A Measurement Study on Co-residence Threat inside the Cloud

Zhang Xu
The College of William and Mary
zxu@cs.wm.edu

Haining Wang
University of Delaware
hnw@udel.edu

Zhenyu Wu
NEC Laboratories America
adamwu@nec-labs.com

Abstract

As the most basic cloud service model, Infrastructure as a Service (IaaS) has been widely used for serving the ever-growing computing demand due to the prevalence of the cloud. Using pools of hypervisors within the cloud, IaaS can support a large number of Virtual Machines (VMs) and scale services in a highly dynamic manner. However, it is well-known that the VMs in IaaS are vulnerable to co-residence threat, which can be easily exploited to launch different malicious attacks. In this measurement study, we investigate how IaaS evolves in VM placement, network management, and Virtual Private Cloud (VPC), as well as the impact upon co-residence. Specifically, through intensive measurement probing, we first profile the dynamic environment of cloud instances inside the cloud. Then using real experiments, we quantify the impacts of VM placement and network management upon co-residence. Moreover, we explore VPC, which is a defensive network-based service of Amazon EC2 for security enhancement, from the routing perspective. On one hand, our measurement shows that VPC is widely used and can indeed suppress co-residence threat. On the other hand, we demonstrate a new approach to achieving co-residence in VPC, indicating that co-residence threat still exists in the cloud.

1 Introduction

Entering the era of cloud computing, Infrastructure as a Service(IaaS) has become prevalent in providing Information Technology (IT) support. IT giants such as Amazon [1], Microsoft [4], and Google [2] have deployed large-scale IaaS services for public usage. Employing IaaS, individual IT service providers can achieve high reliability with low operation cost and no longer need to maintain their own computing infrastructures. However, IaaS groups multiple third-party services together into one physical pool, and sharing physical resources with other customers could lead to unexpected security breaches such as side-channel [25] and covert channel [19] attacks. It is well-known that IaaS is vulnerable to the co-residence threat, in which two cloud instances (i.e., VMs) from different organizations share the same physical machine. Co-residence with the victim is the prerequisite for mounting a side-channel or covert-channel attack.

The security issues induced by co-residence threat have been studied in previous research. However, most previous works focus on "what an attacker can do" [14, 19, 25], "what a victim user should do" [24], and "what a cloud vendor would do" [12, 15, 26]. In contrast, to the best of our knowledge, this measurement work initiates one of the first attempts to understand how cloud service vendors have potentially reacted to co-residence threat in the past few years and explore potential new vulnerabilities of co-residence inside the cloud. While Amazon Elastic Compute Cloud (EC2) is the pioneer of IaaS, it has the largest business scale among mainstream IaaS vendors [11, 18]. Therefore, we focus our study on Amazon EC2. More specifically, our measurement is mainly conducted in the largest data center hosting EC2 services: the northern Virginia data center, widely known as US-East region.

In our measurement study, we first perform a 15-day continuous measurement on the data center using ZMap [10] to investigate the data center's business scale and some basic management policies. With the basic knowledge of the cloud, we explore how EC2 has adjusted VM placement along with its impact on security. We further evaluate how much effort an attacker needs to expend to achieve co-residence in different circumstances. Comparing our evaluation results with those from 2008 [14], we demonstrate that the VM placement adjustment made by EC2 during the past few years has mitigated the co-residence threat.

As network management plays a critical role in cloud performance and security, we also investigate how the networking management in EC2 has been calibrated to

suppress co-residence threat. We conduct large scale trace-routing from multiple sources. Based on our measurements, we highlight how the current networking configuration of EC2 is different from what it was and demonstrate how such evolution impacts co-residence inside the cloud. In particular, we measure the change of routing configuration made by EC2 to increase the difficulty of cloud cartography. We also propose a new algorithm to identify whether a rack is connected with Top of Rack switch or End of Row switch. With this algorithm, we are able to derive the network topology of EC2, which is useful for achieving co-residence inside the cloud.

To provide tenants an isolated networking environment, EC2 has introduced the service of Virtual Private Cloud (VPC). While VPC can isolate the instances from the large networking pool of EC2, it does not physically isolate the instances. After profiling the VPC usage and the routing configurations in VPC, we propose a novel approach to speculating the physical location of an instance in VPC based on trace-routing information. Our experiments show that even if a cloud instance is hidden behind VPC, an adversary can still gain co-residence with the victim with some extra effort.

The remainder of the paper is organized as follows. Section 2 introduces background and related work on cloud measurement and security. Section 3 presents our measurement results on understanding the overview of Amazon EC2 and its basic management policies. Section 4 details our measurement on VM placement in EC2, including co-residence quantification. Section 5 quantifies the impact of EC2-improved network management upon co-residence. Section 6 describes VPC, the most effective defense against co-residence threat, and reveals the haunted co-residence threat in VPC. Section 7 proposes potential solutions to make the cloud environment more secure. Finally, Section 8 concludes our work.

2 Background and Related Work

To leverage physical resources efficiently and provide high flexibility, IaaS vendors place multiple VMs owned by different tenants on the same physical machine. Generally, a scenario where VMs from different tenants are located on the same physical machine is called co-residence. In this work, the definition of co-residence is further relaxed. We define two VMs located in the same physical rack as co-residence. Thus, two VMs located in the same physical machine is considered as machine-level co-residence, while two VMs located in the same rack is defined as rack-level co-residence.

2.1 Co-residence threat

The threat of co-residence in the cloud was first identified by Ristenpart et al. [14] in 2009. Their work demonstrates that an attacker can place a malicious VM co-resident with a target and then launch certain attacks such as side channel and covert channel attacks. Following Ristenpart's work, Xu et al. [20] studied the bit rate of cache-based covert channel in EC2. Wu et al. [19] constructed a new covert channel on a memory bus with a much higher bit rate, resulting in more serious threats in an IaaS cloud. Zhang et al. [25] proposed a new framework to launch side channel attacks as well as approaches to detect and mitigate co-residence threat in the cloud [24, 26]. Bates et al. [7] proposed a co-resident watermarking scheme to detect co-residence by leveraging active traffic analysis.

The reason we define different levels of co-residence is that some attacks do not require VMs to be located on the same physical machine, but rather in the same rack or in a higher level network topology. For instance, Xu et al. [23] proposed a new threat called power attack in the cloud, in which an attacker can rent many VMs under the same rack in a data center and cause a power outage. There are also some side channel and covert channel attacks that only require the co-residence in the same sub-network [5].

In parallel with our work, Varadarajan et al. [16] performed a systematical study on placement vulnerability in different clouds. While their work mainly stands at the attacker side to explore more effective launch strategies for achieving co-residence in three different clouds, our work performs an in-depth study to understand the evolution of cloud management and the impact on co-residence threat in Amazon EC2. The two complementary works both support the point that public clouds are still vulnerable to co-residence threat.

2.2 Measurement in the cloud

In contrast to the measurement on private clouds from an internal point of view[9], the measurement works on public data centers are mostly conducted from the perspective of cloud customers. Wang et al. [17] demonstrated that in a public cloud, the virtualization technique induces a negative impact on network performance of different instance types. The work of Xu et al. [21] measures network performance in Amazon EC2 and demonstrates a long tail distribution of the latency. Their work also analyzes the reason behind the long tails and proposes a new VM deployment solution to address this issue. Bermudez et al. [8] performed a large-scale measurement on Amazon AWS traffic. Their study shows that most web service traffic towards Amazon AWS goes to the data center in Virginia, U.S. Some recent studies [11, 18] measure how web services are deployed in public clouds. They found that although many top-ranked domains deploy their subdomains into the cloud, most subdomains are located in the same region or zone,

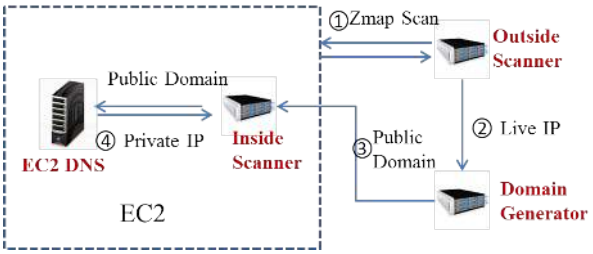

Figure 1: The system used to scan EC2.

resulting in a relatively poor fault tolerance.

In contrast to those measurement efforts, our study provides a measurement analysis from the perspective of security to reveal the management policies of a public cloud and their impact upon co-residence threat.

3 An Overview of EC2 Management

As the pioneer of IaaS, Amazon EC2 deploys its data centers all around the world, hosting the largest scale of IaaS business. In this section, we introduce some terminology in EC2 and provide an overview of the EC2 environment.

