Efficient, Adversarial Neighbor Discovery using Logical Channels on Microsoft Azure

Mehmet Sinan İnci, Gorka Irazoqui, Thomas Eisenbarth, Berk Sunar
Worcester Polytechnic Institute, Worcester, MA, USA
Email:{msinci, girazoki, teisenbarth, sunar}@wpi.edu

***Abstract*—We introduce an effective technique that exploits *logical channels* for malicious co-location and target identification on Microsoft Azure cloud instances. Specifically, we employ two co-location scenarios: targeted co-location with a specific victim or co-location with subsequent identification of victims of interest. We develop a novel, noise-resistant co-location detection method through the network channel that provides fast, reliable results with no cooperation from the victim. Also, our method does not require access to the victim instance neither as a legitimate user nor a malicious attacker.**

The efficacy of the proposed technique enables practical QoS degradation attacks which are easy and cheap to implement yet hard to discover. The slightest performance degradation in web interfaces or time critical applications can result in significant financial losses. To this end, we show that once co-located, a malicious instance can use memory bus locking to render the victim server unusable to the customers. This work underlines the need for cloud service providers to apply stronger isolation techniques.

Keywords*—*Microsoft Azure, Logical Channels, Denial-of-Service, Co-location in Cloud, Memory Bus locking

I. Introduction

Cloud service providers provide on-demand scaling and promise cost savings as a result of shared resources. In most clouds, tenants share physical hardware, i.e. services of many users run on the same underlying physical machine. In order to ensure security, Cloud Service Providers (CSPs) promise strong isolation between the instances: each instance runs in its own virtual machine (VM), managed by a CSP-controlled hypervisor. The hypervisor ensures logical isolation as well as fair distribution and access to system resources. With increasing adoption of the cloud, researchers have started to question the strong isolation claims of widely used hypervisors, and indeed a number of weaknesses have been found. For instance, hardware side-channels are known for enabling cross-VM information extraction, including cryptographic keys [27], [13], [25] and other sensitive user information [28], [17]. All of these attacks require co-location as a necessary first step: The adversary can only extract sensitive information if it manages to place instances on the same physical machine as the victim. CSPs have tried to make co-location difficult by employing undisclosed and unpredictable placement algorithms. However, in the early days of cloud adoption, Ristenpart et al. showed that *logical side-channels*, i.e. information of the internal network topology of the Amazon cloud, was sufficient to deduce co-location [18].

It is widely believed that CSPs have since made co-location detection difficult by closing logical channels. Co-location detection is still feasible by using hardware side-channels, i.e. the memory bus channel [10], [21], [24] or the cache side-channel [11]. However, these side-channel based co-location tests are much harder to exploit: they take more time to run and use noisy channels, resulting in false positives and negatives, and do not scale as well as logical side-channels.

Either way, once co-location is achieved, an attacker can perform a wide range of attacks. While the previously mentioned information extraction attacks present a significant threat, they are applicable only to security-critical services, and—due to recent updates in major cryptographic libraries—mainly affect outdated implementations. An alternative way of attacking cloud instances is by degrading their performance in terms of latency and responsiveness and thus their Quality of Service (QoS). Such *QoS attacks* can affect any cloud service, regardless of whether it has a security component. QoS attacks are easier to perform and are very difficult to prevent from the client-side—while crypto-attacks are mitigated by simply updating crypto libraries to the patched versions.

On e-commerce sites like Amazon, eBay or Macy's, even the slightest delay in the user interface leads to a serious loss of customers and revenue. According to [9], Amazon loses 1% of sales for every 100 ms latency. In addition, a number of application domains, e.g. finance and streaming media, on-line gaming are extremely sensitive to latency. In high frequency stock trading, for instance, milliseconds of head start can translate into great financial gains [6]. In short, the responsiveness of a web page or an application server is crucial for the quality of the service provided.

A. Our Contribution

This work presents a novel and highly efficient co-location detection method on Microsoft Azure compute cloud. Note that Azure is the fastest growing CSP, and only second to AWS in current business size, with a 16% IaaS market share [20]. By utilizing a combination of *logical channels*, we present an extremely fast, cheap and reliable detection method. The new method does not need any collaboration between co-located instances and is able to identify known targets in only seconds. More importantly, our method also allows the attacker to identify whom he is co-located with. That is, rather than launching instances until the attacker happens to succeed in co-locating with a specific target[1], he can first launch instances, check with whom the instances are co-located with, and then decide on potential actions and victims. Identifying unknown targets takes only minutes and has a *100% success rate*.

[1]Co-location may not even be possible if there are no open slots on the target machine for an additional VM.

We further show the efficacy of QoS attacks of a co-located victim. QoS attacks are enabled due to the lack of isolation in the memory channel. Locking the memory bus significantly degrades bandwidth and increases latency of many applications, especially in memory-intensive web applications. In summary, this work:

- **Fast Co-location:** Presents the fastest and cheapest co-location detection mechanism that has been implemented in Microsoft Azure utilizing the combination of pure logical network channels.
- **Stealthy and Precise:** The presented co-location detection mechanism does not suffer from false positives and is very difficult to detect.
- **Target Identification:** Utilizes the obtained neighbor network parameter information to develop a methodology that determines the identity of the co-located neighbors.
- **QoS Attack:** Demonstrates that non-virtualized access to the memory bus enables QoS attacks which, unlike information extraction, can impact virtually all services on a cloud server, are easy to perform, and yet can have strong impact.

While our co-location results are obtained only on Microsoft Azure, we believe that it is highly likely that other CSPs suffer from the same or similar placement vulnerabilities. As for the described QoS attack, it works across all CSPs not monitoring or blocking memory bus locking instructions. So far we are not aware of any CSPs planning to take this step, since blocking atomic instructions will make concurrency and cache coherency impossible.

The rest of the paper is organized as follows: Section II describes the background knowledge, Section III explains the approach to map the Microsoft Azure network and Section IV describes the co-location attack scenarios. Performance degradation results are shown in Section V while countermeasures and conclusion are presented in Sections VI and VIII respectively.

II. Background

A. IaaS Public Clouds

IaaS public clouds are a service in which a third party provider owns hardware and software resources that are provided to end users in the form of virtualized hardware. In contrast to PaaS and SaaS clouds where the user is provided with the software and a software development platform respectively, IaaS clouds deliver operating system, servers, network and storage on-demand in the form of a Virtual Machine (VM) instance. In fact, many PaaS and SaaS services run behind an IaaS instance. The host providing the physical resources is in charge of the physical machine maintenance and backup while the user manages only his own service.

In order to maximize profit and reduce cost, IaaS cloud providers host several instances that belong to different customers on the same physical server. While this approach minimizes costs, it can also lead to security breaches if the isolation between instances is not perfect. To make targeted attacks harder to perform, CSPs make it difficult for tenants to know which instances they are co-located with. This placement is usually managed by *placement algorithms* and use several launch parameters such as the time of the day, the region in which the instance will be opened and the instance type. As an example, instances that are created within a short time period are more likely to be placed in the same physical server, due to a phenomenon called *parallel placement locality*.

