providers have taken steps to prevent them [10].

Categories of co-residency tests: We divide co-residency tests into two categories; controlled/internal and external. In internal co-residency tests (ICT), we control both victim and attacker VMs. These tests primarily demonstrate co-residency with high fidelity (and can serve as ground truth) but cannot necessarily be used by an external adversary. Specifically, we create contention on a shared resource using the attacker VM and measure the changes in access times to that resource experienced by the victim VM (as compared to the access times experienced with no contention). We assume that a co-residency test is successful if an associated ICT test classifies the test as being successful.

In the external co-residency tests (ECT), we only control the attacker VM. We exploit a service running on the victim VM to try to create contention on a (possibly) shared resource. The attack VM then compares the response times to the service with and without contention. Here, we also propose the use of new timing tests to decide whether two machines co-reside or not. The accuracy of the ECTs (which represent the mode of operation of an adversary in a real setting) is assessed by comparing the result with that of a ICT. The time taken for successful external co-residency tests is a critical metric that we are interested in. In all of these experiments we assume an active attacker as discussed in Section 3.

General setup: We initiate 20 micro or 20 small instances each, for a victim and an attacker account (for ground truth validation). All the instances run Ubuntu 14.04 LTS [24]. We conducted our experiments on three different datacenters (us-west-2a, us-west-2b and us-west-2c). Four different side channels that are exploitable to verify co-residency, reported in the past three years, were implemented [13], [3], [15] and tested. A key contribution we make is the design of new, very effective, timing-based co-residency tests.

The victim VM hosts 3 services, each very different in nature; thus, the co-residency tests in the three cases, for identical set ups, take different times. The services are Taiga, ownCloud, and MediaServer. Taiga is an open-source project manager software that involves a mix of CPU, disk (frequent), and memory workloads. ownCloud is an open-source file hosting service (resembles Dropbox) that involves memory and disk intensive workloads. MediaServer is an open-source wiki-page server that involves a mix of CPU, disk (rare), and memory workloads. We use different type of VMs; their specifications are summarized in Table 1 (we use the Amazon EC2 jargon [25]). Later in the section, we provide further details on each specific experiment.

ICT experiments: In each experiment, as mentioned, the attacker and the victim have 20 VMs each. We end up with 400 possible combinations of attack and victim VMs (20×20); thus, 400 co-residency tests will need to be performed for each considered shared resource. With four shared resources this translates to 1600 tests[2]. A comprehensive study using this approach would take a prohibitive amount of time, considering that the failure of co-residency would incur termination and relaunching of attack VMs. To speed up the process, we create a pre-configured VM image that can be readily instantiated. Specifically, we create an image of all your setup and programs and when we launch a new VM, instead of using a brand new OS to launch our programs and configure them, we simply use the image. Second, we recognize that the failure of co-residency tests using certain shared resources, can with high probability suggest that other tests (on other shared resources) will also fail. Therefore, by ordering the tests, we drastically reduce the number of tests (by eliminating tests that are highly likely to fail). For example, the failure of the co-residency test that considers the memory bus as the shared resource (bus contention test) described in [13] (details later) will indicate that the attack and the victim process do not share the same CPU. Finally, for each attack VM, we perform a co-residency test with regards to a shared resource, with all the victim VMs in one shot. By using these reductions, we were able to reduce the time taken per run to 20 minutes, on average. Thus, by running experiments for 15 hours per day, we were able to test more than 50,000 pairs of attack and victim VMs.

ECT experiments: We go through a similar process (with all the optimizations outlined above) except that we can only induce workloads on victim VMs by sending external requests (e.g., to upload a new file onto the ownCloud server). Further, we will allow all 20 attack VMs to perform the tests simultaneously. If any of them detects the co-residency successfully, we further cross-validate the result with a follow-up ICT experiment. If the validation holds true, the process is deemed as success and stopped.

Launching and termination: For every failed attempt at co-residency (the co-residency test fails with respect to a considered shared resource), the attacker must terminate his VMs and then re-launch them in an attempt to again successfully co-reside with his target victim. The time taken to launch and terminate these processes are denoted by t_l and t_d, respectively. These times will contribute to the overall time that an attacker will have expend, in order to successfully co-reside with the victim VM. Using the process described above, we experimentally quantify these times.

2. We wish to point out here that these tests are primarily used as benchmarks and cannot be actually used by an external attacker.

4.1 Implementation of prior co-residency tests

We implement and test previously proposed co-residency tests on EC2. In these tests, an attacker creates contention on various shared resources (e.g., cache, bus) and uses a side channel to determine if his process co-resides with a victim process. The time taken to perform a co-residency test is denoted by t_c; this time primarily depends on type of shared resource used to determine co-residency, as well as the type of service running on the victim VM. A successful co-residency test indicates with high probability that the attack and the victim VM share the same physical resources.

Bus contention based ICT and ECT: Bus contention tests were designed and evaluated in [1] and [13].

ICT: In the ICT, the attacker VM allocates a chunk of memory that is larger than the size of the last level cache (64 MB in our experiments). Is then misaligns the memory access pointer by adding an offset to it (two bytes). The issuing of an unaligned, atomic access operation (such as a read, or the XADD operations for x86 processors) [13] causes the locking of the memory bus. The causes the victim VM to see a significant increase (around 3X) in the memory access time if both the VMs share the same physical machine.

ECT: In the ECT, the attacker engineers a set of external requests (such as HTTP or FTP) which cause the victim VM to access the memory bus; the request sizes will have to be larger than the last level cache (LLC) size [13]. Internally, the attack VM locks the memory bus (similar to what is done in the ICT). By comparing the response times in this case, with the response times without the locking of the memory bus, the attacker is able to determine if his VM co-resides with the victim VM (a significant increase is seen if the VMs co-reside).

LLC based ICT: Next, we test if two VMs share the same CPU by creating contention on the LLC [26]. If this causes an increase in the LLC access time for the victim VM, we conclude that both VMs share the same CPU. The LLC in our experiments is the L3 cache; its size is 25.6 MB with a cache line size of 64 bytes and an associativity of 20. The page size is 4096 bytes.

The attacker VM allocates 1 or more GB of memory and regularly reads and writes in multiple page size increments (to ensure a page miss for the victim VM). The victim VM allocates an L3 cache size (of 25.6 MB) and iteratively reads pages in order. If the two VMs share the L3 cache, a significant increase (≈ 1.8 X) in the access time is observed by the victim VM.

ICT with L1 cache: Two VMs that share the same CPU do not necessarily share the same core. We test if two VMs share the came core via a contention based test on the L1 cache. The L1 cache size is 32 KB, its associativity = 8, and the page size is 4096 bytes. As in the previous test, the attack VM repeatedly evicts the

L1 cache by requesting data that is not in the cache. The victim process continuously tries to access the same data (e.g., a certain data structure) repeatedly, and measures the access times. With contention a 1.5-3X increase in the access time is experienced. Note here that even if the two VMs share the same CPU (the LLC test could yield a success), this test will fail if they do not share the same core. If two VMs share the same core, an active side channel attack can probe the contents of the L1 cache during execution [27].

4.2 A new ICT

Storage based ICT: The test seeks to determine if two VMs share the same disk. Although the idea of checking if a disk is shared is not new [14], [13], to the best of our knowledge, we are the first to design and implement an approach to determine co-residency based on disk storage access in a cloud environment. We find that this test does not inherently provide any advantage over the previously proposed ICTs discussed above (as shown later), but we present it nevertheless, for completeness. To create contention for disk access, the attack VM accesses a relatively large file ($>$ 6 GB). It repeatedly does this access while varying the block size from 512 Bytes to 8 MB. The victim tries to perform a storage operation; in our experiments, it copies a similar large file from storage and we measure the average time taken. For each block size used, we measure the average time taken by the victim for copying the file from storage. The attacker's goal is to cause an increase in the average seek time (compared to when it does not access the large file). For the Amazon EBS (Elastic Block Store) storage, a minimum increase of 33% in the total transfer time is observed when there is disk contention.

4.3 New timing based ECTs

Next, we propose a set of new, stealthy yet very effective timing-based ECTs.

