Malicious Co-Residency on the Cloud: Attacks and Defense

Ahmed Osama Fathy Atya*, Zhiyun Qian*, Srikanth V. Krishnamurthy*,
Thomas La Porta‡, Patrick McDaniel‡, and Lisa Marvel†
*University of California, Riverside, ‡Pennsylvania State University, †U.S. Army Research Laboratory
{afath001,zhiyunq,krish}@cs.ucr.edu, {tlp,mcdaniel}@cse.psu.edu, lisa.m.marvel.civ@mail.mil

***Abstract*—Attacker VMs try to co-reside with victim VMs on the same physical infrastructure as a precursor to launching attacks that target information leakage. VM migration is an effective countermeasure against attempts at malicious co-residency. In this paper, 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. 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.**

I. Introduction

The risk of a VM sharing a physical machine with a malicious VM on the cloud is very real [1], [2], [3]. Once an attack VM co-resides with a victim VM on a physical machine, it can launch arbitrary attacks (e.g., using side channels to achieve information leakage from the victim). Although providers continuously try to better isolate resources across VMs, new vulnerabilities are expected to emerge as hardware architectures and hypervisor technologies evolve [4].

Cloud providers assign each VM a public and a private IP address, and the latter is not exposed externally. Today, an adversary cannot determine correlations between the public and private IP addresses (towards determining co-residency), since the provider dynamically changes the mapping of public addresses to private addresses [2]. Popular prior approaches require the attacker to launch its VMs, and use side-channels to create congestion on a shared resource; the attacker then uses a side channel to check for the presence/absence of congestion on that resource and thereby ascertains co-residency. Upon failures, the attacker has to 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 goals. First, we seek an in depth understanding of the ways and the effectiveness with which an attacker can achieve co-residency with a victim VM. Towards this, we undertake an extensive experimental effort on Amazon's EC2 cloud, to understand the side channels that an attacker can use to ascertain co-residency with a victim VM. En route, we discover new stealthy and highly effective timing based side channels that can be used 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 the time that a victim VM has resided on a host machine and the very 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 over five months to develop a comprehensive understanding of the efficacy of an adversary in successfully co-residing its VM with a victim's VM. We also build a simple model that provides rough estimates of how long it takes for an attacker to successfully co-reside with a victim.
- We discover a set of new highly effective timing based side channels that can be used by an attacker to determine co-residency with a targeted victim VM. Our side channels provide the highest accuracy in ascertaining co-residency as compared to previously proposed side channel tests ($\approx$ 86 %), but with lower false positive rates. In addition, we believe that they are much harder to detect 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 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 %.

II. 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. There are several side channel attacks on cloud tenants that have been previously studied (e.g., [2], [7], [8]). Based on the attack, the time taken to successfully extract information ranges from the order of minutes to hours.

Co-residency with a victim process: For almost all side-channel attacks reported, an attack VM will need to co-reside with the victim VM on the same physical machine. The attacker will need to use some kind of side channel to ascertain if its VM has co-resided with a victim (e.g.,[9]). However, prior efforts do not provide a comprehensive understanding of how effective the proposed side-channels (for ascertaining co-residency) 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, (e.g.,[9], [10]) although hard, might be useful in the short term. However, placement algorithms are likely to dynamically change over time [10]. Further, cloud providers have ensured that many co-residency checks proposed much earlier (e.g., [2], [11]) are no longer feasible (we have also experimentally verified that this is the case). 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 but they are quite different (e.g., [12]).

Defending side-channel attacks: Efforts such as [13] introduce random delays while accessing a resource to thwart timing based side channel attacks targeting information leakage. [14] and [15] employ software level defense mechanisms as countermeasures against cache based side channel attacks. Although such methods can defend against known side channel attacks, there could exist other attacks, currently 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 thwart the attacker's ability to launch such attacks.