3.1 Instance type

An instance represents a virtual machine (VM) in the cloud, so we use the term "instance" and "VM" interchangeably throughout the rest of the paper. EC2 provides a list of instance types for clients to select while launching a new instance. The type of an instance indicates the configuration of the VM, determining the amount of resources the VM can use. The instance type is defined in the format *XX.XXX* such as *m1.small*. The first part of the instance type reveals the model of the physical server that will host this type of instance. The second part indicates the "size" of the VM, i.e., the amount of resources allocated to the instance. The detailed configuration of different instance types can be found at [3].

3.2 Regions and zones

Amazon EC2 has the concept of "region," which represents the physical area where the booted instance will be placed. Amazon has 9 locations around the world hosting EC2 services. Therefore, the instances in EC2 can be located in 9 regions: US east (northern Virginia), US west (Oregon), US west (northern California), South America (Sao Paulo), Asia Pacific southeast (Singapore), Asia Pacific southeast (Sydney), Asia Pacific northeast (Tokyo), EU west (Ireland), and EU central (Frankfurt). As pointed out in previous work [11], the majority of IaaS business is hosted in the US east region, e.g., in the data center located in northern Virginia. Most existing research on cloud measurement was conducted on this region [8, 13, 14]. Therefore, we also focus our study on the US east region. For the rest of the paper, we use the term "cloud" to mean the EC2 US east region and the term "data center" to mean the Amazon EC2 data center in northern Virginia, US.

In addition to regions, Amazon EC2 also allows clients to assign an instance to a certain "zone." A zone is a logical partition of the space within a region. Previous work shows that the instances in the same zone share common characters in private IP addresses, and likely instances within the same zone are physically close to each other [14, 19]. There are four availability zones in the US east region: us-east-1a, us-east-1b, us-east-1c, and us-east-1d.

3.3 Naming

The naming service is essential to cloud management. On one hand, the naming service can help customers to easily access their instances and simplify resource management. On the other hand, the naming service should help the cloud vendor to manage the cloud efficiently with high network performance.

In EC2, an instance is automatically assigned two domain names: one public and one private. The public domain name is constructed based on the public IP address of the instance, while the private domain name is constructed based on either the private IP address or the MAC address. Performing a DNS lookup outside EC2 returns the public IP of the instance, while performing a DNS lookup inside EC2 returns the private IP of the instance.

3.4 Scanning EC2 inside and outside

To better understand the environment and business scale of EC2, we performed a 15-day continuous measurement on the EC2 US east region.

Figure 1 illustrates our system to scan EC2. First we deployed a scanner outside EC2 to scan the cloud through a public IP address. Since EC2 publishes the IP range for its IaaS instances, our scanner uses ZMap [10] to scan the specified ranges of IP addresses. The ports we scanned include: ports 20 and 21 used for FTP, port 22 used for SSH, port 23 for telnet, ports 25 and 587 for SMTP, port 43 for WHOIS, port 53 for DNS, port 68 for DHCP, port 79 for Finger protocol, port 80 for HTTP, port 118 for SQL, port 443 for HTTPS, and port 3306 for MySQL. We also performed an ICMP echo scan. After scanning, our outside scanner obtained a list of live hosts in EC2 with the corresponding public IP addresses. In the next step, we performed automatic domain name generation. As mentioned above, the public domain name of an instance in EC2 can be derived using its public IP. This step produces a list of public domain names of live hosts. The generated public domain names were then sent to our inside scanner deployed inside EC2. Our inside scanner then performed DNS lookups for these domain names.

Due to the DNS lookup mechanism of EC2, the DNS server in EC2 answered the queries with the private IP addresses of the hosts. Reaching this point, our measurement system can detect live hosts in EC2 with their domain names, IP addresses, as well as the mapping between the public IP address and private IP address.

The scan interval is set to 20 minutes, which is a trade-off between cost and accuracy. Scanning the entire EC2 US east region per port takes about 40 seconds, and we have 14 ports to scan. This means that scanning all the ports will take around 10 minutes. Note that our measurement also includes DNS lookups for all the detected live hosts. Performing these DNS lookups takes around 20 minutes, which is approximately the time for two rounds of scanning.

Our scanning measurement provides us an overview of the large business scale of EC2, the diversity of services, and the dynamic running environment. This scanning measurement also gives us the knowledge base to understand co-residence threat. The detailed results and analysis of our scanning measurement can be found in the Appendix A and B.

4 The Impact of VM Placement upon Co-residence

The VM placement policy of the cloud determines how easy or hard it is for an attacker to achieve co-residence. In this section, we present our measurement on VM placement and quantification of achieving co-residence. By comparing our measurement results with previous work, we demonstrate how the VM placement policy has been evolving in EC2 and its impact on mitigating co-residence threats.

4.1 Basic understanding of VM placement

We first launched a sufficiently large number of instances with different types in EC2. Then, we had two tasks to fulfill: (1) collecting networking (i.e., location) information of launched instances and (2) quantifying co-residence threat, i.e., given the current VM placement policy of EC2, how much effort an attacker needs to make to achieve co-residence. Since the process of achieving co-residence requires the knowledge of instance location, we can complete the two tasks together. For every instance we launched while seeking co-residence, we recorded its private IP address and public IP address. We also performed an automatic traceroute from the instance to its "neighbors" that share the /24 prefix with it. This information can provide us the basic knowledge of where the instances are placed.

During our measurement, we recorded the detailed information of 2,200 instances of type t1.micro, 1,800 instances of type m1.small, 1,000 instances of type m1.medium, 1,000 instances of type m3.medium, 80 instances of m3.large, and 40 instances of m3.xlarge. We selected some random samples from the instances we recorded to study the internal IP distribution. We investigated how private IP addresses are associated by the instance type and availability zones, i.e., whether the VM placement has type and zone locality. Our results demonstrate that currently EC2 still exhibits certain type and zone locality, i.e., instances with the same type in the same zone are more likely to be placed close to one another. However, compared with corresponding results in 2008 [14], such locality has been significantly weakened. More details of locality comparison can be found in Appendix C.

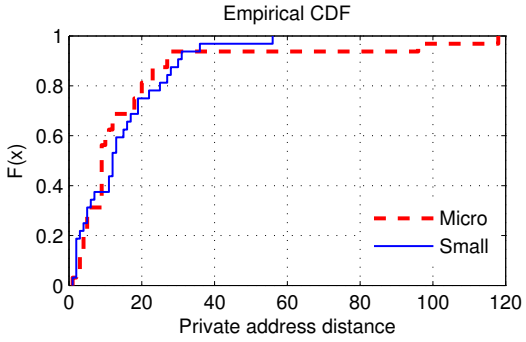

Figure 2: CDF of IP address distances between co-resident VMs.

After understanding the current VM placement in EC2, we further investigate co-residence threats in EC2.

4.2 Quantifying machine level co-residence

To understand how VM placement will affect co-residence, we assess the effort one needs to make to achieve machine level co-residence in two scenarios. The first scenario is to have a random pair of instances located on the same physical machine, and the second scenario is to have an instance co-reside with a targeted victim.

4.2.1 Random co-residence

To make our random co-residence quantification more comprehensive, we perform our measurement with different instance types and in different availability zones. Since zone us-east-1c is no longer hosting t1, m1, c1, and m3 instances, our measurement is performed in zone us-east-1a, us-east-1b, and us-east-1d. We achieve co-residence pairs with t1.micro, m1.small, m1.medium, and m3.medium. We did not achieve co-residence with large, xlarge or 2xlarge instances, because there are only 1 to 4 such large instances on one physical machine and it will be very difficult and costly to achieve co-residence with these types. Overall, we conduct 12 sets of experiments, with each set targeting a specific type of instances in a specific availability zone.

In each set of experiments, we perform rounds of co-residence probing until we find a co-residence pair.

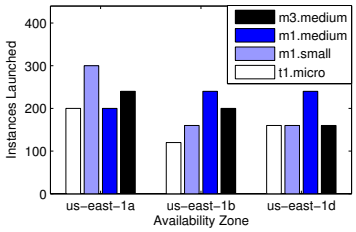

Figure 3: The service hour spent, i.e., the number of instances booted to achieve co-residence.

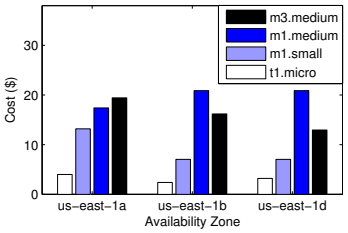

Figure 4: The financial cost (in US dollar) to achieve co-residence.