ECT based on RTT timing behaviors: Today, cloud providers employ load balancers and multi-path routing to be able to dynamically handle high traffic loads and denial of service attacks [28]. However, it is reasonable to assume that packets which are destined to VMs that reside on the same physical machine are exposed to similar effects at a given time (same paths), and experience similar delays along the route. Thus, one might expect that while the behaviors change dynamically, the delays experienced by packets that are destined to VMs on the same physical machine exhibit consistent (similar) temporal variations over short time scales (the packets to the attacker VM and the victim VM are sent back to back). Note here that while the congestion at routers can change over time, given that TCP gradually responds to congestion variations, these can be expected to happen over coarser (relatively larger) time scales. Given this observation, we (playing the role of the attacker) utilize probes to measure the delays to an attacker VM and victim VM to determine whether they co-reside on the same physical machine. The probes do not have to be explicit overt messages; as an example, they could potentially be seemingly legitimate http requests to a victim web server. We point out here that since it is an attacker who does the probing, the additional load due to the same on the network is of no concern to the attacker; from a detection standpoint it would seem like legitimate load, and hard to disambiguate especially if the probes are sent from various vantage points.

In order to ensure that the results are not biased by hypervisor scheduling delays (typically the maximum time slice that a VM can obtain before being switched out of context $\approx$ 10 msec) we probe at coarse time granularities (i.e., the probing period is set to $\geq$ 100 msec). With this set up, we receive one response from each of the two VMs under consideration (the attack and the victim VM) per probe. If the two VMs are on the same physical machine a delay observed between the responses from the VMs is much less than the probing period; unless (in rare cases) where the physical machine load rapidly changes, these delays are of the order of OS scheduling delays. Otherwise, much higher variations are observed in the delays (because of different routes and traffic on those routes). In order to perform the comparisons, we normalize the observed values with respect to the maximum observed response times (to eliminate temporal variations in load across the paths taken by the probes).

We measure the round trip times (RTTs) continuously (by instantiating connections separated by randomly chosen periods) from an outside observer (attacker) to the two VMs in question. We then perform a time series analysis to determine whether the timing profiles observed with respect to the two VMs are very similar. We call this test the "behavioral timing test".

To ensure that we can accurately compare the time profiles, we instantiate the connections to the two VMs (almost) simultaneously. To measure the RTT, we primarily rely on the TCP handshake. We also use ICMP messages when applicable. By collecting a long enough RTT trace, we can accurately determine if the two VMs are on the same physical machine or not. Further, we can deduce if the VMs are connected to the same TOR (top of the rack) switch. The intuition is that if the two VMs are co-located on the same physical machine, they experience similar processing delays (which depends on the workload of the machine). Indeed, as shown later (Table 3), the results of this test are fairly accurate as validated by comparison with our ground truth.

Let $\vec{rtt}(o, d_i) = [rtt(o, d_i, t_1) \ \ldots, \ rtt(o, d_i, t_n)]$, denote the two time series for the attack and victim VM. ($i \in \{1, 2\}$, where, the d_1 is the attack VM and d_2 is the victim VM); o and n represent the observer and duration of the observation, respectively. We use two commonly used metrics to measure the distance between the two time series, viz., the Mean Square Error (MSE) [29] and the Pearson Coefficient (PeC) [30].

The mean square error given by:

$$MSE(o, d_1, d_2) = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(rtt(o, d_1, t_i) - rtt(o, d_2, t_i))^2} \quad (1)$$

Here, we also measure the MSE relative to a time-shifted versions of the the vector; this is to account for the fact that in practice, the connections to the two VMs cannot be established simultaneously (one is time shifted slightly with respect to the other).

The PeC between the time series is given by,

$$PeC(o, d_1, d_2) = \frac{COV(\vec{rtt}(o, d_1), \vec{rtt}(o, d_2))}{\sigma(\vec{rtt}(o, d_1)) \times \sigma(\vec{rtt}(o, d_2)).} \quad (2)$$

COV is the covariance between the two time series.

In our experiments described in Section 4.4, the test declares a success (the two VMs co-reside) if the MSE is < 0.15 and the PeC is ≥ 0.8. Note that ideally, if there is an exact match between the time series, the MSE will be 0 and the PeC will be 1; we choose thresholds that are close to these ideal values.

Timing based ECT with multiple observers: In this test, the attacker uses multiple observers (from different vantage points)

Resource	type	# Avg. VMs	# of hits	Percentage	ECT
Disk	ICT	108000	3340	3%	NA
Bus	ICT	108000	22666	21%	NA
CPU	ICT	98000	19540	20%	16808
Core	ICT	98000	8820	9%	NA

TABLE 2: Hit rates with different ICTs and ECT (NA: Not applicable)

and examines the RTTs between the observers and the VMs in question (as in the previous test). The attacker only records the minimum RTT observed between each observer and the target VMs for a specific time span. These RTT values are quantized to the nearest decile (i.e., a value of 46 is rounded to 50, while a value of 41 is rounded to 40). A vector of these quantized RTT values are constructed for both the attack and the victim VM. For example, if there are three observers and ρ_j and ρ'_j are the quantized RTT values with respect to the attack and the victim VMs at the j^{th} observer, the vectors $\{\rho_1, \rho_2, \rho_3\}$ and $\{\rho'_1, \rho'_2, \rho'_3\}$ are constructed. The similarity between these vectors is now computed by means of a Hamming distance type measure. The two VMs are considered to be on the same physical machines, if their these distances coincide (similar to the network triangulation [31]). If the two signatures are very similar (but don't exactly match) it is very likely that that the VMs are on the same physical machine. (in our experiments described in Section 4.4, we assume that they are similar if the fraction of elements that match is ≥ 0.8). The accuracy of the approach depends on the number of observers used and the distribution of observers inside and outside the datacenter. Our general observation is that the accuracy improves with the number of observers, and the diversity in their locations. While we could ensure the diversity of observers outside the datacenter, we cannot control the locations of those within. In an extreme case, the results commiserate with those of the behavioral test (only one observer). As the diversity of observers increase, we see that the accuracy improves (and the variance is within 1 % of the reported results). The accuracy also improves as we increase the duration of the observations. We call this test the "signature based timing test."

ECT based on RTT timing behaviors from multiple vantage points: The principles of the first two timing tests described above are used in conjunction to improve the accuracy of the co-residency determination. This also decreases the time required to get accurate results. In brief, multiple observers are again used; however, each observer applies the first test on the behaviors of the RTTs (analyzes the similarity between the time series) as opposed to using the minimum RTT a signature (as in the second test). If from most (80 % or higher) vantage points, the behaviors of the time series obtained with respect to the attack and victim VMs are deemed similar, the attacker assumes that its process has successfully co-resided with its victim. We call this test the "hybrid timing test."

4.4 Experimental results

Next, we provide our experimental results on Amazon EC2 with regards to (i) the accuracy and (ii) the time taken for successful co-residency with the different side-channels to verify co-residency.

Launching and Termination times: Launch and termination times are part of the overheads consumed by an attacker while trying to achieve co-residency. They could differ from datacenter to datacenter. Note that the attacker may have to try different sizes of attack VMs to achieve co-residency since the cloud providers placement policies may result in the placement of VMs of different sizes on different physical machines (e.g., according to load) [32].

Fig 1 shows the minimum, maximum and average launch times over five months. The average launch time is $\approx$ 1.5 minutes. The termination times are longer in general than launch times as shown in Fig 2. The average termination time is 2.5 minutes. Upon closer examination, we find that VMs that are on a heavily loaded physical machine (longer response times) take longer to terminate. We also find that larger VMs take longer to terminate (see Fig. 4); this is because the deallocation of resources takes longer in such cases.

Hit rate of ICTs: We next quantify the hit rates of the ICT tests on Amazon's EC2. This refers to the percentage of times that the ICT test flags a success across all our experiments. We conducted on average, 3 experiments per hour. As described earlier, in each experiment we launch 20 attack VMs and 20 victim VMs. Table 2 shows the hit rates with the various ICTs discussed earlier. The results show that the disk is shared only 3% of the time. From among this 3%, we find that approximately 86% were sharing the same physical machine (the bus test was also a success). As shown, in approximately 21% of the cases the attack and victim VMs shared the same physical machine. A significant percentage of these also shared the same CPU ($> 90\%$); the LLC cache test was a success. In only 9% of the cases an attack VM shared a core with a victim VM (L1 cache test succeeded).