VM migration to mitigate side-channel attacks: VM migration and placement with Nomad, is proposed to counter side-channel attacks targeting information leakage in [5]. However, it is assumed that (i) the attacker has successfully co-resided with the victim and (ii) 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 occur when VMs are migrated for coping with side-channel attacks. We make no assumptions on what the provider will do in terms of placement of VMs. Unlike in Nomad, we account for the fact that an attacker takes an additional time to successfully co-reside with a victim in addition to the time it takes for an information leakage attack, while making migration decisions; this can reduce the bandwidth and downtime costs from such migrations. The users can configure when to migrate their VMs based on their risk averseness and the costs they are willing to bear.

III. Threat model

We assume that an attacker seeks to co-reside its VM on the same physical machine as a certain targeted victim VM for as long as possible. First we consider an attacker who launches a set of VMs, repeatedly if needed, to achieve its goal. We assume that the attacker has no knowledge or control over the cloud provider's policies for VM placement (as with Amazon's EC2). We call such an attacker an "reactive attacker"; this scenario reflects what the attacker can do on today's commercial clouds.

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 I: Instance Type Comparison

Next, we consider an attacker who can migrate its VM, if user driven migrations are allowed. We assume that the provider does not unilaterally perform migrations for thwarting side-channel attacks (without user requests) like in [5], since this may cause downtimes which some users may find undesirable. However, the provider may still invoke migrations for performance reasons (e.g., load balancing). An attacker VM could simply choose to stay put on its physical machine assuming that it will eventually co-reside with the migrating target VM. We call such an attacker a "static attacker". Finally, we also consider an attacker that migrates periodically. We call such an attacker, a "periodic" attacker. In both of the above cases, we assume that the attacker continuously checks for co-residency, since the victim could now at any point, migrate to the machine on which its VM resides. Note that these 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 VII.

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 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; however, we provide some rough estimates of how such attacks are affected by our approach in Section VII. For simplicity, 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).

IV. 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. As a major contribution here, we design new timing based co-residency tests that are stealthy and yet, very effective.

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 offer high fidelity and can serve as ground truth, but cannot necessarily be used by an external adversary. 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. The accuracy of the ECTs (which represent adversarial operations in a real setting) is assessed by comparing the result with that of a ICT.

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 [16]. 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 [9], [3], [11] and tested. A key contribution we make is the design of new, very effective, timing-based co-residency tests.

A victim VM hosts one of 3 very different services viz., Taiga, ownCloud and MediaServer; thus, the co-residency tests in the three cases, for identical set ups, take different times. Taiga is a project manager software with a mix of CPU, disk (frequent), and memory workloads. ownCloud is a file hosting service (resembles Dropbox) with memory and disk intensive workloads. MediaServer is a wiki-page server that involves a mix of CPU, disk (rare), and memory workloads. We use different types of VMs; their specifications are summarized in Table I (we use the Amazon EC2 jargon [17]).

In each experiment, as mentioned, the attacker and the victim have 20 VMs each unless otherwise specified (we have done experiments with other numbers and the behavioral results are similar; we discuss some of these later). We do several optimizations to reduce the time taken to perform an exhaustive set of experiments (e.g., for each attack VM, we perform a co-residency test with regards to a shared resource with regards to all the victim VMs in one shot). We omit the details of these optimizations in the interest of space.

After launching its VMs, the attacker performs what is called a "co-residency test" as mentioned earlier. A successful co-residency test would indicate that with high probability the attack and the victim VM co-reside on the same physical machine. For every failed attempt at co-residency (the co-residency test fails), the attacker must terminate its VMs and then re-launch them in an attempt to successfully co-reside with his target victim. The time taken to launch and terminate these VMs are denoted by t_l and t_d, respectively. These times contribute to the overall time taken by an attacker to successfully co-reside with the victim VM. The overall time taken to achieve co-residency is denoted by t_c. This depends on the type of co-residency test that is conducted, and the service provided by the victim VM.

A. New timing based ECTs

As a major contribution, we propose a set of new, stealthy yet very effective timing-based ECTs that can be used to ascertain co-residency by an external attacker.