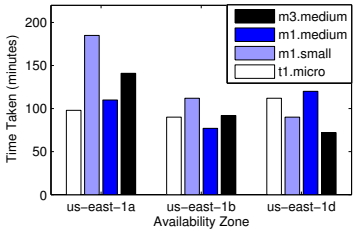

Figure 5: The time spent to achieve co-residence.

For the sake of robustness, EC2 has never placed instances from the same user on the same physical machine [14]. Therefore, we set up two accounts to launch instances simultaneously. Within one round, each account launches 20 instances, which will produce 400 pairs of co-residence candidates. Once a co-residence pair is verified, this set of experiments are terminated and the corresponding cost is recorded. If there is no co-residence pair found in this round, we move on to the next round by terminating all running instances and launching another 20 instances in each account, and then repeat the same procedure.

Given a pair of instances, verifying whether they are located on the same physical machine involves two steps: (1) pre-filtering unlikely pairs and (2) using a covert channel to justify co-residence.

For the first step, we need to screen out those pairs that are not likely to be co-resident to reduce probing space. Since the private IP address of an instance can indicate its physical location to some extent, and if the private IP addresses of two instances are not close enough, the two instances will have little chance to be co-resident. Based on this heuristic, we use the share of /24 prefix as the prerequisite of co-residence, i.e., if two instances do not share the /24 prefix, we consider them as not being co-resident and bypass the highly costly step 2. The rationale of setting the /24 prefix sharing as pre-filter is twofold:

1. First, the prerequisite of the /24 prefix sharing will not likely rule out any co-residence instance pairs. The number of instances that are hosted on the same physical machine is limited. Even for micro instances, there are no more than 32 instances running on a physical machine. For the instance type with larger size, there are even fewer instances running on a physical machine. In contrast, a /24 address space can contain 256 instances. Therefore, two co-resident instances are unlikely to be in different /24 subnets. Moreover, we obtained some co-residence pairs without any pre-filtering and recorded the private IP address distance between a pair of co-residence instances. Figure 2 illustrates the CDF of IP address distance between two co-residence instances. The distance is calculated as the difference between the two 32-bit integers of the two IP addresses. From the results we can figure out that most of these co-residence instances share the /27 prefix, which further confirms that the /24 prefix filtering will introduce very few, if any, false negatives.

2. Second, the prerequisite of sharing the /24 prefix can effectively narrow down the candidate space. Each time we use one account to launch 20 instances and use another account to launch another 20 instances, we will have 400 candidate pairs. During our measurement, we generated more than 40 rounds of such 400-pair batches. The average number of instance pairs that share the /24 prefix among 400 candidates is only 4. This means the /24 prefix sharing prerequisite can help us to screen out 99% of the candidates, which significantly accelerates the process of co-residence verification. During the 40 rounds of measurement, five co-residence pairs are observed.

The second step is to use a covert channel to verify whether two instances are actually located on the same physical machine. We use the technique introduced by Wu et al. [19] to construct a memory-bus-based covert channel between two instances. If the two instances can communicate with each other via the covert channel, then they are located on the same physical machine. This covert-channel-based verification can guarantee zero false positives.

The cost of achieving co-residence includes financial cost and time. According to the pay-as-you-go billing system, the financial cost is mainly determined by the service hours consumed during the co-residence probing. Every time an instance is launched, one billing hour is charged. Thus, the more probing instances an attacker needs to launch, the higher financial cost it will cause. In our experiments, we use only two accounts. In a real world attack, an attacker could use more accounts to launch the attack in parallel, which will result in less time required to achieve co-residence. However, under the same condition, regardless of attack process op-

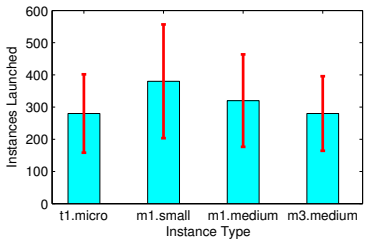

Figure 6: The service hour spent, i.e. the number of instances booted to achieve co-residence with a target.

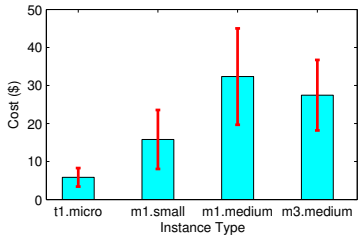

Figure 7: The financial cost (in US dollar) to achieve co-residence with a target.

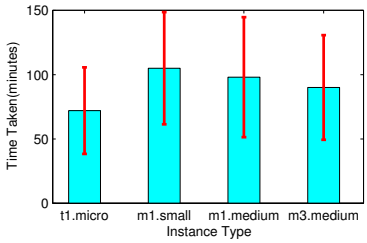

Figure 8: The time spent to achieve co-residence with a target.

timization, the time spent to achieve co-residence should have a positive correlation with the number of instances to launch, i.e., the more instances need to launch, the more time spent for detecting co-residence.

Figure 3 illustrates how many instances are required to achieve co-residence, while Figure 4 illustrates the actual financial cost. Figure 5 illustrates how much time it takes to achieve co-residence, i.e., the time cost. For each type of instance, the measurement repeats for five times and the mean value is shown in the figures. From the figures, it is evident that the cost for achieving co-residence of different types in different availability zones is quite different. Intuitively, as a larger instance has higher resource charge, it costs more money to achieve co-residence with those instances at a larger size. However, there is no such rule that the smaller size an instance is, the lower time cost we need to pay for co-residence.

4.2.2 Target co-residence

In the quantification of achieving co-residence with a particular target, we first randomly launched one instance with specific type from one account as the target. Then, from the other account, we also performed many rounds of co-residence probing until we found the instance that is co-resident with the target. The process of verifying co-residence remains the same. As demonstrated by the verification results of random co-residence above, different availability zones do not greatly impact the difficulty of achieving co-residence. Here we only show the results when our target instances are placed in zone us-east-1a.

Figures 6, 7 and 8 illustrate the number of instances to launch, the financial cost, and the time taken to achieve co-residence with a particular target, respectively. For each type of instance, the measurement is repeated for 15 times and the mean value is illustrated. The error bar with standard deviation is also shown in the figures. As is intuitive, achieving co-residence with a particular target requires launching more instances than achieving random co-residence. Getting a random co-residence pair requires launching 200 to 300 instances with two accounts (i.e., 100 to 150 instances per account), which can be done in 5 to 8 rounds. In contrast, achieving co-residence with a particular target requires launching 300 to 400 instances, which will take 15 to 20 rounds with each round launching 20 instances from one account. However, achieving co-residence with a particular target does not cost more time than achieving a random co-residence pair. The reason for this is simple: To get a random pair, we need to check 400 candidate pairs in each round, but to get a co-residence pair with a target, we only need to check 20 candidates in one round.

It is also possible that an attacker is unable to achieve co-residence with a certain target due to various reasons, e.g., the target physical machine reaches full capacity. During our study, we failed to achieve co-residence with two targets, one is m1.medium type and the other is m3.medium type. By failing to achieve co-residence we mean that after trying with more than 1,000 probing instances in two different days, we still cannot achieve co-residence with these two targets.

Overall, it is still very feasible to achieve co-residence in EC2 nowadays. However, an attacker needs to launch hundreds of instances to reach that goal, which may introduce considerable cost. In Section 4.4, we will compare our results to previous studies, demonstrating that achieving machine-level co-residence has become much more difficult than before, due to the change in cloud environments and VM placement policies.

4.3 Quantifying rack level co-residence

While covert channel and side channel attacks require an attacker to obtain an instance located exactly on the same physical machine with the victim, some malicious activities only need coarse-grained co-residence. Xu et al. [23] proposed a new attack called power attack. In their threat model, the attacker attempts to significantly increase power consumption of multiple machines connected by the same power facility simultaneously to trip the circuit breaker (CB). Since these machines located in the same rack are likely to be connected by the same CB, in a power attack the attack instances are not required to be placed on a same physical machine. Instead the attacker should place many instances within the same rack as the victim, i.e., achieving as much rack-level co-residence as possible. We performed measurement on how much effort is required to place a certain number of

Table 1: The number of co-residence pairs achieved by one round of probing in 2008 [14].

	Account A	Account B	Co-residence
Zone 1	1	20	1
	10	20	5
	20	20	7
Zone 2	1	20	0
	10	20	3
	20	20	8
Zone 3	1	20	1
	10	20	2
	20	20	8

instances under the same rack.

We first use one account to launch 20 instances, and then we check whether there are any instances in this batch that are located within the same rack. If there are no instances located in the same rack, we just randomly pick an instance and set its hosting rack as the target rack. Thanks to the Top of Rack(ToR) switch topology, verifying whether two instances are in the same rack is simple. Through a simple trace-routing, we can verify whether an instance has the same ToR switch with our target rack. This rack level co-residence can be further verified by performing trace-route from the candidate instance to the target instance. If the two instances are in the same rack, there should be only one hop in the trace, i.e., they are one hop away.