Accuracy of the ECTs: Next, we examine the accuracy of the various ECTs described in the previous section. The results are summarized in Tables 3-6. We report the results with 3 different services. Each service was tested for 5 consecutive days. We approximately conducted 50 runs per day. We re-launch the victim VMs every 10 runs, while the attacker VMs were re-launched every run. We used the bus contention based ICT to establish the ground truth; in other words, that test was also conducted and the result we obtained was considered the truth. This is because, first, the work in [13] shows that this test has very high accuracy with regards to determining the co-residency of two processes. Second, the bus test is the only test for which we have both the ICT and ECT tests. Finally, if the bus test is successful, as discussed above, it is very likely that the LLC cache test will also be successful.

We examine the efficacy of each ECT by measuring the number of true positives (TP), number of true negatives (TN), number of false positives (FP) and number of false negatives (FN). We compute the true positive rate (aka *sensitivity* [33]) to be $\frac{TP}{TP+FN}$. Similarly, the true negative rate (aka *specificity* [33]) is $\frac{TN}{FP+TN}$.

The results show that the bus based ECT and the hybrid timing test exhibit similar (very high) sensitivity and specificity. They are both more accurate than behavioral and signature based timing tests. Note however that designing the requests for the bus contention based ECT is complex. It needs to be tailored to the type of service running on the victim VM. Furthermore, some services do perform heavy memory transfers; if a victim VM hosts such a service, (e.g., ownCloud) it is difficult for an adversary to successfully carry out a bus based ECT. This demonstrates that for some workloads, the bus contention ECT may not be effective; the hybrid timing tests seem to work well with all the workloads we considered.

We notice that for the ownCloud service, the sensitivity takes a hit with the hybrid timing test as compared to the first two timing tests. This is because these results have to do with observations in RTTs assoiated with transfers from remote clients. If the remote

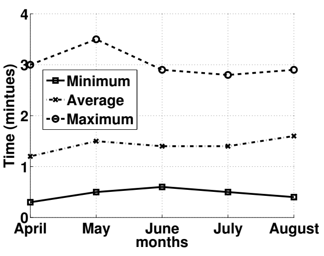

Fig. 1: Launch times over months

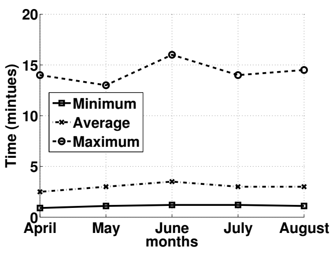

Fig. 2: Termination times over months

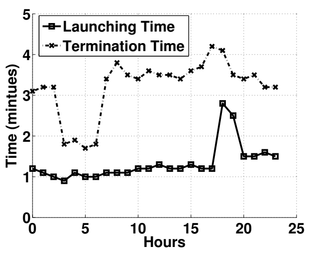

Fig. 3: Avg. launch and termination times over days

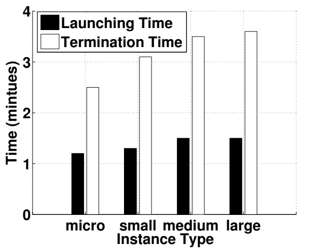

Fig. 4: Avg. launch and termination times for different VM sizes.

Application	Sensitivity	Specificity
Taiga	0.94	0.84
MediaServer	0.95	0.88
ownCloud	0.95	0.96

TABLE 3: Sensitivity and specificity with the behavioral test.

Application	Sensitivity	Specificity
Taiga	0.92	0.89
MediaServer	0.93	0.88
ownCloud	0.93	0.95

TABLE 4: Sensitivity and specificity with the signature test.

Application	Sensitivity	Specificity
Taiga	0.95	0.95
MediaServer	0.95	0.95
ownCloud	0.90	0.94

TABLE 5: Sensitivity and specificity with the hybrid test.

Application	Sensitivity	Specificity
Taiga	0.95	0.95
MediaServer	0.95	0.95
ownCloud	0.81	0.95

TABLE 6: Sensitivity and specificity with the Bus based ECT.

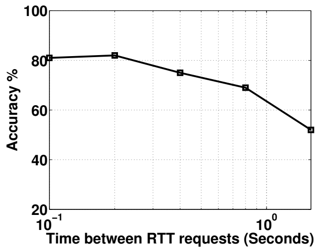

Fig. 5: Accuracy vs the average time between RTT samples.

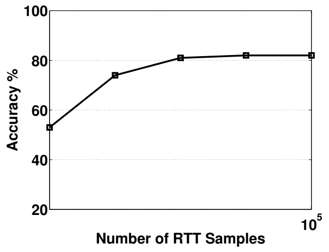

Fig. 6: The improvement in accuracy by increasing the number of samples.

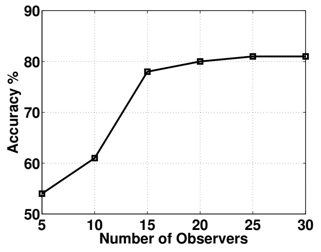

Fig. 7: The improvement in accuracy as the number of observers increase.

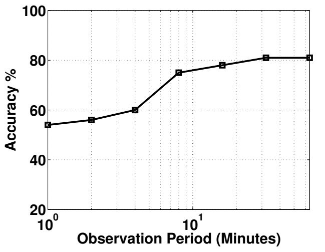

Fig. 8: The impact of observation period on the accuracy

access has greater variance in the observed transfer times (which happens with increased external/internal observers), this can cause the sensitivity to dip. This only manifests itself with ownCloud because of the large file sizes that are fetched. However, we see that the dip is not substantial. For Taiga and MediaServer, where the transfer sizes are much smaller, the variations do not cause any decrease in sensitivity.

A microscopic view: In Figs. 9 and 10, we show snapshots of the normalized time series at a single observer (using the hybrid timing based ECT), when we have a mismatch and a match, respectively. As evident, with a match or hit, the difference in the normalized response times obtained with respect to the victim and the attack VM is much smaller than 0.1 for each sample. With a mismatch, this can be as high as 0.6. In the rare cases with false positives (the ICT yields a mismatch), we find that the normalized RTT is slightly higher ($\approx$ 0.2 for some of the sampled points); this could be a result of the two VMs being close to each other (same rack) but not on the same physical machine.

Properly configuring the ECT tests: The work in [13] discusses how to properly configure the parameters for an effective bus contention ECT (we follow the same approach). Instead we focus on determining how to appropriately configure our new behavioral and signature based timing tests in the next set of experiments. The results reported here are from experiments done over 30 days; we perform 30 runs per day.

Configuring the behavioral timing test: We vary both the total number of samples taken (to construct the time series) and the average time between samples. As seen in Fig. 5, an average period of 200 msec yields the highest accuracy from among the values we considered. While this value could change depending on the dynamics inside the cloud (how often paths change etc.) we find that roughly choosing this average time between probes is enough; slightly higher or lower sampling rates do not cause significant degradations in performance. Note here that since, the path from the observer to the cloud provider remains fairly stable all the packets experience similar delays on this part of the path; the delays within the cloud are more dynamic. Fig. 6 shows the sensitivity of the accuracy to the number of samples taken to form the time series. We observe that going beyond 1000 samples yields little improvement in the accuracy. With these values for the average time between samples and the number of samples, it takes around 200 seconds (3.5 minutes) to perform the behavioral timing test. To decrease the false positives and negatives for the test, the experiments are repeated thrice with at least a 2 - 5 minute pause time, between the tests. Thus the total time taken is between 15 and 26 minutes per attempt.