After the instance launch, CSP assign two IP addresses to each instance, i.e. a *public* and a *private* IP address. While the public IP address is utilized to establish communication with the outside world, the private, i.e. the internal IP address is utilized for intra-network communications. The instance is also assigned a unique virtualized Media Access Control (MAC) address within the internal network to establish communications in the data link layer. Further the instances are assigned an SSH port (either the port 22 or a custom one) to give users access to their instances. Finally, the hypervisor is also assigned an IP, called the first hop IP, and is in charge of implementing filtering protocols with intra-instance requests to prevent the generation of spoofed traffic [1].

The above described network parameters (among others) are usually referred to as *logical channels* that can be exploited to infer information about the placement of the instance. Ideally these parameters should give no information about the identities of the co-located neighbors. For instance, the first hop IP address does not reveal location information since `traceroute` requests are filtered by the hypervisor. However, by combining more than one of these logical channels we show that co-location can still be inferred in Microsoft Azure and that we can also reveal the identity of our co-located neighbors.

B. Memory Bus Locking

In this study, we use the memory bus locking technique for two purposes: co-location verification and the performance degradation. Former is to verify the co-location between instances discovered via logical channels. And the latter is to degrade the performance of the co-located neighbor by frequently locking the memory bus and slowing down memory accesses.

The memory bus lock is a cache coherency feature employed in CPUs. Whenever an atomic operation such as XADDL or CMPXCHG is issued, the CPU locks the cache line that holds the operated data. This locking mechanism prevents Read After Write (RAW) hazards and maintains cache coherency. On the other hand, when the operated memory block spans multiple cache lines, the CPU cannot lock the adjacent cache lines and instead issues a bus lock. Nowadays however, CPUs have multiple memory channels and the locking of a single memory channel is not sufficient since other channels can be active. Also considering the multi-socket systems with inter-CPU connections like Quick Path Interconnect (QPI), the problem gets even more complicated. To cope with this complex system, newer Intel CPUs use pipeline flushing.

The pipeline flushing drops all the memory operations in the pipeline hence ensuring that no instruction will operate on the protected data. This however results in a significant performance degradation to the system. The degradation is

even more apparent for applications that rely heavily on the memory. In our test platforms, we measured memory access times during the lock as high as 4000 clock cycles as opposed to regular 250-300 cycles. As for degradation of applications, we have observed varying levels of performance degradation, as given in detail in Section V.

III. Co-location Deduction via Network Parameters

Our goal is to find the easiest, cheapest and fastest method to determine co-location in Microsoft Azure with high accuracy, without having to take noise-prone measurements (cache or memory) that can be detected easily by the hypervisor. To this aim, we analyze the network parameters assigned by Microsoft to our instances and deduce physical co-location. To the best of our knowledge logical channels to determine co-location have never been studied in Microsoft Azure. We believe that, this study is crucial for an IaaS cloud that has 16% of the cloud computing market share and is the second most used IaaS cloud. In short, we aim to discover whether Microsoft Azure provides any information that will help us determine co-location just by looking at logical channels, without having to take side-channel measurements.

With the goal of co-location detection through logical channels in mind, we create 4 different accounts in Microsoft Azure, that will be referred as A, B, C, D. In each of these accounts, we launch 20 of the cheap, *ExtraSmall* instances using the `manage.windowsazure.com` portal. For consecutive experiments, we have used the Azure Command Line Interface (CLI) to launch same type of instances in an automated fashion to save time. Our *ExtraSmall* instances had the following specifications; 1 vCPU core, 0.75 GB of RAM, low net bandwidth and 19 GBs of total disk size, running an Ubuntu-14.04.02-LTS guest operating system, located in East US2 region i.e. one of the largest Microsoft Azure regions. Note that at the time of our instance launch, we have used the default (old) management portal that was available to us. As explained later, Microsoft now offers a new, improved portal that places each VM in a separate virtual private network by default. Nevertheless, the old portal is still available to users that choose to use it.

After instances launch, we record the network parameters such as the internal and public IP addresses, SSH port numbers and MAC addresses of all instances. Note that we do not record the hypervisor IP address since it is filtered by the virtual switch in Microsoft Azure. In order to identify the network parameters that might infer co-location, we first verify co-location with the memory bus locking mechanism. Since Microsoft Azure allows instances from the same account to be co-located, we obtain both inter-account and intra-account co-locations.

A. Memory Bus Lock Verification Method

We use the memory bus locking method described in Section II as a verification for our co-location assumptions based on network parameters. We use the memory bus as a covert channel to send messages between two instances. In particular, if an instances wants to send a 1 it will lock the memory bus to increase the memory latency in the co-located instance, whereas if it needs to send a 0 it will not trigger the memory lock. If the one instance receives the message sent by another instance correctly, then it is inferred that both instances use the same physical system hence co-located. After obtaining the base truth regarding co-locations with this method, we start investigating various network parameters in co-located instances.

Note that after a method to detect co-location through network is developed, the bus locking mechanism will not be needed for co-location detection. However, it will be used for QoS degradation attack.

B. Network Information Examination

In the following, we examine the data related to the network that we have collected during instance launches. We use this data to test our hypothesis that network information indeed reveals or at least hints at co-location.

1) Public IP Address: Instances created within the same Virtual Private Cloud (VPC) are assigned the same public IP address by Microsoft Azure, with a different SSH port to log in to individual machines. Since there can be up to 20 instances in a VPC, inspecting the public IP address most likely would not give any precise information about co-location. It is highly unlikely that 20 instances co-locate on a single physical machine. However, the public IP address can still give some partial information about co-location.

In order to test our hypothesis, we distribute 80 instances with different IPs across 4 accounts. Then applying the memory bus locking verification method, we check for co-location. Results in Table I verify that the instances with similar public IPs are not co-located and the public IP address is not a co-location indicator. Indeed, if public IP address proximity was related to co-location, `40.79.81.175`, `40.79.42.204` and `40.79.43.56` would show a higher probability of being co-located with each other, since they share first two octets. However, we observe the opposite behavior, i.e. `40.79.42.204` is co-located with a completely different public IP address (`13.68.18.16`), while `40.79.81.175` is co-located with `40.84.59.3`. Note that each of the public IPs does not necessarily correspond to more than one VM in the same account (Microsoft only allows a limited number of IPs per account). Thus, Microsoft Azure seems to successfully randomize the public IP address and it is not a co-location detection metric. Also, within an account where all instances share the same public IP, we had co-located and not co-located instances. Since these instances share the same public IP, it does not affect our hypothesis.