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 DoS attacks [18]. However, it is reasonable to assume that packets 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. The attacker, from an external vantage point, thus probes both its own VM and the victim's VM almost simultaneously; the expectation is that the delays experienced by such probe packets will have consistent (similar) temporal variations, if the two VMs are on the same physical machine.

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 ≈ 10 msec) we (as the attacker) probe at coarse time intervals (i.e., the probing period is set to ≥ 100 msec). With this, we can associate exactly one response from each of the two VMs (the attack and the victim VM), in each probing iteration. If the two VMs are on the same physical machine the delays observed between the responses from the VMs are much less than the probing period; unless the physical machine load rapidly changes (very rare), 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 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 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 III), 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) [19] and the Pearson Coefficient (PeC) [19].

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 (slight time shifts are experienced).

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 IV-C, the test declares a success (the two VMs co-reside) if the MSE is low (< 0.15) and the PeC is high (≥ 0.8).

Timing based ECT with multiple observers: In this test, the attacker uses multiple observers (at different vantage points); each observer examines the RTTs between the the VMs in question and only records the "minimum RTTs" observed with respect to the target VMs in 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 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 $\{\rho'_1, \rho'_2, \rho'_3\}$

Resource	type	# Avg. VMs	# of hits	Percentage
Bus	ICT	108000	22666	21%
CPU	ICT	98000	19540	20%
Core	ICT	98000	8820	9%

TABLE II: Hit rates with different ICTs

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 distances coincide. If the two signatures are very similar (but don't exactly match) it is very likely that the VMs are on the same physical machine. (in our experiments described in Section IV-C, 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. It also depends on 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 two tests described above are combined to improve the accuracy of the co-residency determination and decrease the time required to get accurate results. Multiple observers are again used. However, each observer analyzes the similarity between the time series of the observed RTTs (as in the first test) 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."

B. Implementation of prior co-residency tests

We implement and test (a) previously proposed ICTs and (b) the only prior ECT that is still effective on EC2. The former tests serve as ground truth to evaluate the ECTs, and the latter serves as a benchmark for comparing the effectiveness of our newly proposed timing tests. In this subsection, we briefly summarize these tests; details can be found in the citations.

Bus contention based ICT and ECT: Bus contention tests were designed and evaluated in [1] and [9]. The key idea in the ICT is for the attack VM to create contention on the memory bus and cause it to lock. If the victim VM observes a significant increase in the memory access time (around 3X), the conclusion is that the the two VMs share the same machine. With the ECT, the attacker uses an external presence which engineers a set of external requests to the victim VM, which causes the latter to access the memory bus. Internally, the attack VM locks the memory bus as in the case of the ICT. If the response times to the engineered requests increase compared to a case where there is no locking of the memory bus, the attacker infers that the two VMs co-reside on the same physical machine; if there is no significant increase, the two VMs are likely to reside on different physical machines.

LLC based ICT: In this test the attack VM creates contention on the LLC and if this increases the LLC access time ($\approx$ 2X) for the victim VM, it concludes that it shares a CPU with the victim. To create contention, the attacker VM allocates 1 or more GB of memory and regularly reads and writes in multiple page size increments (to ensure page misses for the victim VM). 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.

ICT with L1 cache: Two VMs that share the same CPU do not necessarily share the same core. The attacker can cause contention on the L1 cache to determine if it shares a core with the victim [20]. 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 will as a consequence experience misses and thus see an increase in access times (1.5 to 3X).

C. 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 tests.

Launching and Termination times: Launch and termination times are part of the overheads incurred 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) [21].

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 ICTs on Amazon's EC2. Table II shows the hit rates with the ICTs discussed earlier. We find that in approximately 21% of the cases the bus based ICT indicates that 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 present the measured accuracies with the ECTs in Tables III-VI. We report the results with 3 different services. Each service was tested for 5 consecutive days. We approximately conducted 50 runs per day. We used the bus contention based ICT to establish the ground truth (that test was also conducted and its result was considered the truth). This is because, first, the work in [9] shows that this test has very high accuracy in determining the co-residency of two VMs. 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.