Configuring the signature based timing test: We vary the number of observers considered while creating a signature. As seen in Fig. 7, the accuracy does significantly improve if we increase the number of observers above 15. Beyond this, the accuracy is pretty much stable and equal to 81%. To remove fluctuations in RTTs resulting from traffic variations and intra-cloud dynamics (paths on which traffic is routed may change dynamically [34]), we need a larger set of samples in order to obtain accurate signatures. We find that typically we need to collect observations over 20 minutes for each run. However, using the hybrid test where we essentially perform a behavioral RTT test at each observer improves the accuracy without the need for a

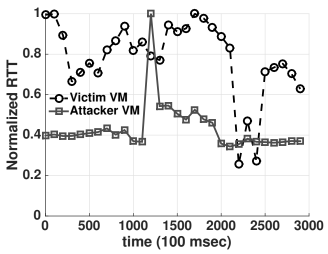

Fig. 9: Time series snapshot of responses from attack and victim VMs upon a miss

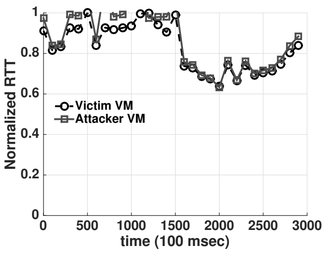

Fig. 10: Time series snapshot of responses from attack and victim VMs upon a hit

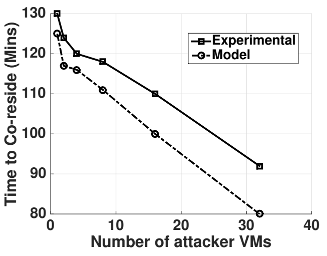

Fig. 11: Average time to co-reside with any of 8 victim VMs.

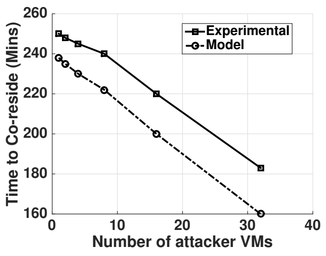

Fig. 12: Average time to co-reside with any of 4 victim VMs.

ECT Test	Average	75%	85%	95%
Behavioral	110	72	45	21
Signature	135	93	53	26
Hybrid	120	78	45	23
Bus	105	76	65	33

TABLE 7: Average and percentiles (mins) of times taken for co-residency.

longer observation period; conducting the same behavioral test as before at 16 observers, we find that the accuracy improves to $\approx$ 86 % (the time taken is less than 20 mins).

Experimentally computed times for co-residency: In Table 7 we present the experimentally determined, average times with the different ECTs. The times shown include the (a) the launch time, (b) the time taken to determine co-residency (for the tests described) and (c) the termination time. We also show the percentile times taken for co-residency; for example, the column that shows the 85th percentile depicts the maximum time taken by the most effective (lowest time) 85 % of the tests. We note that the average times in all cases are about 2 hours. We find that 75 % of the tests took more than 72-93 minutes depending on the ECT in use. More than 95 % took $>$ 20 minutes. The minimum times taken to successfully determine co-residency could be small depending on the test (as seen above). However, the attacker has to really get lucky and must be able to colocate with the targeted victim VM with very few launches of his VMs. We discuss risk assessment and various policies on VM migration (when should a VM be migrated to minimize the risk of long co-residency with an attacker?) in the Section 6.

5 Modeling co-residency times

Prior to successfully co-residing with a victim process, an attacker may have to iteratively launch attack processes, check for co-residency (perform the co-residency tests), and upon failure, terminate the processes and relaunch the set of processes. Here, we assume that the attacker has a single account on the cloud, and goes through this procedure until he is able to successfully co-reside with the victim. The time taken for successful co-residency depends on (i) the number of VMs the victim has on the cloud (a web provider may have multiple re plicas of his web server running [35]) (ii) the number of attack processes launched in each iteration and (iii) the cloud provider's policy in placing a customer's VMs. In our experiments described earlier, we assumed that the attacker is able to launch 20 VMs in an iteration and the victim has 20 processes running on EC2. Note that Amazon's EC2 limits the number of VMs one can launch with a single account to 20. The provider's policy on VM placement here is unknown.

It is hard to consider all possible cases and perform experiments to characterize the times taken for establishing co-residency. Thus, we seek to develop a simple model that allows us to estimate this time, based on the number of attack and victim VMs; we show that while this model is not very precise, it provides good enough (rough) estimates that can be used to guide migration decisions. To keep things simple, we assume only t2-micro instances which run on subset of the machines in the region under consideration. As shown later, we do so for a meaningful comparison of the results with the model with that via experiments (constrained by the degree of experimentation we could perform). We expect that (as the model and experiments indicate), the results can be extrapolated for a larger number of machines.

If u is the victim, and the probability of successfully co-residing an attack process with any of the m replica VMs the victim is running, in a given attempt, is $p_c(u)$, the expected time for successful co-residency is given by:

$$E_{CR}[p_c(u)] = (t_l + t_d + t_c) \sum_{j=1}^{J} j(1 - p_c(u))^{j-1} p_c(u) \quad (3)$$

where, J is the maximum number of attempts the attacker makes at co-residency. Since we assume a presistent attacker, we set $J = \infty$. With this it is easy to see that:

$$E_{CR}[p_c(u)] = (t_l + t_d + t_c) \frac{1}{p_c(u)} \quad (4)$$

Let $p_c(u, m)$ be the probability of successful co-residency with the m^{th} victim VM replica in an attempt and t_c is the time taken to perform the co-residency test. We assume that a victim's VM replica does not share the same physical machine with another replica; conservatively, this maximizes the attacker's chances since he has a better chance of *hitting* a victim VM replica in each attempt. When a user creates an instance on EC2, it is guaranteed a certain CPU resource based on the type of instance (e.g., a certain long term CPU allocation for a small instance is guaranteed). Furthermore, it is well known that EC2 physical machines are not operating at near 100 % utilization; in fact, every server typically runs at fairly low utilizations [36]. As a consequence, we argue that the likelihood of any of the machines being equally likely to be chosen is a realistic assumption. In fact, our experiments did not hinge on any assumption, and the results seem to roughly correspond to what our model predicts.

With the above assumption, the probability of co-residing an attack process with any of the replica VMs is given by:

$$p_c(u) = \sum_{m} p_c(u, m). \quad (5)$$

It is hard to determine $p_c(u)$ without knowing the placement policy of the provider. If one assumes a completely random placement policy (meaning that a physical machine is chosen at random for placement), this probability is $p_c(u,m) = \frac{1}{\mathcal{N}}$, where, $\mathcal{N}$ is the number of available physical machines.

If a plurality of the attacker's VMs are placed on the same physical machine, then he is at an inherent disadvantage (the number of physical machines on which he can check for co-residency in that attempt, is reduced). Thus, we conservatively (to minimize the co-residency time) assume that the attack VMs are always placed on different machines (as with the victim VMs). This policy is not far from what is likely to happen in reality (e.g., see [32]). For example, a provider may place the VMs of a customer on different physical machines for reliability (robustness to machine failures). If we assume that such a policy is in place, it is easy to see that one can conservatively bound the probability $p_c(u,m)$ by:

$$p_c(u,m) = \frac{1}{\mathcal{N}} + \frac{1}{\mathcal{N}-1} \cdots + \frac{1}{\mathcal{N}-\mathcal{A}} \leq \frac{\mathcal{A}}{\mathcal{N}-\mathcal{A}} \qquad (6)$$

where, $\mathcal{A}$ is the number of attacker VMs. Thus, if there are $\mathcal{L}$ victim VM replicas, $p_c(u) = \frac{\mathcal{L}\times\mathcal{A}}{\mathcal{N}-\mathcal{A}}$.

Evaluating our model for co-residency time estimation: Next, we compare the co-residency estimates using our model, with that from experiments on EC2. Figs. 11 and 12 depict the times taken to co-reside with any of the victim VMs, where the victim has deployed 8 and 4 VMs respectively. The number of attacker VMs are varied (for the experiments, we have two accounts and can have up to 40 VMs in total). The value of $\mathcal{N}$ can be estimated based on the number of IP addresses made available on the provider's launch interface and the maximum number of VMs that can be hosted per machine; we use $\mathcal{N} = 500$ since EC2 provides around 4000 available IP addresses and the Xen hypervisor allows 8 VMs per physical machine. The model takes as input, the average (possibly offline) measurements of t_l, t_d and t_c with one attack VM. We see that with different number of attack VMs, we are able to get relative good (but rough) estimates of the co-residency times with the model. This shows that the model can be useful in predicting how long attackers with differing capabilities will take, to successfully co-reside with a victim.