2) SSH Port Number: Microsoft Azure assigns private SSH port numbers ranging between 49152 and 65535 to instances within the same VPC. We investigate whether the assigned port number has any relation to the co-location. We consider port number distances of 10, 100, 200, 500, 1000, 3000 and check if any low distance hints co-location. Figure 1 shows the distribution of the assigned port number distances that we observed among our co-located instances in comparison to a random distribution. It can be observed that while the measured distribution is a bit lower, it still is not a valid indicator for co-location. Furthermore, Table II shows the ratio of correctly guessed co-located pairs when the

TABLE I. PUBLIC IP ADDRESS AND CO-LOCATION RELATION

PubIP	40.84.59.3	13.68.29.129	40.84.50.99	13.68.18.16	40.79.43.56	40.79.81.175	40.79.42.204	13.68.20.10
40.84.59.3	*	X	X	✓	X	X	✓	X
13.68.29.129	X	*	✓	X	X	X	X	X
40.84.50.99	X	✓	*	X	X	X	X	✓
13.68.18.16	✓	X	X	*	X	X	✓	X
40.79.43.56	X	X	X	X	*	X	X	X
40.79.81.175	X	X	X	X	X	*	X	X
40.79.42.204	✓	X	X	✓	X	X	*	X
13.68.20.10	X	X	✓	X	X	X	X	*

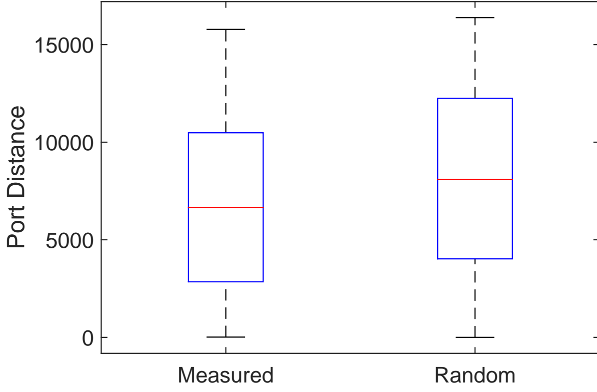

Fig. 1. Port distance distribution across co-located instances: measured vs random.

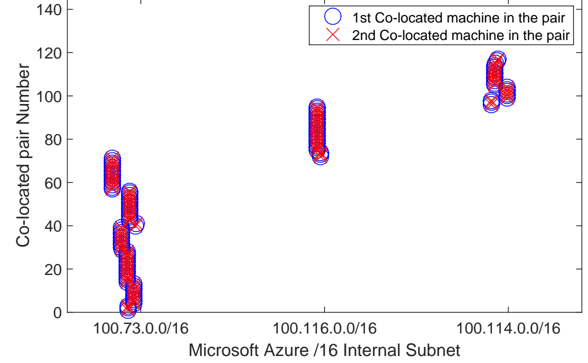

Fig. 2. Internal /16 subnet IP address proximity test. The x axis shows the different /16 addresses where our instances are placed, the y axis indicates the co-located pair number, and the markers represent each member of the pair.

instances are grouped according to their SSH port number. For simplicity reasons, we do not include the ratio of incorrectly guessed co-locations. It is clear that the SSH port assignment is random and even with the maximum distance considered (3000), 80% of the co-located instances remain undetected.

3) Internal IP Address: We now analyze the relationship between co-location and the auto-assigned internal IP addresses. First we examine the correspondence among addresses with the same first 2 octets, i.e. within the same /16 subnet. Note that if this test shows that the instances with different first 2 IP address octets can be co-located within the same machines, it would mean that the internal IP address proximity of instances has no effect on co-location.

In our experiments, we observed 3 different /16 subnets for 80 instances in 4 accounts. Figure 2 shows the distribution of our co-located machines in terms of the /16 internal IP addresses. Respectively the x and y axes represent the three different subnets and the co-located pair numbers. Each member of a pair is represented with an x and an o.

In total we confirmed 120 pairs to be co-located using the memory bus lock out of the 3200 potential co-located pairs from our 80 instances. Furthermore, we observe that *none* of the members of a co-located pair belong to different subnets. Note that, if the instances were assigned internal IP addresses at random, we would have a higher probability of observing co-location across different subnets than observing co-location within the same subnet. Thus, we can infer from this first observation that the internal IP addresses at least have partial relation with the co-located instances. However, if the co-located instances are placed within the same /16 subnet, this still yields a maximum of 2^{16} possible targets. In order to reduce this number, we proceed to analyze the subnets that have identical first 3 octets i.e. same /24 subnet of the Internal IP address.

Figure 3 shows the distribution of the 52 (out of 120) co-located instance private IPs in the `100.73.0.0/16` address range. As before, the x and y axes represent the different subnets and the co-location pair numbers respectively. For each y-value, members of the co-located pair are represented by an x and an o. We obtain similar results with the previous test. Again, our co-located instances share the same subnet of private IP addresses. Moreover, none of our 118 co-located pairs have a different /24 subnet, further supporting our hypothesis that the private IP addresses are strongly correlated to co-location. With this information, we can greatly reduce the number of candidates when we are looking for co-located instances with a specific target. Note that if the private IP addresses were assigned at random, we would have observed -with high ratio- co-located pairs with different subnets within the given range.

page

4) MAC Address: After observing that co-located instances are in the same /24 subnet, we proceed to check whether the MAC addresses are an indication of co-location as well. We first analyze the diversity of the MAC assignment in terms of the network device manufacturer code that is the first 6 hex-digits of the MAC. We utilize the popular *nmap* tool to discover the MAC addresses of the neighboring instances in the same subnet. Note that this step can also be performed with other tools like *arpscan*. We choose 3 different subnets at random and calculate the distribution of first 3 octets i.e. the Organizationally Unique Identifier.

Figure 4 shows the number of instances assigned to each identifier in 3 different subnets. We observe 3 identifiers

TABLE II. SUCCESS RATE OF CO-LOCATION GUESSES BASED ON PORT DISTANCE

Port Distance	10	100	200	500	1000	3000
Co-location Guess Success	0.8%	1.6%	1.6%	3.3%	7.6%	10.49%

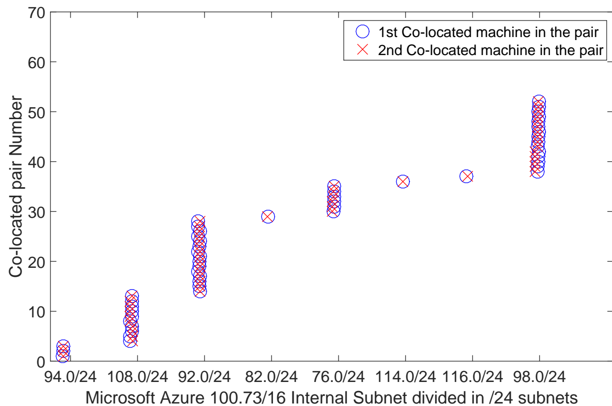

Fig. 3. Internal /24 Subnet IP address proximity test for the `173.70/16` internal IP address. The x axis shows the different /24 addresses where our instances are placed, the y axis indicates the co-located pair number, and the markers represent each member of the pair.