For each ECT we measure the number of (i) true positives (TP), (ii) true negatives (TN), (iii) false positives (FP) and (iv) false negatives (FN). We compute the true positive rate (aka *sensitivity* [22]) to be $\frac{TP}{TP+FN}$. Similarly, the true negative rate (aka *specificity* [22]) 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 the 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 suggests that for some workloads, the bus contention ECT may not be

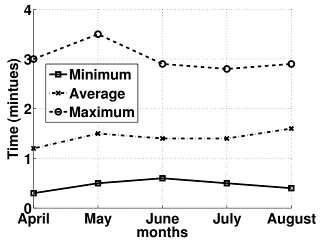

Fig. 1: Launch times over months

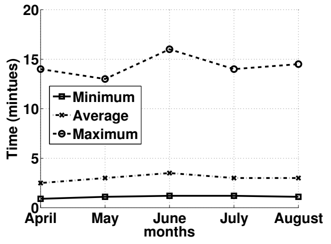

Fig. 2: Termination times over months

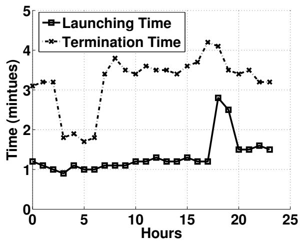

Fig. 3: Avg. launch and termination times over days

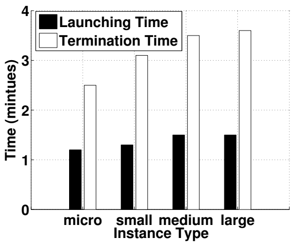

Fig. 4: Avg. launch and termination times for different VM sizes.

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

TABLE III: Sensitivity/Specificity with the behavioral ECT.

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

TABLE IV: Sensitivity/Specificity with the signature ECT.

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

TABLE V: Sensitivity/Specificity with the hybrid ECT.

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

TABLE VI: Sensitivity/Specificity with the bus ECT.

effective; the hybrid timing tests seem to work well with all the workloads we considered. Finally, we argue that the RTT based methods are less invasive and thus harder to detect.

A microscopic view: In Figs. 5 and 6, we show snapshots of the normalized time series at a single observer (using the hybrid timing ECT), when we have a mismatch and a match, respectively. 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 because of the two VMs being on the same rack but not on the same physical machine.

Machines on the same rack: Unfortunately, we cannot categorically identify cases where two physical machines are on the same rack on EC2. To examine the impact of cases where the attacker and the victim VMs are on different physical machines which are on the same rack (share the same ToR switch), we perform experiments on our in house cloud (details later). The VMs are placed on the same physical machine or on different machines that share the same rack, and the timing tests are carried out from vantage points from within EC2. We find that the false positive rates are about 12.4 % and the false negative rates are about 0.3 %. This demonstrates the efficacy of our timing tests in filtering out cases where the VMs are on different machines on the same rack.

Properly configuring the ECT tests: The work in [9] discusses how to properly configure the parameters for an effective bus contention ECT (we follow the same approach). It is obvious that increasing the number of observers, gathering a larger number of samples, and low sampling rates improve the fidelity of the timing tests. We find that choosing between 15 and 20 observers, an observation period of about 20 minutes, and a 200 msec period between samples yields accuracies of over 80 %. We find that further increasing the number of observers, getting more samples or decreasing the sampling interval does not provide significant additional benefits (we reach a point of diminishing returns). We do not present detailed results here due to space constraints.

Experimentally computed times for co-residency: In Table VII 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 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 Section VI.

V. Modeling co-residency times

The time taken for successful co-residency depends on (i) the number of VMs the victim has on the cloud (e.g., a web provider may have multiple replicas of his web server running [23]) (ii) the number of attack VMs 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 VMs 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 develop a simple model that allows us to effectively estimate this time.