6 Determining when to migrate

Next, we seek to develop guidelines on when a VM should be migrated. Towards this, we first propose a set of indicators that capture "the risk of a VM co-residing with an adversarial VM." Note that without knowing the capabilities of an adversary it is hard to quantify risk; thus, the risk indicators are based on what can be directly measured either by the customer (user) or the provider.

6.1 Risk indicators

To assess risk, we consider a set of measurable indicators, the variations in which implicitly indicate an increase in risk. These indicators are: **(i)** The time that a victim VM spends on a physical machine relative to the time taken by an adversary to successfully achieve co-residency. As evident, the longer the time spent on the same physical machine, the more probable it is that an adversary has successfully co-resided on the same machine. **(ii)** The level of utilization of the memory bus on the physical host machine. This is the same side channel used by an attacker using the bus contention ECT to ascertain co-residency. From the perspective of the victim, a heavy utilization of the bus can be the result of the bus contention based ECT. It is quite possible that such heavy utilization is because of benign congestion; we argue that even then, migration would help in improving performance. Note that with the timing based ECTs that we discover, the second risk indicator is not useful; in other words, migration has to be based on the time spent by the victim VM on the physical machine.

Time indicator: The first very simple risk indicator is the time for which the VM has resided on the current physical machine and is represented by $\tau = t - t_i$ where, t is the current time, t_i is the time at which the VM was first placed on that physical machine. If one assumes that the timing based ECTs are used, the time indicator is the only metric that can be used to guide migration decisions.

Heavy memory bus utilization indicator: A heavy utilization of the memory bus may indicate that the bus contention based ECT is underway. We sample the utilization of the bus periodically at intervals t_s. If this utilization, on machine m is greater than a threshold for a specific sample (say j), we set a boolean variable associated with that resource $S(m,j)$ to 1 (it is set to 0 otherwise). A composite risk indicator $V(m,K)$ is obtained by jointly considering say K consecutive samples. Specifically,

$$V(m,K) = \sum_{j=k}^{k+K} S(m,j), \qquad (7)$$

for any k. If this risk indicator yields a value of K, then all K consecutive samples indicated that the bus experienced a high utilization; this would indicate that the VM is at risk of being subjected to a bus contention ECT.

Threshold for determining heavy bus contention: Typically, for specific platforms, there are specifications for the maximum values associated with this heavy utilization indicator. For example, for SDRAM, the specification says that the maximum memory access time is 70 - 150 ns depending on the vendor [37]. One could set the threshold to be a certain fraction of the specified maximum value (e.g., the threshold could be 0.8 of the maximum specified time). Clearly, the higher the threshold, the less likely it is that an alert is issued (leading to a low true positive rate with regards to detecting a threat); on the other hand, setting too low a threshold would incur a higher cost (because of possibly frequent VM migrations). Thus, this threshold could be set based on the user's risk averseness and the costs she is willing to bear. In our work, we conduct an extensive empirical experiments and we set the threshold to a pre-defined value ($Th = 0.8$) so as to keep the false positive rate below 1 %. Similar thresholds are used with the bus contention ECT [13]. We assume that this threshold will be fixed and other parameters are tuned to determine when VMs are to be migrated as discussed next.

6.2 Migration guidelines

Next, we try to develop guidelines for migration based on the risk indicators. We assume that the provider does not unilaterally take decisions to migrate a VM (since some users may be unwilling to experience downtimes towards reducing risk). Instead, it monitors the bus utilization at some preset time intervals t_s (not controlled by the user) and based on user preferences (with regards to certain parameters as discussed later), migrates her VMs.

Our guidelines for performing migrations are characterized using the flow chart in Fig. 13. A user's virtual machine enters

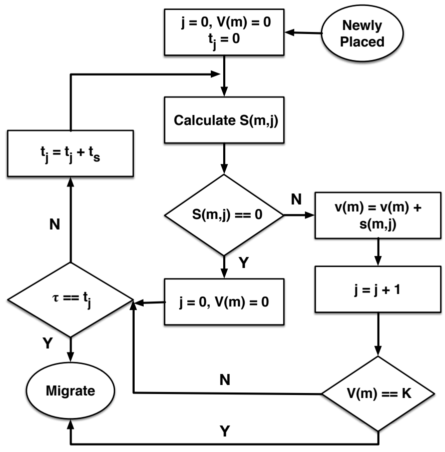

Fig. 13: Migration Guidelines

a safe state when it is placed on a physical machine. The value of $S(m, jt_s)$ on that physical machine m, is checked by the provider at each sampling instance of jt_s (sampling is done every t_s time units i.e., $j = 1, 2, \ldots$). If this value is 1, the VM enters the monitor state. If the VM remains in the monitoring state for K consecutive monitoring instances, this implies that $V(m, K) = K$ and and it should be migrated. The machine returns to the safe state if at any point while in the monitoring state, the the value of $S(m, (j + l)t_s)$ (where $l < K$) becomes zero (i.e., the utilization of the bus gets back below the chosen threshold). If the VM remains on the physical machine (regardless of whether how long it spends in the safe or the monitoring state) for τ seconds a decision is made to migrate. We wish to point out here, that there is an implicit assumption that $Kt_s << \tau$.

For ease of discussion, as mentioned earlier, let us assume that the threshold Th is fixed. The two parameters that define the user's cost and risk averseness are K and τ. If the values chosen for these parameters are too small, the number of false positives with respect to detecting a co-residency threat increases; unnecessary high migration costs are experienced. On the other hand, if the values chosen are too high, an attacker can succeed in its attempt to co-reside and do so for long periods. In our experiments reported in Section 7 we choose empirical values that provide a good trade-off between the cost and risk averseness (as measured offline). It is hard to come up with optimal values for these parameters; thus, we empirically measure this (as a provider would do) and provide a recommendation with regards to values that provide reasonable cost versus security trade-offs.

Costs: In the best case, the user does not migrate for a period of τ seconds. If the size of her VM is X MB, her bandwidth expense in this case will be $\frac{8X}{\tau}$ Mbps. She will experience a downtime every τ seconds. In the worst case, she will continuously observe bus contention, and will migrate every Kt_s seconds. Here, the bandwidth cost will be $\frac{8X}{Kt_s}$; she will experience a downtime every Kt_s seconds. In practice, if there is no attack or if there is a timing based ECT, the former (best case) will hold true. If there is a bus-contention based ECT, the times between the migration instances will be somewhere in between (and not excluding) the best and the worst case scenarios.

7 Evaluations

In this section, we experimentally evaluate the VM migration in terms of (a) reducing the times for which an attack VM co-resides with a victim and (b) the incurred costs; the migrations are based on the guidelines put together in Section 6.2. Unfortunately, Amazon EC2 or other cloud providers do not yet offer a service wherein a user can control (or request) when VM migrations are performed; therefore, our evaluations are on an in house private cloud.

Evaluation Scenarios: We consider the two best ECTs that can be used by the attacker (the hybrid timing based ECT and the bus contention ECT) to ascertain co-residency. We consider the two risk indicators separately and jointly, to invoke migrations.

Our private cloud testbed: Our private cloud consists of 13 Servers (11 DELL and 2 HP), two Cisco 20-Port gigabit switches and 9 DELL hosts. It can host up to 140 micro VMs or 70 small VMs, simultaneously (equivalent to t2.micro and t2.small on EC2, respectively). We run the KVM hypervisor on top of Ubuntu 14.04. On the VMs, we run Centos 7 and Ubuntu 15 images. We deploy Apache CloudStack [6] to provision the VMs. We perform live migration by using virt-manager (KVM + QEMU). We host Taiga, ownCloud and Mediaserver on the VMs and use 9 hosts to initiate requests to the deployed VMs (background traffic).

Although our testbed is much smaller than commercial clouds, it suffices for a proof-of-concept implementation and showcasing the effectiveness of the VM migration based on our guidelines. On commercial clouds, it is our hope that VM migration will be even more effective than what we show because of scale. However, we acknowledge that our smaller scale cloud might not capture factors that exist in a real cloud setting (such as route changes due to intra-cloud dynamics); a detailed study of these at scale will be considered in the future.

Evaluation results: Next, we present our results. Migration costs are averages over 24 hour periods unless specified otherwise. In the first set of results we consider the reactive attacker model described in Section 3; this is what the attacker can do to-day on commercial clouds. Subsequently we consider the cases where it can make choices of whether or not to migrate (static and periodic attackers).