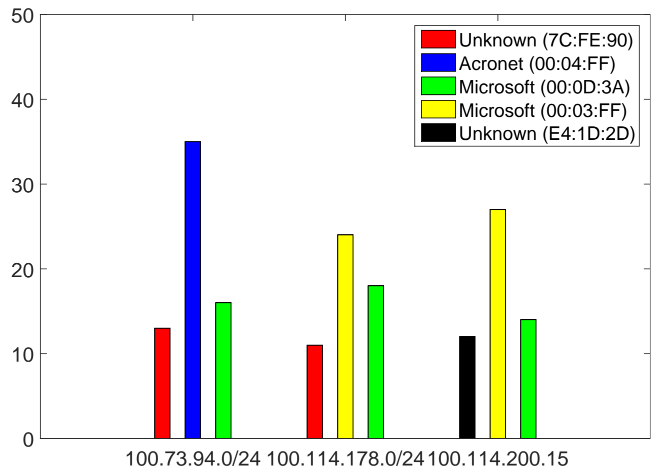

Fig. 4. Distribution of MAC identifiers in 3 different Microsoft Azure subnets.

TABLE III. FALSE NEGATIVE RATIO WHEN VARIOUS NUMBER OF IDENTICAL MAC DIGITS ARE USED TO GUESS CO-LOCATION.

Identical MAC Address Digits	Co-location Guess False Negative Rate
6	0%
7	0%
8	15.25%
9	70.34%
10	97.46%
11	100%

assigned to Microsoft, Acronet and some other organization in each subnet. The fact that there are different MAC identifiers within the same LAN leads us to believe that different organizational identifiers belong to different types of network devices and we can further reduce the co-location target radius. Note that, if the MAC address assignment was performed at random, we would either observe many MAC identifiers (not a normal distribution) or a unique one (and MAC addresses assigned at random within that identifier).

Similarly to the approach followed in the previous experiment, we try to identify our co-located instances by their MAC proximity. However, we only consider instances that are within the same subnet, i.e. we do not evaluate the MAC proximity across instances in different subnets since we already know that they are not co-located. We evaluate this proximity in terms of number of identical MAC address hex-digits. The experiment is carried out as follows: we fix the first n hex-digits of the MAC address, and assume that all the instances within that same subnet with identical first n MAC digits are co-located. For each n, we calculate the number of truly co-located instances that were incorrectly guessed.

Table III shows the outcome of the experiment. Fixing the first 6 digits yields a very high success rate and after 6, the more digits considered the less accuracy we obtain in the results. We observe that the 100% success rate is achieved by fixing either the first 6 or 7 digits, i.e. the instances that belong to a co-located pair have identical 6 or 7 digits and are placed in the same subnet. Having said that, all our instances that had identical first 6 digits also coincide in the 7th digit. Therefore we do not have enough entropy in our MAC pool to determine whether the 7th digit would support or refute the hypothesis. Also note that our experiments did not have false positives, supporting the hypothesis that if two instances have identical first 6 MAC hex-digits, then they are co-located.

In conclusion, we observed that if and only if virtual machines are located in the same /24 subnet and have identical MAC vendor are co-located. Our results further prove the opposite direction of the statement, i.e., we did not observe co-located pairs when any of those parameters are different. Thus, one can directly identify the co-located VMs by excluding those that do not meet the subnet and MAC vendor requirement. For the rest of the paper, this co-location verification method will be referred as the Network Co-location Detection (*NetCold*) approach.

IV. IMPLICATIONS AND ATTACK SCENARIOS

We demonstrated in the previous sections that we can determine whether two machines are co-located or not using basic network parameters; the internal IP address and the MAC address. The MAC address is specially interesting, since we obtain a unique identifier inside the LAN of our neighbors. This opens new features and attack scenarios than can be carried out inside the Microsoft Azure cloud.

A. LAN Scanning and Target Identification

With the new *NetCold* mechanism we can obtain certain unique parameters of the Virtual Machines that are co-located with our instances. In particular, we can perform a subnet scanning to detect which of our neighbors within the subnet

share the same first 6 MAC digits with our instances. If the MAC addresses are identical in the first 6 digits, we know that two instances are co-located. In short, the attacker has to obtain the following information about the co-located instances:

- **MAC address:** The attacker can obtain the unique MAC address assigned to the co-located target instances (identical MAC vendor ID) inside the subnet.
- **Internal IP address:** An attacker can also obtain the unique Internal IP addresses that share MAC vendor ID with his instance. Note that the internal IP address does not necessarily give any information about the public IP address.
- **Open Ports:** The attacker can also obtain the open ports on any internal IP address inside his subnet. Note that, a particular port filtered for the internal IP address might still be open in the public IP.

Thus, we pick one of our instances in each of the /24 subnets where they are allocated and perform the local subnet scanning together with the *NetCold* mechanism to find the number of our co-located neighbors. For all of our neighbors, we recover the internal IP, the MAC address and whether the port 80 or 443 are opened. We will use the port information later on.

Along with the parameters obtained, we observe a unique identification mechanism in Microsoft Azure. Indeed, given a victim v with a particular MAC address m in a subnet s and an attacker instance a, when a uses $nping$ to establish TCP connections with either the public or the internal IP address to whom that MAC address belongs to, we observe the following:

- If a, regardless of whether he is in the same subnet s or not, does not specify the correct destination IP address (either internal or public) or the correct destination MAC address m, the victim v never responds and the connection is not established.
- If a does not belong to the subnet s, even if he forges the TCP packet with the correct destination IP address (either internal or public) and the correct destination MAC address m, the TCP connection will never be established. The external router drops the package when the destination MAC field is determined, since it only implements layer 3 communications.
- If a is in the same subnet s and he forges the package with the correct destination IP address (internal or public) and the correct destination MAC address m, the connection is established and a response is given back to a.

Thus, forging a TCP package in Microsoft Azure determining the destination MAC address and either the internal or the public IP addresses allows us to determine whether that particular target is in our subnet or not. In other words, data link communications are not filtered by the hypervisor switch if they are performed from within the same subnet. The parameters obtained with our *NetCold* co-location detection mechanism together with the identification mechanism open two attack vectors that we will describe in the following sections.

B. Achieving Faster and Cheaper Targeted Co-location

Once we know who our neighbors are and what their MAC addresses are using *NetCold*, we can use our TCP connection approach to verify whether any of our co-located neighbors is the target that we are looking for. We only try those neighbor MAC addresses that share the same 6 MAC digits with the attacker instance, since we observed that is a key factor to be co-located. With this approach, if and only if the target is co-located it will send a response back to the TCP connection request.