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. Let $p_c(u, m)$ be the probability of successful co-residency with the m^{th} victim VM replica in an attempt. 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

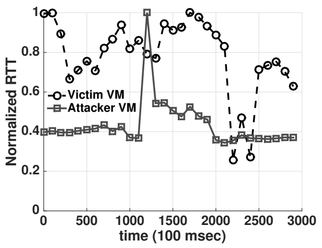

Fig. 5: Time series snapshots upon a miss

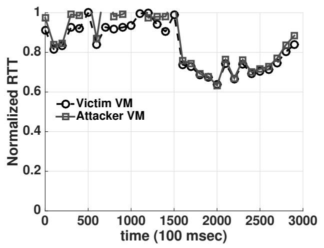

Fig. 6: Time series snapshots upon a hit

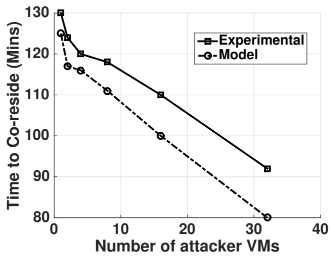

Fig. 7: Avg. time to co-reside with any of 8 victim VMs.

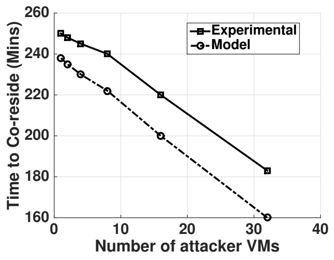

Fig. 8: Avg. time to co-reside with any of 4 victim VMs.

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 VII: Co-residency times: average and percentiles (mins)

attempt. Then, the probability of co-residing an attack process with any of the replica VMs is $p_c(u) = \sum_m p_c(u,m)$.

It is hard to determine $p_c(u)$ without knowing the placement policy of the provider. 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 unique 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 [21]). For example, a provider may place the VMs of a customer on different physical machines for reliability (robustness to machine failures). With this policy, it is easy to verify that the probability $p_c(u,m)$ can be bounded by:

$$p_c(u,m) = \frac{1}{\mathcal{N}} + \frac{1}{\mathcal{N}-1} \dots + \frac{1}{\mathcal{N}-\mathcal{A}} \leq \frac{\mathcal{A}}{\mathcal{N}-\mathcal{A}} \tag{4}$$

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: We compare the co-residency estimates using our model, with that from experiments on EC2. Figs. 7 and 8 depict the times taken to co-reside with any of the victim replicas (we consider 8 and 4 replicas). 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, our model yields relatively good (but rough) estimates of the co-residency times.

VI. Determining when to migrate

Risk indicators: Migrations should be invoked when a VM is at a high risk of being attacked. Unfortunately, without knowing the capabilities of an adversary it is hard to quantify risk. To assess risk, we consider two 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 a VM spends on the same physical machine, the more probable it is that an adversary will successfully co-reside with it 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 because of this ongoing 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. With the timing based ECTs, note that the second risk indicator is not useful.

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 ECT is underway. We sample the utilization $\mathcal{U}$ of the bus periodically at intervals t_s. If on machine m, $\mathcal{U}$ is greater than a threshold for a specific sample (say j), we set a boolean variable associated $S(m,j)$ to 1 (and 0 otherwise). The risk indicator $V(m,K)$ is obtained by jointly considering (say) K consecutive samples. Specifically,

$$V(m,K) = \sum_{j=k}^{k+K} S(m,j), \tag{5}$$

for any k. If $V(m,K) = K$, then the bus experienced a high $\mathcal{U}$ for the K consecutive samples; this suggests 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 [24]. One could set the threshold to be a certain fraction of the specified maximum value based on the user's risk averseness and the costs she is willing to bear. In our work, we conduct extensive empirical experiments based on which, 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 [9]. We assume that this

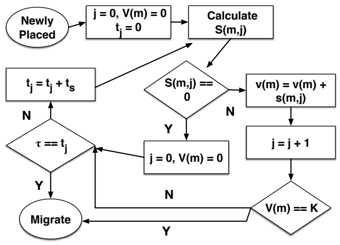

Fig. 9: Migration Guidelines

threshold is fixed and other parameters are tuned to determine when VMs are to be migrated as discussed next.