Figure 9 shows our measurement results. It is clear that an attacker can easily have multiple instances located within the same rack. The information of ToR switch helps the attacker quickly verify the rack-level co-residence. Since the malicious attack based on the rack-level co-residence is newly proposed [23], EC2 is unlikely to take any action to suppress rack-level co-residence.

4.4 Battle in VM placement

Table 1 lists the data from the original work on co-residence [14]. We can see that it was extremely easy to achieve co-residence in 2008. With two accounts each launching 20 instances, there were 7 or 8 co-residence pairs observed. In the 2012 work [19], the cost of achieving a co-residence instance pair is also briefly reported: A co-residence pair (micro) is achieved with 160 instances booted.

As we can see, nowadays it is much more difficult to achieve co-residence than in 2008 and 2012. EC2 could have adjusted its VM placement policies to suppress co-residence.

4.4.1 A larger pool

The business of EC2 is scaling fast, and thus it is intuitive that Amazon keeps deploying more servers into EC2. The measurement in 2008 [14] shows that there were three availability zones in the US east region. At present, the availability zones are expanded to four. Such expansion in availability zones also indicates that the business scale of EC2 is growing rapidly.

The measurement in 2008 [14] also shows 78 unique Domain0 IP addresses with 1785 m1.small instances, which means it only observed 78 physical machines that host m1.small service. Due to the evolution in EC2 management, we are no longer able to identify Dom0. However, we have identified at least 59 racks of servers that host m1.small instances. This suggests that the number of physical machines hosting m1.small instances is significantly larger than that in 2008. The enlarged pool provides EC2 with more flexibility to place incoming VMs, which is one of the reasons that it is now much more difficult to achieve co-residence than before.

4.4.2 Time locality

Time locality can help to achieve co-residence. Time locality means if two accounts launch instances simultaneously, it is more likely that some of these instances with time locality will be assigned to the same physical machine.

To verify whether such time locality exists in the current EC2, we performed another measurement. We set up four groups of experiments. In the first group, the two accounts always launch 20 VMs simultaneously. In the second group, the second account launches 20 VMs 10 minutes after the first account launches 20 VMs. In the third group, the launching time of the second account is one hour apart from that of the first account. In the fourth group, the second account launches VMs four hours after the first account. All instances are t1.micro type. In each group, the measurement terminates whenever a co-residence pair is observed and the number of instances required to achieve co-residence is recorded. All the experiments are repeated 5 times and the average is noted.

Figure 10 illustrates the number of instances required to achieve co-residence in each case. We can see that the efforts required to achieve co-residence do not vary significantly with the change of instance launching intervals. This implies that time locality seems to be very weak in the current EC2, which increases co-residence cost.

4.4.3 Dynamic assignment

In 2008, the IP addresses and instances in EC2 were assigned in a relatively static manner [14]. However, as we have demonstrated before, there are considerable mapping changes in our measurement, which indicates that the IP assignment has introduced a certain dynamism.

Meanwhile, in 2008, the instances were placed strictly based on the instance type, i.e., one physical machine can only host one type of instance [14]. In contrast, our measurement results show that such an assumption may not hold anymore. First, some small instances use internal IP addresses that were used by micro instances

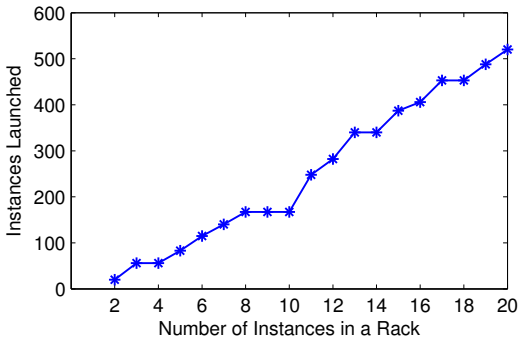

Figure 9: Instances launched to place certain number of instances within the same rack.

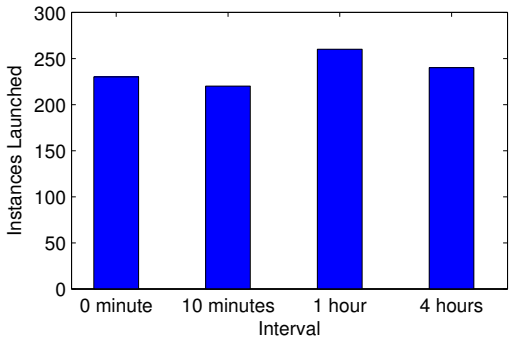

Figure 10: Effort to achieve co-residence with different time locality.

before. Second, during our measurement, by accident we observed that one live small instance has very close IP to a medium instance. We then attempted to build a covert channel between them. It turned out that the covert channel did work, which verifies that these two instances with different types are indeed located on a same physical machine. Following such an observation, in the rest of our rest measurement we also kept checking co-residence between different types of instances. Overall, five pairs of different-type co-residence instances are observed throughout our study. Our results indicate that in certain cases current VM placement policies in EC2 can mix different types of instances on one physical machine, potentially to reduce fragmentation. Such a policy also increases the difficulty of achieving co-residence.

5 The Impact of Network Management upon Co-residence

As network management plays a critical role in data center management, it has a significant impact on co-residence. On one hand, an attacker attempts to obtain as much networking information inside the cloud as possible to ease the gaining process of co-residence. On the other hand, the cloud vendors try to protect sensitive information while not degrading regular networking management and performance. In this section, we introduce the adjustments made by EC2 in network management during recent years to mitigate co-residence threat and the effectiveness of these approaches.

5.1 Methodology

To study the adjustment made by EC2 in network management, we performed large scale trace-routing. First, for the instances we booted, we performed "neighborhood trace-routing" from our instances to their "neighbors." Here we define neighbors as all those instances that share the /23 prefix of their private IP addresses with our source instances. Such trace-routing can inform us of the routing paths between an instance and other instances in the same rack and neighboring racks.

We next performed trace-routing from several of our instances (i.e., the instances we booted) to all the instances in a target list. We use the live host list from our scanning measurement (see Section 3.5 and Appendix A) as the target list. Trace-routing from our instances to over 650,000 target instances takes more than 8 days, but it can help us to understand network management in EC2 in a more comprehensive manner.

5.2 The evolution in routing configuration

The routing information has been leveraged to perform cloud cartography [14], which can further be used to launch co-residence-based attacks. However, our trace-routing results demonstrate that, as a response to cloud cartography, EC2 has adjusted its routing configurations to enhance security in the past few years. The adjustments we found are listed as follows.

5.2.1 Hidden Domain0

EC2 uses XEN as the virtualization technique in the cloud. According to the networking I/O mechanism of XEN [6], all the network traffic of guest VMs (instances) should travel through the privileged instance: Domain-0 (i.e, Dom0). Thus, Dom0 acts as the gateway of all instances on the physical machine, and all instances on this physical machine should have the same first-hop in their routing paths. Such Dom0 information provides an attacker with a very efficient probing technique: by simply checking the Dom0's IP addresses of two instances, one can know whether they are co-resident. Therefore, to prevent this Dom0 information divulgation, EC2 has hidden Dom0 in any and all routing paths, i.e. at present the Dom0 does not appear in any trace-routing results.

5.2.2 Hidden hops

To suppress cloud cartography enabled by trace-routing, EC2 has hidden certain hops in the routing paths. According to the work in May 2013 [13], traffic only needs to traverse one hop between two instances on the same physical machine and two hops between instances in

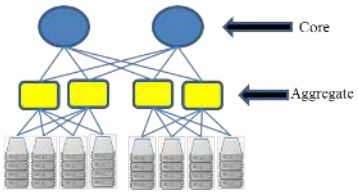

Figure 11: A common tree topology of a data center.

Figure 12: The topology with End of Row switch.

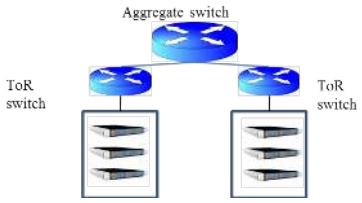

Figure 13: The topology with Top of Rack switch.

a same rack but not on the same physical machine. The paths between instances in different racks typically have 4 or 6 hops. However, our neighborhood trace-routing results show that the routing management has been changed in EC2.

First, a path of one hop does not necessarily indicate co-residence anymore. Our neighborhood trace-routing results show that an instance can have a very large number of 1-hop neighbors. For instance, one m1.small instance can have more than 60 1-hop neighbors. It is technically impractical to host so many instances on an m1 machine. To verify our hypothesis, we selected several pairs of instances with a 1-hop path and checked co-residence using covert channel construction. Our co-residence verification fails for most of these pairs, confirming that two instances with a 1-hop path do not necessarily co-locate on the same physical machine. This observation indicates that EC2 even hides the ToR switches in the routing path in some cases, leaving only one hop in the path between two instances in the same rack.