In order to verify our approach, we set up public IP addresses to 5 of our instances at random, and open the port 80 on them. We will refer to these instances as the blue team. The goal is, for the rest of the instances (we will call them the red team) to find out where the members of the blue team reside and whether they are co-located with them. For that purpose, we set up a script that will first, for each instance, find the neighbors using *NetCold*, and then, establish a TCP connection with the targeted public IP addresses (assumed to be known) including each of the co-located destination MAC addresses and wait for a response. Clearly if any of the targets are co-located with any of our instances, those will receive a response from the blue team member they are co-located with, whereas if they are not, no response will be received. There is no collision problem if we run our targeted co-location scripts in one of the red team members, since our *NetCold* script excludes the IP address of the instance where the script is running from. We utilize the popular `nping` tool to establish the TCP connection, which allows specifying the destination MAC address as part of the IP address packet. Each member of the red team (in parallel) establishes 5 TCP connections (to the members of the blue team) with a 5 ms delay between them, and then waits 1 second to trigger a response. Thus, our targeted co-location approach only needs 1 second for each neighbor MAC verification.

Figure 5 shows the number of neighbors found in each of the subnets that have coincident MAC address with the instance together with the number of instances that found each of the public IP address assigned hosts. The number of co-located neighbors are represented in red, whereas each of the members of the blue team is represented with a different color. As we can see, the number of neighbors (and thus, the number of MAC addresses we have to try) varies from 6 to 32, averaging around 20. This limits the maximum number of TCP connections we need to establish to 32. It can also be observed that all the members of the blue team were found to be co-located with at least one member of the red team. Moreover, one of the blue team members is co-located with two red team members. As for the rest, only one instance was co-located with each of them and that is why they get a single count.

In short, the experimental observations prove our hypotheses and show the viability of our network based targeted co-location detection method. Due to its simplicity, our method has several advantages over previously proposed methods:

- The new targeted co-location method succeeds in the range of seconds, i.e. it is the fastest targeted co-location method proposed.

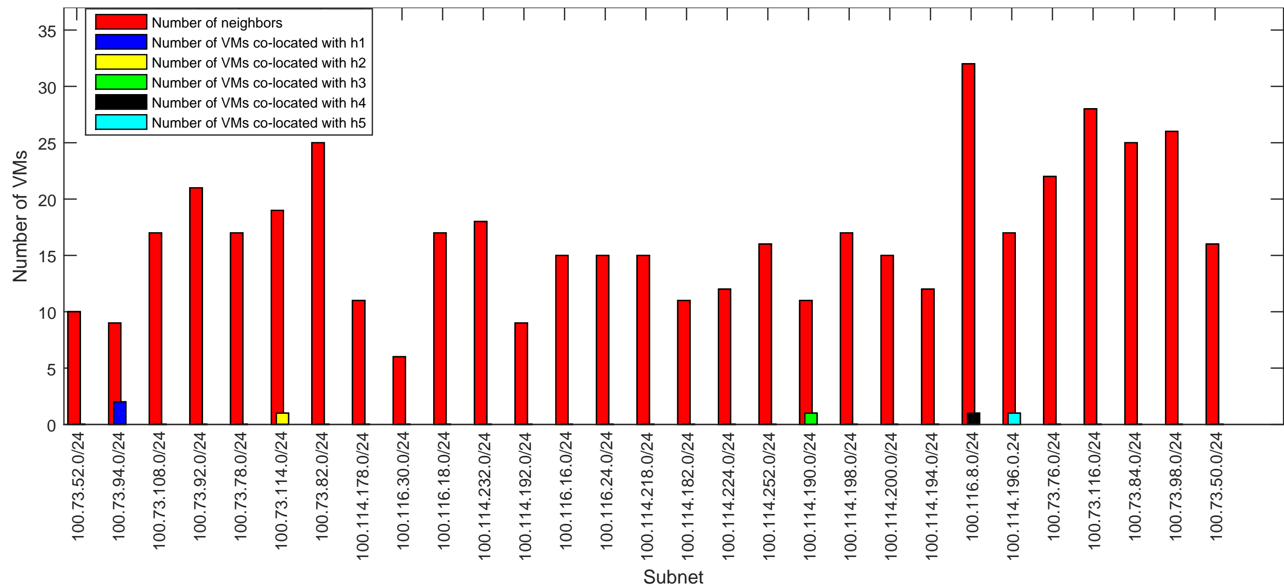

Fig. 5. Number of neighbors and number of targeted hosts found in each subnet

- The method does not have to cope with network timing noise or micro-architectural sources of noise and gives a clear accurate response.
- The method can easily be adjusted to introduce multiple targets without substantially increasing the success time. In fact, one can establish several TCP connections almost at the same time with the desired n targets.
- The proposed mechanism is hardly detectable by CSPs, since it only uses a single TCP request/second.

C. Achieving Co-location with a Priori Unknown Targets

Our co-location and identification methodology opens up another interesting scenario. In fact, a malicious attacker might not have an a priori list of targets, but he might want to build it a posteriori. For instance, consider the scenario in which a malicious attacker initiates various instances, making sure that they are not co-located with each other with the *NetCold* co-location detection method. The malicious attacker does not have any target in mind before opening his instances. Instead, the attacker might want to find out who is he co-located with after he has opened a sufficient number of instances. Then, once he knows whom he is co-located with, he can exploit his co-location if he finds his targets are of interest to him. More interestingly, the attacker can **rent** his co-located instances to the victim's competitor that *can* be highly interested in performing malicious activities such as DoS attacks or side-channel attacks.

Indeed, our co-location method together with our identification mechanism allows an attacker to find the identity of the instances he is co-located with a posteriori. The only thing the attacker needs to know a priori is what kind of service he wants to be co-located with. For simplicity reasons lets assume that the attacker wants to be co-located with web servers. There are two ways in which the attacker can discover his neighbors:

The HTTP port in the internal IP address is open: If the HTTP in the internal IP address is opened, the attacker can identify co-located services within seconds. Indeed, he can follow a simple two step approach:

- The attacker first uses the *NetCold* co-location detection method to find out what Internal IP addresses from within the same subnet are co-located with him.
- The attacker can, for each of the co-located internal IP addresses, request service to the HTTP port to see if he gets a response. If the attacker is co-located with a web server he directly gets the HTML file associated with that web server.

The HTTP port in the internal IP address is closed: In the case where the service cannot be retrieved through the internal IP, an attacker has to take additional steps to find valuable targets:

- First the attacker scans the entire public IP address range assigned to Microsoft Azure for IP addresses with the port 80 (or 443) opened, as it would be the case for web servers. The attacker stores the public IP addresses with the 80th port opened in list. The public IP address range for Microsoft Azure is available in [2].
- Once the list has been constructed, the attacker runs *NetCold* to detect the number of possible co-located targets and their respective MAC addresses.
- For each MAC address, the attacker runs the unique identification method described above, i.e. he establishes a TCP connection with each of the IP addresses in the list specifying the destination MAC address.
- Whenever the attacker is co-located with a public HTTP server, the server will respond to the TCP request. If the attacker is not co-located with any web server, then the attacker will not receive any response.