Evaluations with a reactive attacker: First, we present our evaluations with a reactive attacker.

Migration based on time of residency: First, we consider migration based on only the time of residency (value of τ). Here, we do not trigger alerts from the heavy bus contention utilization indicator. We consider the case where the hybrid timing based ECT strategy is used by an *reactive* attacker[3]. We migrate a VM if the time spent on a physical machine is equal to $\tau = \beta \times f(T_{CR})$, where f is some monotonically increasing function of the time taken by the attacker to successfully co-reside with the victim. β determines how conservative we are in migrating a VM; a smaller of β invokes more frequent migrations and thus, incurs higher cost. For simplicity, we consider that the function f provides the mean value of T_{CR} (based on the values from from Section 4 we set $f(T_{CR}) = 105$). Two values of β, which correspond to inter-migration times τ of approximately 60 ($\beta \approx 0.6$) and 120 mins ($\beta \approx 1.1$), are considered. The lower the β value indicates high user averseness to risk, while the higher the β means a lower risk averseness (and that cost is more important to the owner of the victim VM). The victim VMs are either Taiga, Mediaserver and ownCloud.

3. Since the bus contention utilization indicator is not used, the results are very similar when the attacker used the bus contention ECT.

Interval	Bandwidth	Downtime	Efficiency
1 hour	25.6M	1.4s	0.008
2 hour	13.2M	0.8s	0.05

TABLE 8: Average cost and attack efficiency with proactive migration (1 victim VM and 1 attack VMs).

Interval	Bandwidth	Downtime	Efficiency
1 hour	25.6M	1.4s	0.03
2 hour	13.2M	0.8s	0.08

TABLE 9: Average cost and attack efficiency with proactive migration (1 victim VM, 2 attack VMs).

Victim VMs	Bandwidth	Downtime	Attack Efficiency
1	13M	0.75s	0.11
2	14M	0.73s	0.11

TABLE 10: Average cost and attack efficiency for migration based on heavy memory utilization (varying victim VMs).

Victim VMs	Bandwidth	Downtime	Attack Efficiency
1	16M	1.1s	0.03
2	16M	1.2s	0.028

TABLE 11: Average cost and attack efficiency for migration based on both residence time and heavy memory utilization (Attack VMs = 2X Victim VMs).

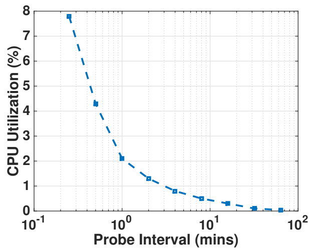

Fig. 14: The CPU utilization costs with memory probing.

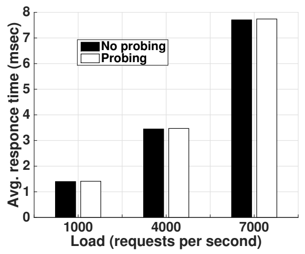

Fig. 15: The average response times with and without memory probes.

We conduct the experiments over a period of 15 days (5 days per type of VM). The cost incurred by the victim is measured in terms of (a) the downtime that it experiences and (b) the bandwidth consumed. The bandwidth consumed corresponds to the memory state (in MB) transferred during the live migration of the victim VM. To capture the security provided, we compute the ratio of the time for which the attack VM co-resides with the victim VM to the total duration of the experiment; we call this the attack efficiency. We have two victim VMs. We populate the machines with 35 additional VMs which are randomly placed, in order to reflect a real operational setting (the cloud has a utilization of approximately 30 %). We consider two and four attacker VMs (i.e., 1X and 2X the number of victim VMs). In this experiment we assume an reactive attacker who performs the hybrid timing based ECT to ascertain co-residency; with this approach, he can re-launch his VMs 1.7 times an hour, on average.

We summarize the costs (downtimes and the traffic generated by migration) and the attack efficiency in Tables 8 and 9 with different numbers of attacker VMs. As expected, the traffic volume is doubled if the migration periodicity doubles. The average downtimes are also doubled. However, the attacker efficiency is less than 1% if the VMs are migrated every hour, compared to 5 % if the period is increased to 2 hours. This is because the drop in the attacker success rate is not linear with increased migration frequency (it is better). We see that the costs in terms of downtimes ($<$ 2 s) and bandwidth (of the order of MB over 24 hours) are reasonable.

Migration based on heavy memory utilization: In our next experiments, we assume that the bus contention ECT is used by an reactive attacker. We only migrate a VM if the heavy bus contention risk indicator is triggered. Note here that if the attacker uses the hybrid timing based ECT, the victim's VM will never be migrated in this case. We set the access time threshold to (100ns) (about 0.8 of the maximum specified time on our platforms). The value of K is set to 10. First, we set $\tau = \infty$, i.e., we only use the heavy memory access time risk indicator as a trigger. Table 10 summarizes the results. The costs of migration decrease significantly compared to the case where migration is proactively performed based on the time indicator (recall results in Tables 8 and 9). However, since migration is only performed upon detecting long memory access times, the attacker is able to co-reside with the victim VM for slightly longer periods (in quite a few cases the heavy utilization is not consistently above the chosen threshold); thus, an increase in the attack efficiency is observed.

Jointly considering the time and heavy utilization indicators: Next, we perform proactive migration once every $\tau = 2$ hours (high β); in addition we perform reactive migration if there is an indication of heavy memory usage i.e., the heavy bus memory bus utilization indicator issues an alert. The reactive attacker employs the bus contention based ECT. Note here that if the attacker were to use the hybrid timing based ECT, the heavy utilization indicator will never kick in and the results will be identical to that of where only the timing based indicator is used to trigger migrations; we have verified that this is the case. The results are in Table 11. We see that there are slight increases in the costs in terms of downtimes and bandwidth compared to the case with only proactive migrations with the same β (see Table 9, row 2). This is because, additional migrations are now invoked on top of proactive migrations; however, the risk in terms of attack efficiency is reduced by a factor of nearly 3. This suggests that the effectiveness of our migration guidelines; combining the indicators provides better protection with a modest increase in cost (for the same β).

Sampling overheads of memory access utilization: Fig. 14 shows the overhead of monitoring memory access times with different sampling rates. This overhead depends on factors such as the type of application, the allocated resources, and the workload. We perform experiments with ownCloud, varying the interval between the memory probes, between 0.25 and 64 minutes. Each probe test lasts for 15 seconds. Approximately 7% of the CPU cycles were consumed even with the smallest probing interval. We perform a similar experiment with Taiga. In Fig. 15, we show the average response times to web requests while varying traffic load, with a probing interval of 0.25 minutes. We see that the response times are relatively unaffected. This demonstrates that clients can monitor the risk indicators with relatively very little impact on performance in the cases of the applications we consider.

Performance with different attacker models: In the next set of experiments, we consider the three different attacker models that we described in Section 3 viz., the reactive attacker, the periodic attacker and the static attacker. We recall that an reactive attacker

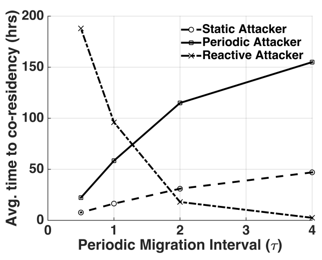

Fig. 16: Time taken by an attacker to co-reside with a victim VM (inter co-residency time).

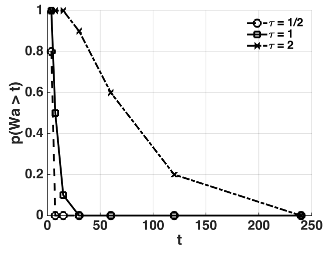

Fig. 17: Probability that the co-residency time is $> t$ for an reactive attacker; attacker uses hybrid timing based ECT.

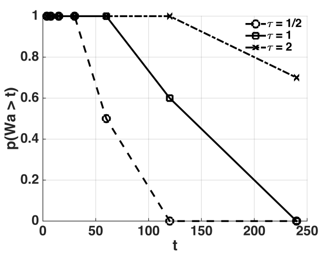

Fig. 18: Probability that the co-residency time $> t$ for a static attacker; attacker uses hybrid timing based ECT.