Second, we observed many odd-hop paths, accounting for 34.26% of all paths. In contrast, almost all the paths in the measurement conducted in May 2013 are even-hop [13]. This indicates that the network configuration of EC2 has changed since May 2013.

Third, the ToR switch of a source instance is shown as the first hop in the path, which indicates that the ToR switch should be an L3 router. However, we cannot observe the ToR switch of a target instance in the traces, implying that EC2 has configured the ToR switch to hide itself in the incoming traffic to the rack. Moreover, among our traces, we observed that 76.11% of paths have at least one hop filled with stars. The hops filled with stars can be a result of the configuration of certain devices such as L2 switches; it is also possible that EC2 has deliberately obscured those hops for security reasons. These paths with invisible or obscured hops significantly increase the difficulty of conducting cloud cartography.

5.3 Introducing VPC

To suppress the threat from internal networks, EC2 proposes a service called Virtual Private Cloud (VPC). VPC is a logically isolated networking environment that has a separate private IP space and routing configuration. After creating a VPC, a customer can launch instances into its VPC, instead of the large EC2 network pool. The customer can also divide a VPC into multiple subnets, where each subnet can have a preferred availability zone to place instances.

Moreover, EC2 provides instance types that are dedicated for VPC instances. These instance types include t2.micro, t2.small, and t2.medium. According to the instance type naming policy, instances with t2 type should be placed on those physical servers with the t2 model.

An instance in a VPC can only be detected through its public IP address, and its private address can never be known by any entity except the owner. Therefore, within a VPC, an attacker can no longer speculate the physical location of a target using its private IP address, which significantly reduces the threat of co-residence.

5.4 Speculating network topology

Besides routing configuration, the knowledge of network topology also helps to achieve co-residence, especially for high level co-residence such as rack-level. Figure 11 depicts the typical network topology in a data center. The core and aggregation switches construct a tree topology. Before connecting to the aggregate switches, there are two mainstream ways to connect servers in a rack/racks: End of Row (EoR) switches and Top of Rack (ToR) switches.

For EoR switches, as illustrated in Figure 12, servers of several racks are connected to the same EoR switch. To be more precise, an EoR switch can be a switch array including a group of interconnected switches. These switches can function as aggregate switches themselves. For ToR switches, as illustrated in Figure 13, all servers in a rack are first connected to a separate ToR switch, and then the ToR switch is connected to aggregate switches. Such a topology has currently become the mainstream network topology in a data center.

There are several variants of EoR topology, such as Middle of Rack (MoR) and ToR switch with EoR management. Meanwhile, there are other potential topologies such as OpenStack cluster in a data center. Therefore, we classify the network topology of a rack/racks into two classes: ToR connected and non-ToR connected. To identify whether a rack uses a ToR switch or a non-ToR switch, we analyze the neighborhood trace-routing results of multiple instances. Based on our analysis, we

proposed a method to identify the network topology of a rack, ToR-connected or non-ToR-connected.

ToR-connected: a rack that deploys ToR switches must satisfy all of the following conditions:

1. For an instance A in the rack, there should be at least one instance B that is only one hop away from A.
2. For an instance A in the rack, there should be at least 8 instances that are two hops away from A.
3. For any two instances A and B, if (i) conditions 1 and 2 hold for both A and B, (ii) the trace-routing path between A and B has no more than two hops, and (iii) for any instance C, the first hop in the trace-routing path from A to C is the same as the first hop in the path from B to C, then A and B are considered as being in the same ToR rack.
4. For an instance A in the rack, for any trace-routing path with A as source and length larger than 2, the first hop in the path should share the /16 prefix with the private IP address of A.

The IP address of the first hop (i.e., ToR switch's IP address) is used to differentiate two ToR racks.

Non-ToR-connected: a rack that deploys non-ToR switches must satisfy all of the following conditions:

1. For an instance A in the rack, there should be no instance B such that the path between A and B has two hops.
2. For an instance A in the rack, for any instance B in EC2, either (i) A and B are machine-level co-resident and the path between A and B has only one hop or (ii) the path between A and B has more than two hops.
3. For two instances A and B, if (i) conditions 1 and 2 hold for both A and B, (ii) A and B share the /24 prefix of their private IP, (iii) the trace-routing path between A and B has 4 or 6 hops, and (iv) for any instance C, the first hop in the path between A and C is the same as the first hop in the path between B and C, then A and B are considered as being in the same non-ToR rack.
4. For an instance A in the rack, for any trace-routing path with A as source and length larger than 2, the first hop in the path should not share the /20 prefix with the private IP address of A.

Again, the IP address of the first hop is used to differentiate two non-ToR racks.

In EC2, there are two "generations" of instances. The old generation carries all the instances with m1 type, and the new generation covers all the instances with other types. We applied our method on m1.small, m1.medium, m3.medium, and m3.large type, which cover both old-generation instances and new-generation instances.

Overall, we identified 59 distinct racks that host m1.small instances, 18 racks that host m1.medium instances, 22 racks that host m3.medium instances, and 10 racks that host m3.large instances. Among the 109 racks, there are only 14 racks identified as non-ToR-connected while the rest are ToR-connected. Among the 14 non-ToR racks, we observed 12 old-generation racks, in which 7 racks host m1.small instances and 5 racks host m1.medium instances, and only 2 new-generation racks host m3.medium instances.

Our results demonstrate that while both ToR racks and non-ToR racks exist in EC2, ToR-connected is the dominating topology in EC2. Moreover, it is evident that new-generation machines are more likely to be located in the ToR-connected topology, indicating that the ToR-connected topology has become the main trend. While the ToR-connected topology is easy to manage, the routing information is very straightforward since the first hop reveals which rack the instance is in. Such information can be leveraged by an attacker to achieve rack-level co-residence.

6 A New Battle in VPC

Using VPC, customers can protect their instances in an isolated network environment. However, VPC only logically isolates the networks. The instances from different VPCs may still share the same physical machine, leaving the opportunity to achieve co-residence. In this section, we first take an overview on the usage of VPC in EC2, and then we introduce a new method to attack instances that are hidden behind VPCs.

6.1 The overview of VPC usage

For those instances in the default networks of EC2, our inside scanner can obtain their private addresses via DNS lookups. However, the DNS query for an instance in a VPC will only return its public IP address. Therefore, the instances in a VPC can be easily identified by checking the DNS query results of our inside scanner, i.e., any instance whose private IP address cannot be detected by our inside scanner is an instance in a VPC. Figure 14 shows the VPC usage in EC2. As we can see, all instances in VPC are assigned public IP addresses in five different ranges: 107.20.0.0/14, 184.72.64.0/18, 54.208.0.0/15, 54.236.0.0/15, and 54.80.0.0/13. This implies that all instances in a VPC are managed in a uniform manner. On average, in each round of our probing we can observe 115,801 instances in a VPC, which are around 17% of all live instances observed, demonstrating that VPC is widely used in EC2 to protect instances.

6.2 Routing paths of VPC instances

Since a VPC should be treated as a private network, the routing policies for instances inside a VPC must be different from those in the default EC2 network. This routing difference can help us further understand the management of a VPC. To connect a VPC to the public Internet, a

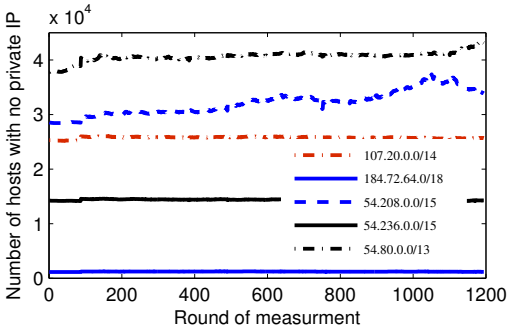

Figure 14: The live instances in VPCs.

customer must create a gateway and attach it to the VPC. The gateway must be included into the route table of the VPC. All traffic from or to the Internet must go through the gateway, but the traffic inside EC2 does not require the gateway to be involved.

Besides the basic understanding of the routing configuration of a VPC, we also need to know how a VPC is connected with the default EC2 network and other VPCs. We created several VPCs with two different accounts. The instances with different types are launched into these VPCs. Trace-routing is performed in four different ways: (1) trace-routing from an instance in a VPC to another instance in the same VPC, (2) trace-routing from an instance in a VPC to an instance in another VPC, (3) trace-routing from an instance in a VPC to an instance in the default EC2 network, and (4) trace-routing from an instance in the default EC2 network to an instance in a VPC.