Migration guidelines: Next, we formulate guidelines for migration, based on the risk indicators. We assume that the provider does not unilaterally migrate a user's VM to reduce risk (since some users may be unwilling to experience high performance costs towards lowering risk). Instead, the provider monitors the bus utilization at preset time intervals t_s, and based on her preferences (discussed later), migrates her VMs.

Our guidelines are characterized using the flow chart in Fig. 9. A user's virtual machine enters 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.

User preferences: The 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. If the user continuously observes bus contention, her VM is migrated every Kt_s seconds. Here, if the size of her VM is X MB, the bandwidth cost will be $\frac{8X}{Kt_s}$; she will experience a downtime every Kt_s s. This corresponds to her highest costs (typically $\tau >> Kt_s$). If there is no attack or if there is a timing based ECT, migrations are triggered every τ seconds. The user will experience bandwidth costs of $\frac{8X}{\tau}$ Mbps and a downtime every τ s. This is her best case in terms of cost. If there is a bus-contention based ECT, the times between the migration instances (and costs) will be somewhere in between (and not excluding) the best and the worst case scenarios.

VII. Evaluations

In this section, we experimentally evaluate our VM migration guidelines in terms of reduction in risk to malicious co-residency and the incurred costs. Unfortunately, Amazon EC2 or other cloud providers do not yet offer a service wherein a user can configure VM migrations for the purposes of security; therefore, our evaluations are on an in house private cloud.

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. 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 showcasing the effectiveness of our VM migration guidelines. On commercial clouds, we believe that migration will be even more effective (because of scale). All experiments were conducted over periods of 15 days.

Metrics: 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.

Evaluation results: Next, we present our results. Migration costs are averages over 24 hours unless specified otherwise.

Evaluations with a reactive attacker: In the first set of results we consider the reactive attacker model described in Section III; this is what the attacker can do to-day on commercial clouds.

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. The attacker uses the hybrid timing based ECT strategy. Since the bus contention utilization indicator is not used, the results are very similar when the attacker used the bus contention ECT. We consider two values of τ viz., 60 and 120 minutes. These times correspond to approximately 60 % and 110 % of the expected time it takes for an attacker to achieve co-residency (from the results in Section IV). A smaller τ results in lower risk but higher costs in terms of bandwidth consumed and downtimes for the user. The victim VMs are either Taiga, Mediaserver or ownCloud.

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). We summarize the costs (downtimes and the traffic generated by migration) and the attack efficiency in Tables VIII and IX. As expected, the traffic volumes and the average downtimes are doubled if the migration periodicity doubles. 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. 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: Next, we consider the case where the bus contention ECT is used by the attacker. We only migrate a VM if the heavy bus contention

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

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

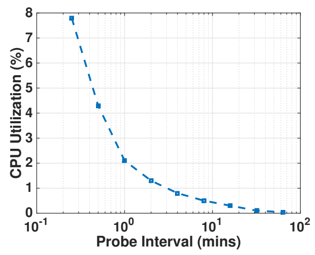

Fig. 10: The CPU utilization costs with memory probing.

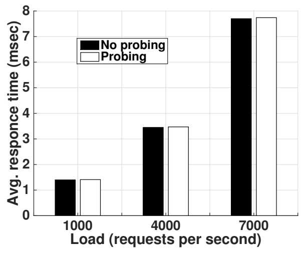

Fig. 11: The average response times with and without memory probes.

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. Table X summarizes the results. The costs of migration decrease significantly compared to the case where migration is proactively performed based on the time indicator (recall Tables VIII and IX). 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 seen.