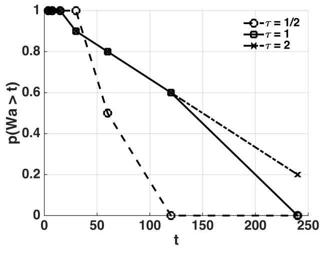

Fig. 19: Probability that the co-residency time $> t$ for a periodic attacker; attacker uses hybrid timing based ECT.

terminates and relaunches its VMs if a co-residency test fails. A periodic attacker simply chooses a period for migration (like a victim VM who migrates based on the timing indicator). A static attacker simply stays put on a single physical machine and awaits the arrival or return of the victim VM (i.e., chooses not to be migrated). As mentioned in Section 3, a periodic (static) attacker continuously checks for co-residency since it is unaware of when the victim VM is placed on its physical machine. Implicit in these experiments is the assumption that migrations are allowed on the cloud for all users (the attacker as well as the victim); the users make decisions on whether to migrate or not. We first consider the scenario where the attacker uses the hybrid timing based ECT. Later we consider the bus contention ECT.

The attacker uses the hybrid timing based ECT: In Fig. 16, we show the average time taken by an attacker to co-reside with its victim VM for different values of τ; we assume that the periodic attacker migrates its VM at the same rate as the victim. The figure captures how often an attacker co-resides with the victim (but not how long he stays with the victim). We see that frequent migrations cause the victim to come back to the same physical machine occupied by a static or periodic attack VM often. Infrequent migrations would cause the inter-coresidency times to increase. In the case of an reactive attacker, the frequent migrations hurt the time taken to get a co-residency hit (as one would expect).

Reducing the time taken to co-reside with the victim does not necessarily translate to a longer co-residency time. Let us represent the time for which the attack VM co-resides with the victim be represented by W_a. In the next three plots, we present the complementary CDF of W_a, i.e., the probability that $W_a > t$, for the three attacker models, respectively. We immediately see that frequent migrations translate to lower overall co-residency times in all cases. The reactive attacker is hurt the most by frequent migrations. With $\tau = 1/2$ hour, or 1 hour, the migrations occur even before it can successfully carry out a co-residency attempt in many cases. Combined with the fact the fact that its average time to achieve co-residency is high in these regimes (as seen in Fig. 16), it is the least effective strategy for the attacker. The static attacker gains time since it does not have to terminate and relaunch his process; we find that if a victim VM, is placed on his machine, the attacker stays with it for the period of the migration (which is to the attacker's advantage). The periodic attacker does better than the reactive attacker; however, it does not do as well as the static attacker since, once if the victim co-resides with it (moves to the physical machine it is occupying), it may be migrated itself. *In summary, the above results suggest that if users are allowed to migrate their VMs, staying put on the same physical machine is the best strategy for an attacker. Performing frequent migrations (to the extent permitted by cost) is the best strategy for the victim.*

SCA	Avg. Time	Shared Resource
Prime+Trigger+Probe [2]	341	Core
LLC Prime+Probe [26]	27	CPU
Deduplication [38]	45	Storage

TABLE 12: Average times (mins) to carry out side channel attacks.

The attacker uses the Bus Contention ECT: In Figs. 20, 21 and 22, we show the complementary CDF of W_a when the attacker uses the bus contention ECT. Our migration guidelines are in place. The value of K is 10. We see that regardless of the attacker strategy, a migration is invoked after $Kt_s = 5$ mins with high probability since the high memory bus utilizaiton indicator is triggered. Thus, the co-residency times are minimal. The value of τ has little impact since migrations are triggered in response to bus contention. The co-residency times are now much smaller from the attacker perspective, compared to the case where it used the bus contention based ECT. *The results not only demonstrate the efficacy of our migration guidelines with regards to minimizing the co-residency periods again, but also demonstrate that the bus contention ECT is much less effective than our hybrid timing based ECT from the perspective of the attacker.*

Bandwidth savings by accounting for time taken for co-residency: Nomad [5], implicitly, triggers migrations only based on the time taken for a successful information leakage attack. By ignoring the time taken to achieve co-residency, Nomad invokes migrations more frequently than necessary. In Fig. 24 we compare the bandwidth costs with Nomad with that of our approach. We consider three side channel attacks targeting information leakage in conjunction (after co-residency) viz., Last Level cache (LLC) [26], Dedup [38] and PrimeTrig [2]. On average, these attacks take 27, 45, 341 mins to be successful respectively. Nomad is assumed to invoke a migration after a duration just less than what it takes for such an attack to succeed. With our approach we migrate proactively with τ equal to the the sum of the times taken for co-residency and the information leakage attack. The results show that, by not accounting for the time taken for co-residency, Nomad increases the bandwidth costs by 30 - 150% compared to our approach.

8 Discussion: Implications on Side Channel Attacks Targeting Information Leakage

In our work, the primary metric of interest was the time for which an adversarial VM co-resides with a victim VM (we tried to minimize this time subject to some constraints on performance

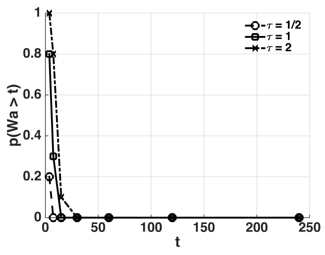

Fig. 20: Probability that the co-residency time is $> t$ for an reactive attacker; attacker uses bus contention ECT.

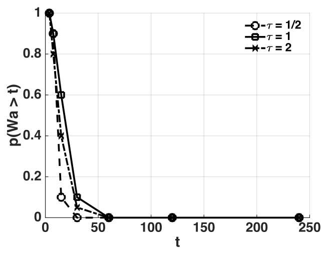

Fig. 21: Probability that the co-residency time $> t$ for a static attacker; attacker uses bus contention ECT.

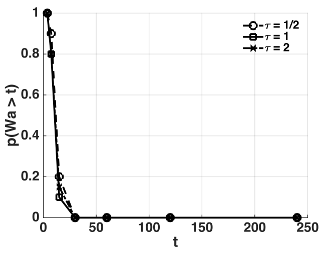

Fig. 22: Probability that the co-residency time $> t$ for a periodic attacker; attacker uses bus contention ECT.

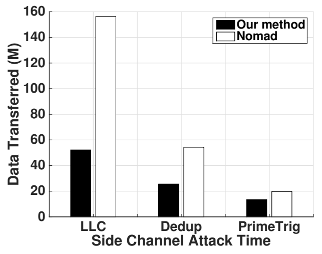

Fig. 23: A rough comparison of the bandwidth costs with Nomad.

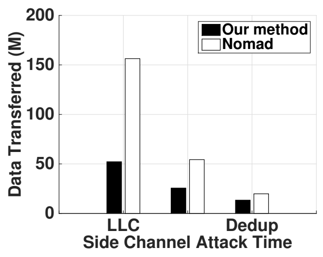

Fig. 24: Comparing bandwidth costs with Nomad.

costs). Reducing the co-residency time directly minimizes the potency of other side channel attacks that target information leakage; for such attacks to succeed an attacker will need to co-reside with the victim for the duration of the attack. In this section, we provide a discussion on how our work applies if one assumes that such attacks are carried out after co-residency is achieved; note that we only provide rough characterizations of the benefits relating to such attacks here, and a more systematic study is left for the future.

There are numerous side-channel attacks studied in the literature (see Section 2). In Table 12, we list three representative prior attacks and the average times taken to carry out the attack successfully (to achieve a desired extent of information leakage) as reported in those papers; each side channel attack targets a shared resource and uses a side-channel relating to that resource. The details can be found in the respective citations. The two things we wish to point out are the following. First, these attacks assume that an attack VM has already co-resided with a victim VM prior to lauching the attack. However, the attacker will first need to co-reside with the victim and this can take significant time, as discussed in Section 4. Second, the times taken for these "information leakage" attacks could be shorter than the time taken by an attacker to achieve co-residency on today's cloud platforms (comparing Table 7 with Table 12). Thus, if one were to use VM migration as a countermeasure against such side-channel attacks which target information leakage, one could potentially use less frequent migrations if one were to account for the time taken to achieve co-residency in addition to launching the information leakage attack (since achieving co-residency is a pre-requisite for the latter attack).