6.2.1 Routing within VPC

Routing inside the same VPC is expected to be simple. We performed trace-routing between two instances in the same VPC, using both private and public IP addresses. The results show that trace-routing with private IP or public IP addresses will yield different routing paths. If trace-routing is performed with the private IP of the target instance, the result path has only one-hop, i.e., the direct connection to the destination, which is reasonable. However, if trace-routing is performed with the public IP of the target, trace-routing will return two hops with the first hop obscured with stars. Apparently, EC2 intentionally hides some routing information. The routing information between the two instances within the same VPC is made transparent to customers. Such obscuration disables a customer from speculating the physical location of the instances.

As discussed in Section V, even within the same VPC, two instances can be located in different "subnets." We also performed trace-routing between two instances in the same VPC but in different subnets. The resulting paths do not differ from the paths between two instances within the same subnet.

6.2.2 Routing between VPCs

The traffic between instances in different VPCs should traverse multiple switches and routers. Surprisingly, we found that any routing path between any two instances in any two different VPCs only has two hops: the first hop is obscured and the second hop is the destination. EC2 once again obscures the routing path between VPCs to prevent an adversary from revealing sensitive information of a VPC, e.g., the IP address of a gateway.

6.2.3 Routing from VPC to default EC2 network

Although instances in a VPC no longer share a private network with the default pool of EC2, the switches/routers that connect VPCs might still be physically connected to the other switches/routers in the data center. How EC2 routes the traffic between instances in a VPC and instances in the default EC2 network can reveal its network topology to some extent. Figure 15 shows a sample trace-routing result from an instance in a VPC to an instance in the default EC2 network. We can see that the first two hops of the path are obscured. This prevents us from knowing the switch/router that connects the VPC, thereby hiding the physical location of VPC instances. However, we can still see parts of the path and can infer the end-to-end latency based on the trace-routing result.

6.2.4 Routing from default EC2 network to VPC

Figure 16 shows a sample trace-routing result from an instance in the default EC2 network to an instance in a VPC. The path is almost symmetric to the path from a VPC to the default EC2 network. Again, the last two hops before reaching the destination are obscured to hide the information of the router/switch.

Overall, EC2 manages a VPC in a transparent fashion, i.e., to a customer it should look like all instances in a VPC are connected by a dedicated switch, just like a real private network. However, instances in the same VPC are not physically located together. These instances are still located in different racks and are connected to different ToR or EoR switches. Thus, the traffic inside a VPC might still traverse multiple switches/routers. Similarly, the traffic between an instance in a VPC and an instance in the default EC2 network can have a similar path to the traffic between two instances in the default EC2 network. However, EC2 hides or obscures certain hops in the path to provide the image of "private network."

6.3 Co-residence in VPC

The traditional way of achieving co-residence relies on the knowledge of private IP address to seek potential candidates. With VPC, this approach no longer works as VPC hides the private IP address of an instance. An alternative is to infer the physical location of a target based on

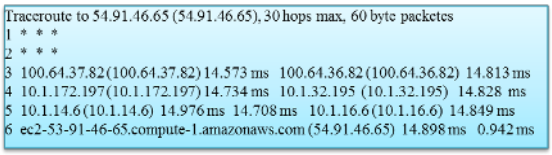

Figure 15: A sample trace-routing result from an instance in VPC to an instance in EC2.

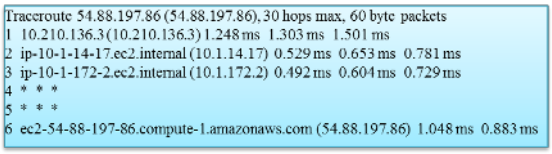

Figure 16: A sample trace-routing result from an instance in EC2 to an instance in VPC.

the routing paths to the target. Unfortunately, our trace-routing results show that sensitive information of a routing path is obscured by EC2, and therefore it also does not work well.

However, in our trace-routing results we found that the end-to-end latency to and from an instance in a VPC varies with different instance types and the location of the instance. This latency variation can be leveraged to help an attacker speculate the type and location of a target instance. Moreover, while performing trace-routing between an instance in a VPC and an instance in the default EC2 network, the number of hops required is not obscured. Therefore, the number of hops in a path can also be leveraged to derive useful information for achieving co-residence.

Based on our measurement analysis, we propose a new method to achieve co-residence with instances in a VPC. It has two steps: (1) speculate the type and availability zone of a target and (2) launch probing instances with the same type in the same availability zone and perform co-residence verification.

6.3.1 Type and zone speculation

We collected statistical data of the end-to-end latency between a pair of instances with different types and in different zones. Table 2 shows part of the end-to-end latency statistics. Each row represents an instance in a VPC with a certain type and availability zone preference. Each column stands for an instance in the default EC2 network with a certain type and availability zone preference. Each value in the table is calculated as the average of 50 samples. Each sample is obtained with a distinct instance pair and is averaged over five rounds of latency measurement. With this latency table, we are able to construct a latency vector for each target instance in a VPC and use the latency vectors to speculate the type and availability zone of a target.

There are three availability zones and each zone has six types: t1.micro, m1.small, m1.medium, m1.large, m3.medium, and m3.large. Thus, the complete version of Table 2 has 18 rows and 18 columns, which can be found in our technical report [22]. Note that each row in the table can represent a latency vector, and such a latency vector derived from our controlled sampling is called a *baseline vector*.

In each different availability zone, we randomly select an instance for each different type, resulting in 18 (3×6) sample instances in total for testing type and zone speculation. For each target in a VPC, we perform trace-routing from each of our sample instances to the target for 5 times and record the average end-to-end latency of each pair. Such measurement can provide us 18 end-to-end latency values, which constitute an input vector of length 18. We then calculate the cosine similarity between the input vector and these 18 baseline vectors. The baseline latency vector that has the highest similarity with the target input vector is selected, and we can speculate that the target instance has the same $\{$instance type, availability zone$\}$ as the instance in the selected baseline vector.

6.3.2 Verifying co-residence

To achieve co-residence with an instance in a VPC, our probing instances are also launched in a VPC. There are two reasons that we do not use the instances in the default EC2 network as probing instances. First, it is possible that EC2 uses a separate algorithm to place instances in a VPC. In other words, compared to an instance in the default EC2 network, an instance in a VPC may have a better chance to achieve co-residence with an instance in another VPC. Second, as we have observed, the end-to-end latency between two instances in two different VPCs is more stable than the latency between an instance in the default EC2 network and an instance in a VPC, which allows us to leverage latency for pre-filtering.

Similar to verifying co-residence in the default EC2 network, verifying co-residence in a VPC also includes two steps: pre-filtering and covert channel construction. While the way of using covert channel construction to confirm co-residence remains the same, the pre-filtering process in a VPC is different.

To verify whether an attack instance is co-resident with a target, we rely on two rounds of pre-filtering to screen out irrelevant candidates. First, we perform trace-routing from our 18 sample instances to our attack instance and the target instance. If any path from the sample instance to the attack instance is not equivalent to the corresponding path from the sample instance to the target in terms of number of hops, this attack instance is abandoned.

Second, if all the paths match in the number of hops, we measure end-to-end latency between our attack instance and the target instance. Figure 17 shows a sample latency distribution between an instance in a VPC

Table 2: End to end latency between different instances.

	1a-t1.micro	1a-m1.small	1a-m1.medium	1b-t1.micro	1b-m1.small	1b-m1.medium
1a-t1.micro	1.224ms	1.123ms	1.025ms	2.237ms	2.221ms	2.304ms
1a-m1.small	1.361ms	1.059ms	1.100ms	2.208ms	2.055ms	2.198ms
1a-m1.medium	1.165ms	1.102ms	0.986ms	2.211ms	2.060ms	1.988ms
1b-t1.micro	2.101ms	2.235ms	2.188ms	1.108ms	1.243ms	1.202ms
1b-m1.small	2.202ms	2.003ms	2.190ms	1.131ms	0.968ms	1.048ms
1b-m1.medium	2.087ms	2.113ms	1.965ms	1.088ms	1.023ms	0.855ms

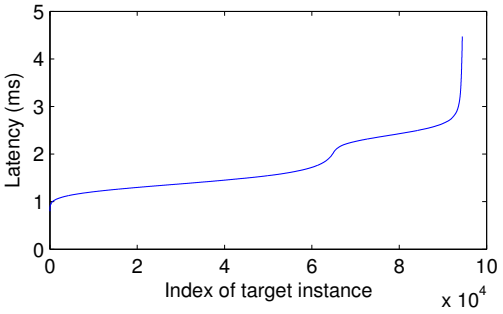

Figure 17: End-to-end latency between an instance in VPC and all other instances in other VPCs in EC2.