We prove the viability of both methods by opening 4 new instances that host a web server. Our goal is, without having them as a target beforehand, to be able to determine which of our original instances are co-located with these web servers (if any). The experiments are carried out using both of the above mentioned mechanisms, and will additionally obtain any web server (not only the ones that belong to us) that is co-located with our instances. The internal network scanning mechanism can test all the possible targets in a matter of seconds. As for the external network approach, we use the `masscan` to find all the public IP addresses within the Microsoft Azure range with their 80th port opened. This scan resulted in around 7000 IP addresses. We again used `nping` to establish the TCP connections. In order to improve the speed of the approach, we established several TCP connections with a very short delay without waiting for the response. Alongside, we observed the incoming packets with the popular tool `tcpdump`. Utilizing this approach we were able to verify each MAC address against all the public IP addresses in 3 minutes. Thus, the latency to check all the MAC addresses within a subnet was at most 90 minutes.

Figure 6 shows the number of services found for each of our /24 subnets. The x axis represent the subnet IP address, while the y axis represents the number of services that were found to be co-located. Servers found through the internal network are shown in red while servers found in the external network are shown in blue. In total, we found out the identity of 14 servers with the 80th port open in the internal network and 29 through the external network including 3 of our web servers. We do not consider broken services or sample IIS services. Among discovered servers, there were web pages, PaaS services, Apache web servers as well as some we were not authorized to have access to. It can be observed that the number of retrieved web servers is higher in the case of the public IP address scan than the internal network scan. However, our results still show that around 50% of the web servers do not properly filter their internal network ports since they are retrievable through the internal network.

Table IV represents the distribution of discovered co-located servers based on their service. During our experiments, we were co-located with 3 out of 4 of our test web pages. The undiscovered web page was residing in an instance that we had no co-location with. As for the rest of the co-located web pages, the majority belonged to companies hosting their web pages on Azure. In addition to these web pages that we could access to, we also discovered 6 web pages which we could not access to due to lack of authorization. Note that we could still discover identities of server owners by simply looking at the issued certificate. However, due to Server Name Indication (SNI) it is likely that we did not detect all of the web pages that may be hosted on a single instance. We also discovered web pages of financial institutions, stores and software developers. Lastly, we discovered a PaaS cloud service running on Azure infrastructure.

TABLE IV. DISTRIBUTION OF THE FOUND ACTIVE CO-LOCATED SERVICES.

Own web pages	Web pages	Apache web servers	PaaS services	Unauthorized access
3	17	2	1	6

The co-location identification mechanism proposed here presents the same advantages as the targeted co-location method. First, it can detect whether any existing web server is co-located in the range of seconds through the internal network and in at most 90 minutes in the case of the external network. These are the fastest (and probably the only viable) mechanism to determine co-location in the wild. Furthermore, the verification mechanism is again hardly detectable, since it only consists of one TCP connection with each host every 3 minutes.

V. QoS Degradation Results

There are several malevolent activities that malicious tenants can execute if they co-reside in the same server as their victim, e.g. side-channel attacks or QoS attacks. In this section we implement a QoS attack launched by one or many collaborative co-located instances to degrade the performance of the target victim. The QoS attack is based on provoking an unfair utilization of the memory bandwidth by continuously issuing memory locks from co-located instances. In order to perform realistic experiments, we utilize our set of co-located instances in Microsoft Azure. In particular we utilize the maximum number of co-located instances that we were able to obtain among our 4 accounts, i.e. 6 co-located instances in a single physical server. One of instances acted as a victim while the others were used to perform memory locks. Note that we performed these experiments for short periods of time and after midnight with the goal of minimizing the performance degradation of the co-located instances not controlled by us.

Table V shows the performance degradation due to the memory channel for variety of applications. For the test, we used the `phoronix-test-suite` that can perform numerous benchmarks. We used a set of 6 co-located instances, one running the tests while the other 5 were reserved to execute memory bus locking. From the 5 instances reserved to execute memory bus locking, we triggered the bus locking mechanism from varying number of instances simultaneously. By doing so, we were able to observe the affects of varying number of instances issuing bus locking in parallel on a single physical system. Note that since the hypervisor schedules instances on the CPU and presents an overhead of its own, this was crucial to show the advantage of having higher number of co-located instances with a target for maximum impact.

Due to the popularity of the Apache web server, the first application on our benchmark list was the **Apache web server**. The test sets up a web server and determines how many requests can the system handle in a second. Without any locks active, the web server was able to handle 2,760 requests per second. However, after running just 1 lock on a co-located instance, the web server performance was down to 2,039 requests per second and the degradation consistently increased when more locks were issued simultaneously. When 5 locks were active, the web server could only handle 479 requests, slowing down by a factor of 5.7. The second test we ran was the **Media Streaming**, aiming to determine how much of streaming would the attack result in. As expected, the memory heavy streaming test showed significant degradations with active memory bus locks. With no locks, the system could handle 4,444 MB/s of media streaming while with 5 locks this was down to 1,198 MB/s. The **Blogbench** test designed to

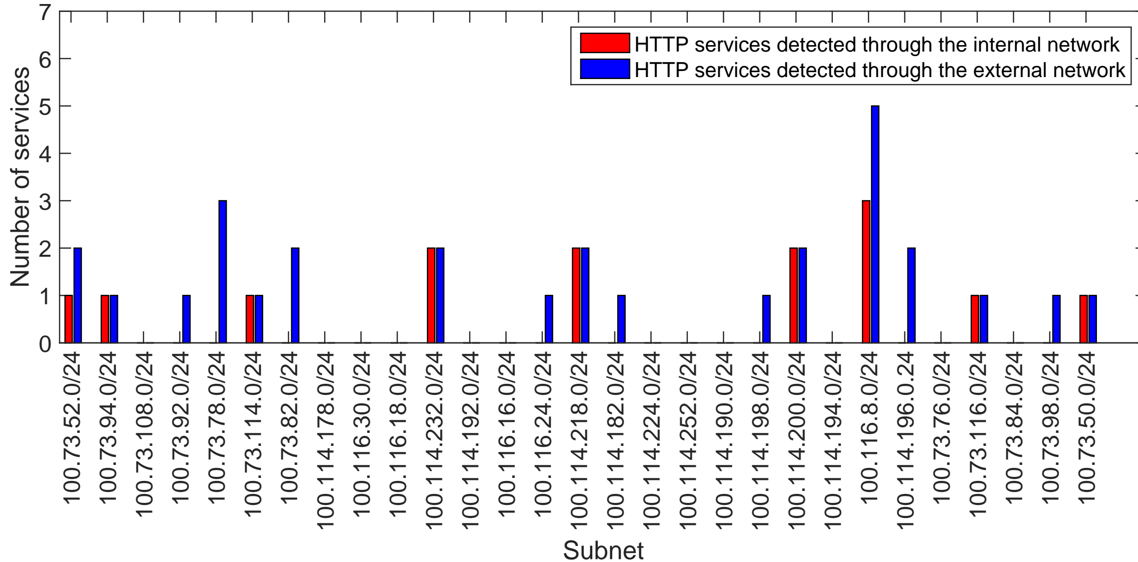

Fig. 6. Number of *active* web services co-located with our instances. Broken or sample web services are not considered.