Bandwidth savings by accounting for time taken for co-residency: Nomad [5], implicitly, only considers the time taken for a successful information leakage attack, to invoke migrations. By ignoring the time taken to achieve co-residency, Nomad invokes migrations more frequently than necessary. To compute a rough estimate of the additional costs due to more frequent migrations with Nomad, we consider the three information leakage attacks listed in Table 12 and assume that they take the times reported to succeed. Nomad is assumed to invoke a migration just prior to the attack succeeding. With our approach, we migrate proactively with a value of τ equal to the sum of the average time taken for co-residency (105 mins) and the time taken for the information leakage attack as reported in the papers listed in Table 12. In Fig. 24, with this setup, we compare the bandwidth costs with Nomad with that of our migration approach. The results show that, by not accounting for the time taken for co-residency, Nomad increases the bandwidth costs by 30 - 150% compared to our approach. A more holistic implementation (with both co-residency and an information leakage attack) to validate the gains in real scenarios, is left for future work.

Decrease in information leakage rates: It is evident that VM migration can decrease information leakage rates (also shown in [5]). It has been shown that without any migration, after co-residing with a victim, an attacker can extract a secret key of length 2048 bits in 27 minutes (corresponding to a leakage rate of 1.26 bps). Our experiments show frequent VM migrations would require an attacker to perform co-residency repeatedly to get this information. Let us assume a strong attacker who can resume his attack once he again co-resides with the victim. With proactive migrations our experiments show that the attacker efficiency is $\approx$ 1 %. This means that he spends about $24 \times 60 \times 0.01 \approx 14$ minutes per day with the victim; thus, it now takes nearly two days to extract a secret key of the same length (a leakage rate of 5.7×10^{-6} bps).

Note that the effectiveness of proactive migrations (as computed above) are limited by our set up; we have only 13 physical machines. On commercial clouds, where the number of machines could be much higher, the process will be even more effective. In addition, if the attacker cannot immediately resume his attack after he co-resides with a victim VM at a later time, the information leakage rate will be further reduced; in the extreme case if the attacker has to restart his attack, the information cannot be retrieved.

Generality across cloud platforms and instance heterogeneity: We perform experiments on Amazon EC2 since it is the most popular cloud platform today [39]. Our belief is that our results and model extend to other cloud platforms. This is on the basis that most cloud providers operate their servers at very low utilizations as discussed in [36]. We acknowledge that the model does not account for heterogeneity in the types of instances that may be placed on the platform. We leave a detailed study of such heterogeneity to future work (given that it will be a non-trivial extension to this study).

9 Conclusions

In this paper, we consider an attacker who seeks to co-reside his VM with a victim VM on the cloud. Achieving such co-residency could allow the attacker to launch various side-channel attacks that target information leakage. Our goals are to (a) get a comprehensive understanding of the ways and the effectiveness with which an attacker can achieve co-residency and (b) develop migration guidelines for the victim VM that can help minimize its co-residency time with an attacker VM, given constraints on performance costs. Towards achieving (a) we consider both previous side-channel attacks and design our own (more effective) attacks towards ascertaining co-residency with a victim, and evaluate the process of co-residency extensively on Amazon's EC2. Based on these experiments, we formulate a set of migration guidelines and evaluate these extensively with different attacker strategies on our in house cloud. We show that our guidelines can limit the attacker efficiency (fraction of the time it co-resides with the victim) to about 1 % with very modest bandwidth and downtime costs (MB of bandwidth and seconds of downtime per day, per VM migrated).

Acknowledgements: The effort described in this article was partially sponsored by the U.S. Army Research Laboratory Cyber Security Collaborative Research Alliance under Cooperative ative Agreement W911NF-13-2-0045. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation hereon. We thank the associate editor Prof. Yanyong Zhang and the anonymous reviewers for their insightful comments that have helped us significantly improve the paper.

References

[1] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *USENIX Security*, 2012.

[2] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM CCS*, 2009.

[3] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *ACM Cloud Computing Security Workshop*, 2011.

[4] Ari Liberman García. *The evolution of the Cloud: the work, progress and outlook of cloud infrastructure*. PhD thesis, Massachusetts Institute of Technology, 2015.

[5] Soo-Jin Moon, Vyas Sekar, and Michael K Reiter. Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration. In *ACM CCS*, 2015.

[6] Apache CloudStack. Open Source Cloud Computing. https://goo.gl/TT2S3R, 2016.

[7] Amittai Aviram, Sen Hu, Bryan Ford, and Ramakrishna Gummadi. Determinating timing channels in compute clouds. In *ACM Cloud Computing Security Workshop*, 2010.

[8] Ralf Hund, Carsten Willems, and Thorsten Holz. Practical timing side channel attacks against kernel space aslr. In *IEEE S&P*, 2013.

[9] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Lucky 13 strikes back. In *ACM Information, Computer and Communications Security*, 2015.

[10] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-VM side channels and their use to extract private keys. In *ACM CCS*. ACM, 2012.

[11] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M Swift. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *ACM CCS*, 2012.

[12] Yuval Yarom and Katrina Falkner. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium*, pages 719–732, 2014.

[13] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. A placement vulnerability study in multi-tenant public clouds. In *USENIX Security*, 2015.

[14] Zhang Xu, Haining Wang, and Zhenyu Wu. A measurement study on co-residence threat inside the cloud. In *USENIX Security*, 2015.

[15] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *IEEE S&P*, 2011.

[16] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security*, 2001.

[17] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2):13, 2010.

[18] Steven J Murdoch and George Danezis. Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195. IEEE, 2005.

[19] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy vulnerabilities in encrypted http streams. In *International Workshop on Privacy Enhancing Technologies*, pages 1–11. Springer, 2005.

[20] Himanshu Raj, Ripal Nathuji, Abhishek Singh, and Paul England. Resource management for isolation enhanced cloud services. In *ACM Cloud Computing Security Workshop*, 2009.

[21] Peng Li, Debin Gao, and Michael K Reiter. Stopwatch: a cloud architecture for timing channel mitigation. *ACM Information and System Security (TISSEC)*, 17(2), 2014.

[22] Ashay Rane, Calvin Lin, and Mohit Tiwari. Raccoon: closing digital side-channels through obfuscated execution. In *USENIX Security*, 2015.

[23] Stephen Crane, Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz. Thwarting cache side-channel attacks through dynamic software diversity. In *NDSS*, 2015.

[24] Trusty Tahr. Ubuntu 14.04.3 LTS. http://goo.gl/mN2Ben, 2014.

[25] Amazon EC2. T2 Instance Requirements. http://goo.gl/GWMxxI, 2016.

[26] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *IEEE S&P*, 2015.

[27] Colin Percival. Cache missing for fun and profit, 2005.

[28] Frank Kargl, Joern Maier, and Michael Weber. Protecting web servers from distributed denial of service attacks. In *World Wide Web*. ACM, 2001.

[29] N Levinson. The wiener RMS (root mean square) error criterion in filter design and prediction. 1947.

[30] Per Ahlgren, Bo Jarneving, and Ronald Rousseau. Requirements for a cocitation similarity measure, with special reference to pearson's correlation coefficient. *American Society for Information Science and Technology*, 54(6), 2003.

[31] Bharat Rao and Louis Minakakis. Evolution of mobile location-based services. *ACM Communications*, 2003.

[32] Amazon EC2. AWS Security White paper. http://goo.gl/GK0sz8, 2011.

[33] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8), 2006.

[34] Fang Hao, TV Lakshman, Sarit Mukherjee, and Haoyu Song. Enhancing dynamic cloud-based services using network virtualization. In *ACM Virtualized Infrastructure Systems and Architectures*. ACM, 2009.

[35] Fangfei Chen, Katherine Guo, John Lin, and Thomas La Porta. Intra-cloud lightning: Building cdns in the cloud. In *IEEE INFOCOM*, 2012.

[36] Jeff Barr from Amazon AWS. Cloud Computing, Servier Utilization, and the Environment. https://goo.gl/xt7Kw2, 2017.

[37] Bruce Jacob, Spencer Ng, and David Wang. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.

[38] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE S&P*, 8(6), 2010.

[39] Quicbooks. The Most Popular Cloud Platforms. https://goo.gl/3L6nR6, 2017.