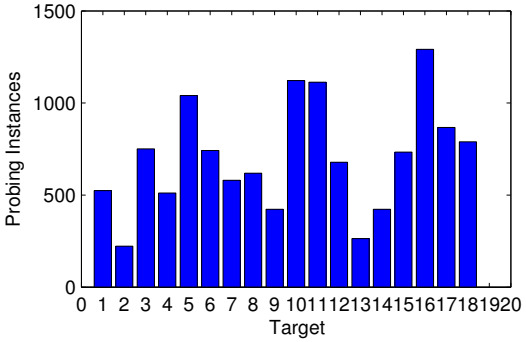

Figure 18: The effort for co-residence with instances in VPC.

with the micro type in availability zone 1a to all live VPC instances in EC2. As we can see, most end-to-end latency values (over 99%) are above 1ms, and in very rare cases (below 0.1%) the latency is below 0.850ms. We perform such latency measurement from 18 sample VPC instances with different types in different availability zones, and similar distribution is repeatedly observed. Based on such observations and the heuristics that instances located on the same physical machine should have lower latency than instances located in a different physical location, we set a latency threshold for each type of instance in each availability zone. The threshold is selected so that for an instance in a VPC with certain type and availability zone, the end-to-end latency between the instance and 99.9% of all other VPC instances should be above the threshold. For example, based on our measurement introduced above, if we speculate that the target VPC instance is located in availability zone 1a with micro type, the latency threshold is set to 0.850ms. Only if the end-to-end latency between a probing instance and a target instance is below the threshold, will the probing instance be considered as a co-residence candidate.

If the probing instance passes the two rounds of filtering, we will perform covert-channel construction to confirm co-residence.

6.4 VPC co-residence evaluation

To verify the feasibility of our VPC co-residence approach, we conducted a series of experiments in EC2. We first tested whether our approach can speculate the type and availability zone of a target instance correctly. We launched VPC instances in three availability zones with six different types. For each combination, 20 instances were launched. We applied our approach to speculate the type and availability zone of the target. If both the type and availability zone are correctly inferred, we consider that the target instance is correctly identified. Table 3 lists our evaluation results. Each number in the table indicates the number of the successfully identified instances among the 20 launched instances for a zone-type combination (e.g., 1a-t1.micro means t1.micro instances launched in the us-east-1a zone). The results show that our type/zone speculation can achieve an accuracy of 77.8%.

We then evaluated the overall effectiveness of our approach for achieving co-residence. We launched 40 instances in one VPC, with different types and availability zones. We performed the full process of achieving co-residence with VPC instances.

First, we measured the effectiveness of our two-stage filtering technique. Among all the probing instances we launched, 63.2% of them did not pass the first step filtering. For the second stage, our technique filtered out 97.9% of the instances that passed the first stage filtering. For all the instances passed the two-stages filtering, 17.6% of them passed the covert-channel verification, which are the instances actually co-resident with the target.

Eventually, among 40 instances, we successfully achieved co-residence with 18 of them. Figure 18 illustrates the effort we paid to achieve co-residence, showing that to achieve co-residence in VPC is not an easy task. An attacker may need to launch more than 1,000 probing instances and such a process can take many hours.

Overall, we are the first to demonstrate that an attacker can achieve co-resident with a target inside a VPC with high cost, and hence VPC only mitigates co-residence threat rather than eliminating the threat all together.

Table 3: The number of successfully identified targets.

	1a-t1.micro	1a-m1.small	1a-m1.medium	1a-m1.large	1a-m3.medium	1a-m3.large
Success	16	13	18	14	16	17
	1b-t1.micro	1b-m1.small	1b-m1.medium	1b-m1.large	1b-m3.medium	1b-m3.large
Success	13	13	19	16	20	17
	1d-t1.micro	1d-m1.small	1d-m1.medium	1d-m1.large	1d-m3.medium	1d-m3.large
Success	12	18	15	13	14	18

7 A More Secure Cloud

Based on our measurement analysis, we have proposed some guidelines towards more secure IaaS cloud management.

First, the cloud should manage the naming system properly. In general, a domain name is not sensitive information. However, EC2's automatic naming system reveals its internal space. In contrast, Azure and Rackspace employ flexible naming systems that can prevent automatic location probing. However, automatic domain name generation is more user-friendly since it allows a user to launch instances in batch, while a customer can only launch instances one by one in Azure and Rackspace. Moreover, automatic domain name generation can help an IaaS vendor manage the cloud more efficiently. To balance management efficiency and security, we suggest that IaaS clouds integrate automatic domain name generation with a certain randomness. For example, a random number that is derived from the customer's account information can be embedded into the EC2 default domain name. This improved naming approach can prevent location probing while not degrading management efficiency.

Second, it is controversial to publish all IP ranges of a cloud. With the introduction of ZMap [10], it is not difficult to scan all public IPs in the cloud. We have demonstrated that such scanning can cause serious security concerns.

Third, the routing information should be well-protected. While trace-routing is a tool for a customer to diagnose a networking anomaly, it can also be exploited by an attacker to infer the internal networking information of the cloud. However, the approach taken by Azure and Rackspace is too strict. The prohibition of networking probing deprives a customer from self-diagnosis and self-management. A good trade-off is to show only part of the paths, but always obscure the first hop (ToR) and the last second hop.

Fourth, VM placement should be more dynamic and have more constraints. Locality reduction will make it more difficult for an attacker to locate a target. IaaS vendors can also leverage some historical information of a user's account to prevent the abuse of launching instances. While EC2 has significantly increased the difficulty of achieving machine-level co-residence, it is also necessary to suppress rack-level co-residence in the future.

8 Conclusion

We have presented a systematic measurement study on the co-residence threat in Amazon EC2, from the perspectives of VM placement, network management, and VPC. In terms of VM placement, we have demonstrated that time locality in VM placement is significantly reduced and VM placement in EC2 becomes more dynamic, indicating that EC2 has adjusted its VM placement policy to mitigate co-residence. Regarding network management, by conducting a large-scale trace-routing measurement, we have shown that EC2 has refined networking configurations and introduced VPC to reduce the threat of co-residence. We have also proposed a novel method to identify a ToR-connected or non-ToR-connected topology, which can help an attacker to achieve rack-level co-residence. As the first to investigate the co-residence threat in VPC, on one hand, we have confirmed the effectiveness of VPC in mitigating the co-residence threat. On the other hand, we have shown that an attacker can still achieve co-residence by exploiting a latency-based probing method, indicating that VPC only mitigates co-residence threat rather than eliminating the threat.

9 Acknowledgement

We would like to thank our shepherd Chris Grier and the anonymous reviewers for their insightful and detailed comments. This work was partially supported by ONR grant N00014-13-1-0088.

References

[1] Amazon elastic compute cloud (ec2). `http://aws.amazon.com/ec2/`.

[2] Google cloud platform. `https://cloud.google.com/compute/`.

[3] Instance types in ec2. `http://aws.amazon.com/ec2/instance-types/`.

[4] Microsoft azure services platfor. `http://www.microsoft.com/azure/default.mspx`.

[5] AVIRAM, A., HU, S., FORD, B., AND GUMMADI, R. Determinating timing channels in compute clouds. In *Proceedings of ACM CCSW'10*, pp. 103–108.

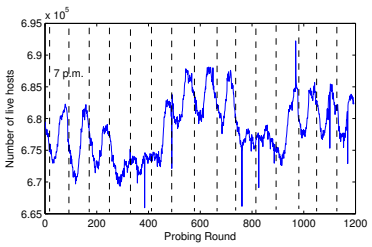

Figure 19: Number of live instances in EC2 US east region.

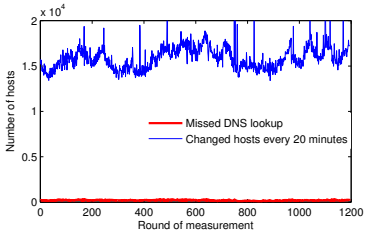

Figure 20: Number of changed instances between each round of measurement and number of missed DNS lookups in each round.

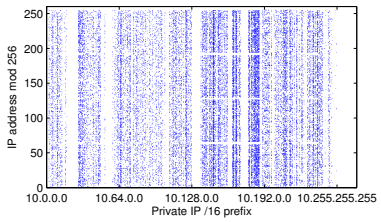

Figure 21: The private IP addresses whose mappings to public IP have changed.

[6] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Proceedings of ACM SOSP'03*.

[7] BATES, A., MOOD, B., PLETCHER, J., PRUSE, H., VALAFAR, M., AND BUTLER, K. Detecting co-residency with active traffic analysis techniques. In *Proceedings of ACM CCSW'12*.