simulate a busy file server with multiple read/write support. While unlocked, the system could handle 114727 reads and writes (of 64 Kb) per second. When even only 1 lock was issued, the system performance was down to 78566 reads and writes. And when all 5 locks were used, only 38,919 operations could be performed in a second, again showing significant performance degradation. Our next test, **Ebizzy** is designed to generate a highly threaded web application workload resembling a real web server. Without any locks the system was able to handle 3,384 records per second. And when all 5 locks were employed this was down to mere 720 records per second. Our final test, **Redis** is a data structure store being used as a database in a server-client model. With no locks present, 210,095 requests were handled per second and with 5 active locks, this was down to 48,792, again showing significant reduction in performance.

In conclusion, we show that variety of commonly used applications are affected by the memory bus locking. In addition to that, the performance degradation is directly related to the number of active locks running on the system. Therefore, in a malicious co-location scenario, more co-located instances the attacker has with the victim translates into stronger attack.

VI. Countermeasures

This work exploits logical network channels to derive co-location with high value targets and then exploits the memory bus to slow down the victim's performance. In order to prevent these two exploits, CSPs should:

- Randomize the MAC addresses assigned to the co-located instances in the same subnet. This at least would increase the entropy and the attacker will not know exactly who is he co-located with.
- The subnet placement was one of the parameters used in this paper to derive co-location. CSPs should randomize the internal IP addresses and increase the IP range to avoid direct malicious co-location identification.
- Placing each VM instance in a separate virtual private network by default would deprive the attacker of network scanning and mitigate this attack.
- Periodic live migration of instances would reduce the time an attacker is co-located with a specific target, forcing the attacker to achieve co-location over and over again.
- The memory bus locking was utilized to degrade the performance of the co-located victims. We believe cloud providers should implement a memory bus checking mechanism to monitor the locking activity. We believe that such an abnormal behavior continuously being executed should be easily detectable.

VII. Related Work

Cloud co-location was not considered until 2009, when Ristenpart et al. [18] showed that co-location with a target on a commercial cloud like Amazon EC2 is within attackers' reach. In order to achieve co-location they utilized several facts like ping timing delay between instances, traceroute to identify the first hop in the network or IP address proximity. Note that, the traceroute approach is now filtered by most of the firewalls (as in Azure and EC2) and the ping delay is not a reliable metric anymore. In comparison, their IP address proximity approach was not a definitive metric for co-location, does not reveal the identity and was fixed in EC2, while we combine it with the MAC address knowledge to make it a definitive co-location identifier and a target identifier.

Researchers have also utilized hardware covert channels to derive co-location in virtualized environments. In 2011 Zhang et al. [26] proposed a defensive co-location detection based on the L2 cache aiming at identifying when a particular user is in exclusive use of a physical machine. The L2 co-location mechanism is not applicable to detect co-location

TABLE V. PERFORMANCE DEGRADATION DUE TO MEMORY BUS LOCKING

Active Locks	Apache Requests/s	Media Streaming MB/s	BlogBench Read-Write/s	Ebizzy Records/s	Redis Requests/s	Average Degradation
0	2,760 - 100%	4,444 - 100%	114,727 - 100%	3,384 - 100%	210,095 - 100%	0%
1	2,039 - 73%	4,093 - 92%	78,566 - 68%	2,332 - 68%	146,583 - 69%	26%
2	1,300 - 47%	3,268 - 73%	73,991 - 64%	1,486 - 43%	96,724 - 46%	46%
3	800 - 28%	2,333 - 52%	57,262 - 49%	1,115 - 32%	71,734 - 34%	61%
4	675 - 24%	1,833 - 41%	42,186 - 36%	849 - 25%	61,194 - 29%	69%
5	479 - 17%	1,198 - 26%	38,919 - 33%	720 - 21%	48,792 - 23%	76%

across core in modern processors, since it is a core private resource. Furthermore, the authors propose a pure defensive approach, i.e. they do not perform any target identification step. In 2015, two new co-location methods where proposed to detect both collaborative and targeted co-location using a memory bus locking mechanism in Amazon EC2, Microsoft Azure and Google Cloud [21], [24]. Later in 2015, Inci et al. [10] presented a targeted co-location method based on Last Level Cache (LLC) contention. All these hardware resource based co-location methods are costly, noisy, detectable if the covert channel is monitored and they need access to the targets service. In contrast our approach is much faster (and thus cheaper), does not suffer from noise, stealthier, easily scalable and only establishes a single TCP connection with the target.

More research has been performed on finding covert channels without co-location purposes in virtualized environments. Most of the works focus on shared resources within the same physical server, such as the cache [23] or the memory bus [22]. Others measured the number of hops between co-located instances [8]. One of the main security issues that malicious co-location implies is the execution of side-channel attacks. Particularly powerful is the LLC, a resource that people have used to obtain cryptographic keys, keyboard strokes, TLS messages or e-commerce application private information across co-located instances [12], [14], [28], [15], [25], [11], [13], [4], [7], [16], [17]. Recently, Allan et al. [3] further showed that these attacks can be amplified through performance degradation. Lower level caches have also been utilized to obtain private information, but these are only applicable if the instances are co-residing in the same core [27].

As for DoS attacks, Shea et al. [19] studied a number of QoS attacks that could damage the performance of virtualized environments. Shortly later, Darwish et al. [5] studied a number of DoS attacks that can be performed in IaaS clouds (IP address spoofing, SYN flooding, etc) and later proposed defensive mechanisms to prevent them.

VIII. CONCLUSION

In conclusion, this paper presents a fast, cheap and stealthy co-location detection mechanism that works in both collaborative and non-collaborative scenarios. In fact, we use the more costly memory bus locking in the collaborative scenario to develop *NetCold*, a pure logical channel based co-location method that is cheap, fast and works great in a non-collaborative scenario. *NetCold* utilizes logical channels that were commonly believed to be closed by CSPs. It also shows how the co-located instances can be identified by simple TCP packet forgery. Once the targets have been identified, the Quality of Service can be degraded by utilizing a memory locking mechanism.