Jointly considering the two risk indicators: Next, we perform proactive migration once every $\tau = 2$ hours, but in addition we perform reactive migration if the heavy bus memory bus utilization indicator issues an alert. The attacker uses the bus contention based ECT. Note here that if the attacker were to use the hybrid timing ECT, the heavy utilization indicator will never kick in and the results will be identical to where only the timing based indicator was used. The results depicted in Table XI, show that there are slight increases in downtimes and bandwidth compared to the case with only proactive migrations with the same τ (see Table IX, row 2). This is because, additional migrations are now reactively invoked; however, the risk in terms of attack efficiency is reduced by a factor of ≈ 3. This suggests combining the indicators provides better protection with a modest increase in cost.

Sampling overheads of memory access utilization: Fig. 10 shows the overhead of monitoring memory access times with different sampling rates. 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. 11, 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 with the applications we consider.

Performance with different attacker models: We next consider the three different attacker models described in Section III viz., the reactive attacker, the periodic attacker and the static attacker. As mentioned in Section III, a periodic or a static attacker continuously checks for co-residency since it is unaware of when the victim VM is placed on its physical machine. These experiments apply when migrations are allowed on the cloud for all users (the attacker and the victim); the users make decisions on configuring migrations.

The attacker uses the hybrid timing based ECT: In Fig. 12, 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 W_a represent the time for which the attack VM co-resides with the victim VM. In the next three plots, we present the complementary CDF of W_a (P($W_a > t$)), for the three attacker models, respectively. We see that more frequent migrations result in 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 that its average time to achieve co-residency is high (as seen in Fig. 12), it is the least effective strategy for the attacker. The static attacker gains time since it does not have to terminate and relaunch its process; we find that if a victim VM, is placed on its 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 the victim is placed on its current physical machine, 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 (as permitted by cost) is the best strategy for the victim.*

The attacker uses the Bus Contention ECT: In Figs. 16, 17 and 18, we show the complementary CDF of W_a when the

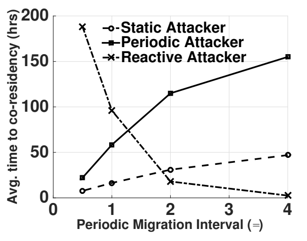

Fig. 12: Inter co-residency times with different strategies.

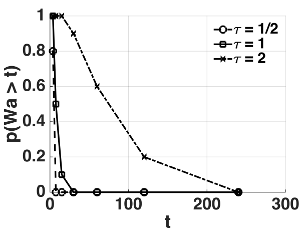

Fig. 13: Probability $W_a > t$ for a reactive attacker using the hybrid timing ECT.

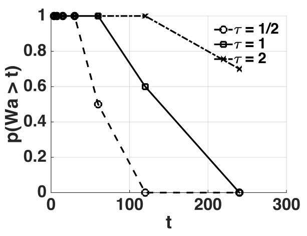

Fig. 14: Probability $W_a > t$ for a static attacker using the hybrid timing ECT.

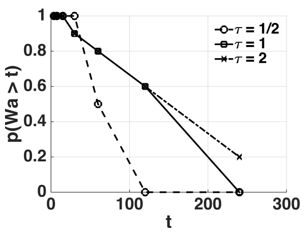

Fig. 15: Probability $W_a > t$ for a periodic attacker using the hybrid timing ECT.

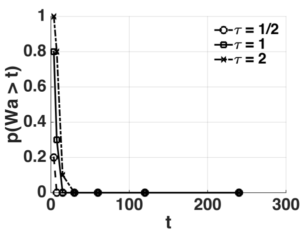

Fig. 16: Probability $W_a > t$ for a reactive attacker using the bus ECT.

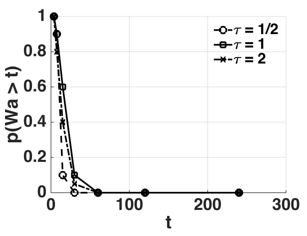

Fig. 17: Probability $W_a > t$ for a static attacker using the bus ECT.