[8] BERMUDEZ, I., TRAVERSO, S., MELLIA, M., AND MUNAFO, M. Exploring the cloud from passive measurements: the amazon aws case. In *Proceedings of IEEE INFOCOM'13*, pp. 230–234.

[9] BIRKE, R., PODZIMEK, A., CHEN, L. Y., AND SMIRNI, E. State-of-the-practice in data center virtualization: Toward a better understanding of vm usage. In *Proceedings of IEEE/IFIP DSN'13*.

[10] DURUMERIC, Z., WUSTROW, E., AND HALDERMAN, J. A. Zmap: Fast internet-wide scanning and its security applications. In *Proceedings of USENIX Security'13*, pp. 605–620.

[11] HE, K., FISHER, A., WANG, L., GEMBER, A., AKELLA, A., AND RISTENPART, T. Next stop, the cloud: understanding modern web service deployment in ec2 and azure. In *Proceedings of ACM IMC'13*, pp. 177–190.

[12] KIM, T., PEINADO, M., AND MAINAR-RUIZ, G. System-level protection against cache-based side channel attacks in the cloud. In *Proceedings of USENIX Security'12*.

[13] LACURTS, K., DENG, S., GOYAL, A., AND BALAKRISHNAN, H. Choreo: network-aware task placement for cloud applications. In *Proceedings of ACM IMC'13*.

[14] 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 *Proceedings of ACM CCS'09*, pp. 199–212.

[15] VARADARAJAN, V., RISTENPART, T., AND SWIFT, M. Scheduler-based defenses against cross-vm side-channels. In *Proceedings of USENIX Security'14*.

[16] VARADARAJAN, V., ZHANG, Y., RISTENPART, T., AND SWIFT, M. A placement vulnerability study in multi-tenant public clouds. In *Proceedings of USENIX Security'15*.

[17] WANG, G., AND NG, T. E. The impact of virtualization on network performance of amazon ec2 data center. In *Proceedings of IEEE INFOCOM'10*.

[18] WANG, L., NAPPA, A., CABALLERO, J., RISTENPART, T., AND AKELLA, A. Whowas: A platform for measuring web deployments on iaas clouds. In *Proceedings of ACM IMC'14*.

[19] WU, Z., XU, Z., AND WANG, H. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Proceedings of USENIX Security'12*, pp. 159–173.

[20] XU, Y., BAILEY, M., JAHANIAN, F., JOSHI, K., HILTUNEN, M., AND SCHLICHTING, R. An exploration of L2 cache covert channels in virtualized environments. In *Proceedings of ACM CCSW'11*, pp. 29–40.

[21] XU, Y., MUSGRAVE, Z., NOBLE, B., AND BAILEY, M. Bobtail: avoiding long tails in the cloud. In *Proceedings of USENIX NSDI'13*, pp. 329–342.

[22] XU, Z., WANG, H., AND WU, Z. Technical Report: WM-CS-2015-03. `http://www.wm.edu/as/computerscience/documents/cstechreports/WM-CS-2015-03.pdf`.

[23] XU, Z., WANG, H., XU, Z., AND WANG, X. Power attack: An increasing threat to data centers. In *Proceedings of NDSS'14*.

[24] ZHANG, Y., JUELS, A., OPREA, A., AND REITER, M. K. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of IEEE S&P'11*, pp. 313–328.

[25] ZHANG, Y., JUELS, A., REITER, M. K., AND RISTENPART, T. Cross-tenant side-channel attacks in paas clouds. In *Proceedings of ACM CCS'14*, pp. 990–1003.

[26] ZHANG, Y., AND REITER, M. K. Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *Proceedings of ACM CCS'13*, pp. 827–838.

A Business scale of EC2

Figure 19 illustrates the number of all the detected live instances in EC2 US east region during the measurement period. We can see that the business scale in EC2 US east region is very impressive. Our scanning can always detect more than 650,000 live instances in the cloud. During the peak time, we can detect almost 700,000 live instances. It is noteworthy that our system only scans some common ports. Besides the instances we detected, there are some instances with no common ports opened or within the VPC that do not have public IP addresses. Thus, the real number of live instances in the cloud could be even larger.

Table 4 lists the break-down statistics, showing the number of instances hosting a certain service on average. It is obvious that web service still dominates the usage in IaaS. Most customers rent the instances to host their web services. Among these web services (i.e., HTTP), more than half of them deploy HTTPS at the same time. Since the default way of accessing an instance in EC2 is through SSH, the number of instances listening on port 22 is the second largest. There are also considerable instances hosting FTP service, DNS service, and database service (MYSQL+SQL). For the rest of services, the number of instances hosting them are less significant.

Table 4: Number of instances hosting a certain service

	FTP	SSH	Telnet	SMTP	WHOIS	DNS	DHCP	Finger	HTTP	SQL	HTTPS	MYSQL
Live instances	24,962	327,294	350	18,376	305	3,392	15	68	441,499	48	261,446	25,872

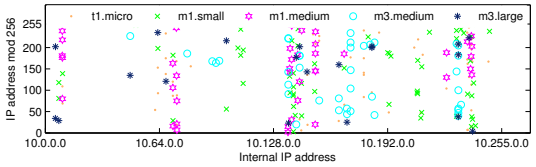

Figure 22: The distribution of internal IP addresses of instances with different types in availability zone us-east-1a.

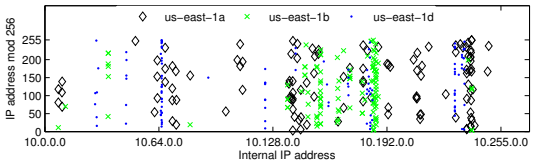

Figure 23: The distribution of internal IP addresses of instances in different availability zones.

B Dynamic environment of EC2

Our measurement can also reflect the dynamic environment of EC2 to some extent. First, as shown in Figure 19, the number of live instances varies over time within a day. We observed a similar pattern each day: the peak time is around 5 p.m. (EST) while the service reaches a valley around 4 a.m. (EST). Despite this diurnal pattern, the difference in the number of live instances between peak and valley is not as significant as we expected. There are only 1,000 more live instances at peak than valley, which is relatively small considering the overall 650,000 live instances. The diurnal pattern is reasonable, as 4 a.m. EST is very early morning for the US east coast and it is also midnight for the US west coast. It is intuitive that at this time period fewer users are using EC2. The small difference between peak and valley can be explained from two aspects. First, most instances run stable services such as web and database services. These instances remain active all the time. Second, although the data center is located in the US, the customers are distributed all around the world. For instance, Bermudez et al. [8] demonstrated that the Virginia data center is responsible for more than 85% of EC2 traffic in Italy. The time of 4 a.m. on the US east coast is 10 a.m. in Italy when customers are very active there.

We are also interested in how dynamic the cloud environment is. Figure 20 illustrates how many instances are shutdown, newly booted, or re-located between each round of measurement. We can see there are more than 15,000 hosts that are changed every 20 minutes, indicating that EC2 is a very dynamic environment with tens of VMs booted and shut down every second.

Besides the dynamics of live instances, we are also interested in the networking dynamics. During our measurement, we observed overall 975,032 distinct private IP addresses and 1,024,589 distinct public IP addresses. We recorded all the mappings from public IP to private IP and the mappings from private IP to public IP during our measurement. We also recorded the mappings that are changed during the measurement period. Over the course of our 15-day measurement, 103,242 mappings changed. This implies that EC2 has likely recruited dynamic NAT for address translation.

Figure 21 shows the private IP addresses that are included in the changed mappings. It is clear that the IP address pool in the cloud is dynamic as well. The density of the IPs in a certain range is significantly higher than other areas. This range of private IPs are mostly assigned to micro and small instances. Since micro and small instances are usually used for temporary purposes, ON/OFF operations on them are more frequent, leading to more frequent changes in private-public IP mappings.

C VM placement locality in EC2

To investigate the VM placement locality in EC2, we launched numerous instances with different types and in different availability zones to study whether the type or zone will impact the physical location of an instance.

Figure 22 illustrates the private IP distribution of some sample instances with different types in zone us-east-1a. The IP distribution exhibits a certain type locality. We can see from the figure that the instances of the same type tend to have closer internal IPs, i.e., they are more likely to be placed physically close to one another. However, compared with corresponding results in 2008 [14], we can see that such type locality has been significantly weakened.

We also study how availability zone could affect VM placement. Figure 23 illustrates the internal IP distribution of instances in different availability zones. As we can see, VM placement still has availability zone locality, i.e., instances in the same zone are more likely to have their internal IP addresses located within a certain range. However, such locality is also much weaker than in 2008 [14].