This co-location detection technique and the QoS attack can be used by a malicious party to offer QoS degradation as a service to competitors of identified targets. In our experiments, we have found instances with live, high value targets that are co-located with our instances. We believe cloud providers should perform a better randomization of their network layout to avoid the co-location technique exploited in this work. They should further check the memory bus so that the QoS attacks exploited in this work can be detected by the hypervisor.

IX. ETHICAL CONCERNS

Our experiments in public IaaS clouds were designed to conform with Azure's acceptable use policy, the law, and proper ethic. In this work, we took all necessary precautions to make sure that we did not interfere with Azure's services in any way. The IP address and MAC collections were lightweight network operations that did not stress the underlying network infrastructure. Also, we have used the LLC cache noise as an indicator of the physical system load and used this knowledge to determine off peak hours. The QoS degradation experiments were run during these off-peak hours (mostly after midnight) in order to minimize interference with other customers. In addition to that, the bus locking was employed for very short time intervals during the data collection.

X. INTERACTIONS WITH MICROSOFT

We have informed the Microsoft Azure about our findings well in advance to this publication. The Microsoft Azure team pointed out that with the new Azure management portal, VM instances are put in virtual private networks by default. This mitigates some of the network scans that can be used for reconnaissance and provides a stronger network isolation. Due to backwards compatibility concerns, the old portal is still available to customers that choose to use it.

In all experiments controlled by us, NetCold had a 100% success rate. To demonstrate the feasibility, we performed a single uncontrolled NetCold experiment with a single machine. Microsoft informed us thirteen days later that this experiment has failed, thus giving a first and singular false positive. In addition, Microsoft has informed us that they performed NetCold on a VM that was not co-located with any other VMs. A local subnet scan and identified several other VMs sharing the same subnet and the same leading 6 digits of MAC addressing, of which (obviously) none was co-located. We have no knowledge whether they have taken or are planning to take any steps to prevent the behavior reported in this work. We recognize that they are in full control of their network topology and can choose to close the logic channel used in NetCold at any time. The covert memory bus locking channel which we used to control the experiments is, however, much harder to close.

XI. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation, under grants CNS-1318919 and CNS-1314770.

REFERENCES

[1] Getting started with Microsoft Azure security. https://azure.microsoft.com/en-us/documentation/articles/azure-security-getting-started/.

[2] Microsoft Azure Public IP ranges. http://www.microsoft.com/en-us/download/details.aspx?id=41653.

[3] ALLAN, T., BRUMLEY, B. B., FALKNER, K., VAN DE POL, J., AND YAROM, Y. Amplifying side channels through performance degradation. Cryptology ePrint Archive, Report 2015/1141, 2015. http://eprint.iacr.org/.

[4] BENGER, N., VAN DE POL, J., SMART, N. P., AND YAROM, Y. "Ooh Aah... Just a Little Bit": A Small Amount of Side Channel Can Go a Long Way. In *CHES* (2014), pp. 75–92.

[5] DARWISH, M., OUDA, A., AND CAPRETZ, L. F. Cloud-based ddos attacks and defenses. In *Information Society (i-Society), 2013 International Conference*.

[6] DUHIGG, C. Stock traders find speed pays, in milliseconds. New York Times, July 2009.

[7] GRUSS, D., SPREITZER, R., AND MANGARD, S. Cache template attacks: Automating attacks on inclusive last-level caches. In *USENIX Security* (2015), pp. 897–912.

[8] HERZBERG, A., SHULMAN, H., ULLRICH, J., AND WEIPPL, E. Cloudoscopy: Services discovery and topology mapping. CCSW '13.

[9] HOFF, T. Latency is everywhere and it costs you sales - how to crush it. High Scalibility, July 2009.

[10] INCI, M. S., GULMEZOGLU, B., EISENBARTH, T., AND SUNAR, B. Co-location Detection on the Cloud. In *COSADE* (2016), Springer.

[11] INCI, M. S., GULMEZOGLU, B., IRAZOQUI, G., EISENBARTH, T., AND SUNAR, B. Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud. Tech. rep. http://eprint.iacr.org/.

[12] IRAZOQUI, G., EISENBARTH, T., AND SUNAR, B. S$A: A shared cache attack that works across cores and defies VM sandboxing?and its application to AES. *IEEE S&P* (2015).

[13] IRAZOQUI, G., INCI, M. S., EISENBARTH, T., AND SUNAR, B. Wait a minute! A fast, Cross-VM attack on AES. In *RAID 2014*. pp. 299–319.

[14] IRAZOQUI, G., İNCI, M. S., EISENBARTH, T., AND SUNAR, B. Lucky 13 Strikes Back. ASIA CCS '15, pp. 85–96.

[15] LIU, FANGFEI AND YAROM, YUVAL AND GE, QIAN AND HEISER, GERNOT AND LEE, RUBY B. Last-level cache side-channel attacks are practical. In *IEEE S&P* (2015), pp. 605–622.

[16] MAURICE, C., NEUMANN, C., HEEN, O., AND FRANCILLON, A. *DIMVA 2015*. ch. C5: Cross-Cores Cache Covert Channel.

[17] OREN, Y., KEMERLIS, V. P., SETHUMADHAVAN, S., AND KEROMYTIS, A. D. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *CCS 2015*, pp. 1406–1418.

[18] RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *CCS '09*, pp. 199–212.

[19] SHEA, R., AND LIU, J. Understanding the impact of denial of service attacks on virtual machines. In *(IWQoS), 2012 IEEE International Workshop*.

[20] SYNERGY RESEARCH GROUP. AWS Remains Dominant Despite Microsoft and Google Growth Surges.

[21] VARADARAJAN, V., ZHANG, Y., RISTENPART, T., AND SWIFT, M. A placement vulnerability study in multi-tenant public clouds. In *24th USENIX Security Symposium (USENIX Security 15)(Washington, DC* (2015), pp. 913–928.

[22] WU, Z., XU, Z., AND WANG, H. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In *USENIX Security 12* (2012), pp. 159–173.

[23] XU, Y., BAILEY, M., JAHANIAN, F., JOSHI, K., HILTUNEN, M., AND SCHLICHTING, R. An exploration of l2 cache covert channels in virtualized environments. CCSW '11, pp. 29–40.

[24] XU, Z., WANG, H., AND WU, Z. A measurement study on co-residence threat inside the cloud. In *24th USENIX Security* (2015), pp. 929–944.

[25] YAROM, Y., AND FALKNER, K. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *(USENIX Security 14)*, pp. 719–732.

[26] ZHANG, Y., JUELS, A., OPREA, A., AND REITER, M. K. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*.

[27] ZHANG, Y., JUELS, A., REITER, M. K., AND RISTENPART, T. Cross-VM side channels and their use to extract private keys. In *CCS 2012* (2012), pp. 305–316.

[28] ZHANG, Y., JUELS, A., REITER, M. K., AND RISTENPART, T. Cross-tenant side-channel attacks in paas clouds. In *CCS* (2014), pp. 990–1003.