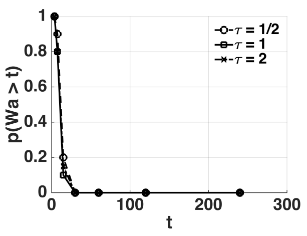

Fig. 18: Probability $W_a > t$ for a periodic attacker using the bus ECT.

attacker uses the bus contention ECT. Our migration guidelines are in place. 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 utilization indicator is triggered. Thus, the co-residency times are minimal. 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, but also demonstrate that the bus contention ECT is much less effective than our hybrid timing based ECT from the attacker's perspective.*

VIII. Conclusions

Achieving co-residency with a victim VM on the cloud allows an attacker to launch various side-channel attacks that target information leakage. In this paper, we first perform an extensive experimental study on Amazon EC2 to obtain an in depth understanding of the ways and the effectiveness with which an attacker can achieve co-residency. We also develop a set of stealthy attacks to achieve co-residency in the process. Subsequently, choosing VM migration as a countermeasure strategy, we develop migration guidelines that can help a victim minimize its co-residency time with an attacker VM, given constraints on performance costs. We evaluate our guidelines extensively with different attacker strategies on our in house cloud and show that they can limit the fraction of the time an attack VM co-resides with its victim to about 1 %, with very modest bandwidth and downtime costs.

Acknowledgment: : The effort described in this article was partially sponsored by the U.S. Army Research Laboratory Cyber Security Collaborative Research Alliance under Cooperative 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.

References

[1] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *USENIX Security*, 2012.
[2] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM CCS*, 2009.
[3] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An exploration of L2 cache covert channels in virtualized environments. In *ACM CCS Workshop*, 2011.
[4] Ari Liberman García. *The evolution of the Cloud: the work, progress and outlook of cloud infrastructure*. PhD thesis, MIT, 2015.
[5] S. Moon, V. Sekar, and M. 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] R. Hund, C. Willems, and T. Holz. Practical timing side channel attacks against kernel space ASLR. In *IEEE S&P*, 2013.
[8] Y. Zhang, A. Juels, M. Reiter, and T. Ristenpart. Cross-VM side channels and their use to extract private keys. In *ACM CCS*, 2012.
[9] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. Swift. A placement vulnerability study in multi-tenant public clouds. In *USENIX Security*, 2015.
[10] Z. Xu, H. Wang, and Z. Wu. A measurement study on co-residence threat inside the cloud. In *USENIX Security*, 2015.
[11] Y. Zhang, A. Juels, A. Oprea, and M. K Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *IEEE S&P*, 2011.
[12] D. X. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *USENIX Security*, 2001.
[13] P. Li, D. Gao, and M Reiter. Stopwatch: a cloud architecture for timing channel mitigation. *ACM TISSEC*, 2014.
[14] A. Rane, C. Lin, and M. Tiwari. Raccoon: closing digital side-channels through obfuscated execution. In *USENIX Security*, 2015.
[15] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz. Thwarting cache side-channel attacks through dynamic software diversity. In *NDSS*, 2015.
[16] Trusty Tahr. Ubuntu 14.04.3 LTS. http://goo.gl/mN2Ben, 2014.
[17] Amazon EC2. T2 Instance Requirements. http://goo.gl/GWMxxI, 2016.
[18] F. Kargl, J. Maier, and M. Weber. Protecting web servers from distributed denial of service attacks. In *World Wide Web*. ACM, 2001.
[19] Sarah Boslaugh. *Statistics in a nutshell*. "O'Reilly Media, Inc.", 2012.
[20] C. Percival. Cache missing for fun and profit, 2005.
[21] Amazon EC2. AWS Security White paper. http://goo.gl/GK0sz8, 2011.
[22] T. Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8), 2006.
[23] F. Chen, K. Guo, J. Lin, and T. La Porta. Intra-cloud lightning: Building CDNs in the cloud. In *IEEE INFOCOM*, 2012.
[24] B. Jacob, S. Ng, and D. Wang. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.