Cloudoscopy: Services Discovery and Topology Mapping

Amir Herzberg*
Computer Science Department
Bar Ilan University
Ramat Gan, Israel

Haya Shulman†
Fachbereich Informatik
Technische Universität Darmstadt/EC-SPRIDE
Darmstadt, Germany

Johanna Ullrich‡
SBA Research gGmbH
Wien, Austria

Edgar Weippl§
SBA Research gGmbH
Wien, Austria

ABSTRACT

We define and study *cloudoscopy*, i.e., exposing sensitive information about the location of (victim) cloud services and/or about the internal organisation of the cloud network, in spite of location-hiding efforts by cloud providers. A typical cloudoscopy attack is composed of a number of steps: first expose the internal IP address of a victim instance, then measure its hop-count distance from adversarial cloud instances, and finally test to find a specific instance which is close enough to the victim (e.g., co-resident) to allow (denial of service or side-channel) attacks. We refer to the three steps/modules involved in such cloudoscopy attack by the terms *IP address deanonymisation*, *hop-count measuring*, and *co-residence testing*.

We present specific methods for these three cloudoscopy modules, and report on results of our experimental validation on popular cloud platform providers. Our techniques can be used for attacking (victim) servers, as well as for benign goals, e.g., optimisation of instances placement and communication, or comparing clouds and validating cloud-provider placement guarantees.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network topology*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server, Network operating systems*

*amir.herzberg@gmail.com
†haya.shulman@gmail.com
‡jullrich@securityresearch.at
§eweippl@sba-research.org

CCSW'13, November 8, 2013, Berlin, Germany.

http://dx.doi.org/10.1145/2517488.2517491.

Keywords

Cloud security; timing side-channels; cloud tomography; I/O performance; hardware interrupts; low rate attacks

1. INTRODUCTION

Cloud computing services allow scalable and efficient sharing of resources, via the use of virtualisation. Resources, such as computing, storage, network and other services are allocated only when needed, and on a pay-per-use basis. Tenants can rent virtual machines (VMs) according to their needs with the required amount of storage, CPU, and other resources. These advantages are predominant factors for the (growing) popularity of cloud computing.

However, placement of services and resources in clouds and sharing thereof (i.e., multi-tenancy) raises security and trust concerns. Indeed, there are significant security issues, that still need to be addressed, most notably, availability and privacy of the data, e.g., see [5, 8, 11]. These issues serve as a deterrent to wide adoption of the cloud computing model by organisations. One natural question is whether customers can trust the cloud provider to operate fairly, as expected and guaranteed; In this work, we focus on different aspects, related to security threats that stem from network sharing with other cloud tenants. Specifically, we address the following questions:

- Which defenses can cloud providers deploy to prevent attacks on their customers (by external attackers and by malicious customers)?
- What is the significance of each defense, and in particular, which defenses can be substituted or complemented by a customer-implemented defense mechanism (and at what costs)?
- How can customers check such defenses on a specific cloud (e.g., to choose among clouds or to verify guarantees of a cloud provider)?
- Which defenses are deployed by popular clouds? Do popular clouds provide the same defenses, or are there significant differences?

Physical co-residence with other cloud tenants poses a particular risk. The ability to attack cloud-based resources, often depends on whether the attacker is able to detect the location of victim services, and has the ability to place its own

instances on the same physical cloud host ('co-residence') or on the same network segment. For example, [31] showed that sharing of resources between instances of different customers, may allow attacks on one cloud application, by another application running on the same physical machine.

Therefore, cloud providers usually take steps to 'hide' the exact identity of the physical server running the hosted cloud services as well as the structure of their internal network (and of particular servers within it). In particular, to prevent identification of services placement, clouds often use a random mapping between external IP addresses (of hosted services) and the internal IP addresses (of the virtual instances running the services), and/or employ other separation techniques, e.g., see [29]. On the other hand, there are legitimate uses for such 'location information', for optimisation or even security, e.g., to ensure that, as promised by the cloud provider, customer instances are isolated from potential attackers and vulnerabilities.

As we show in our work, one of the factors allowing services' placement location is internal communication between the tenants. Indeed, cloud providers that allow internal communication between tenants often facilitate our attacks. However, merely blocking internal communication may not be a plausible solution, since there is an increasing amount of tenant-to-tenant and tenant-provider communication, which amounts up to 35% of the total datacenter traffic and is expected to continue growing in diversity and volume, see [2, 28].

In this work we investigate *cloudoscopy*, i.e., exposing sensitive information about the location of (victim) cloud services and/or about the internal organisation of the cloud network, in spite of location-hiding efforts by cloud providers. The term *cloudoscopy* reflects the fact that we utilise nodes inside the cloud networks to perform our measurements, i.e., services placement inference and topology discovery. Cloudoscopy can be compared and contrasted with *Internet tomography*, [9], which measures its internal structure, utilises nodes only on the edge of the Internet.

We identify and propose three modules comprising a cloudoscopy attack: first expose the internal IP address of a victim, then measure its hop-count distance from adversarial cloud instances, and finally test to find a specific instance which is close enough to the victim (e.g., co-resident) to allow (denial of service or side-channel) attacks.

We next describe our cloudoscopy modules, and present novel technique to implement each of them:

IP address deanonymisation allows the attacker to discover the internal IP address mapped to a given external IP address.

Hop-count measuring is a simple, efficient technique that allows the attacker to learn the number of network segments between an instance of the attacker, and other instances (usually, of victims). This allows detection when the two instances are on the same subnet (or on the same physical machine); it also allows learning the subnet topology of the cloud.

Co-residence verification allows the attacker to test if two instances *co-reside*, i.e., deployed on the same physical host.

These three modules may often be used in sequence. First, the attacker uses IP address deanonymisation to find the internal IP address of some victim service (using the publicly known external IP address). Next, the attacker uses hop-count measuring to find the path to the victim instance. Finally, attacker uses co-residence verification to test collocation with a victim service.

For instance, assume a victim instance, residing on the same physical host as the attacker, runs, e.g., a vulnerable operating system version, or a vulnerable web server, for which an exploit is available. A malicious VM can exploit these bugs to attack the virtualisation software, or may abuse them to attack tenants running on the same host, e.g., [41]. Exploiting such an attack vector would give the attacker the ability to attack or access other virtual machines and therefore breach confidentiality, integrity, and availability of the other virtual machines' code or data. Furthermore, by exploiting a bug, the attacker may be able to take control over the lower layers of the platform. For instance, there have been dozens of reported vulnerabilities from CVE for Xen hypervisor, e.g., it was found that a Xen3 hypervisor contained buffer overflow exploits and other bugs allowing code execution in administrative domain, e.g., [39, 40]. It is also possible to abuse cross VM leakage to learn secret data, such as secret keys, via side channels, e.g., [3, 26, 31], or to launch network attacks, e.g., DoS, against a victim.

Previous work, in particular Ristenpart *et al*, [31], depended on IP address deanonymisation, but used only simple techniques to achieve it, e.g., traceroute/ping. However, following their publication, popular clouds blocked these simple mechanisms. No other IP address deanonymisation techniques were proposed so far. In this work we present techniques that enable IP address deanonymisation.

We propose two techniques for IP address deanonymisation: *interrupt-overloading side-channel* and *server-bounce scan*. The server-bounce scan technique uses the fact that in some protocols, e.g., SMTP, servers open a connection using a domain name from an incoming connection. This allows server-bounce scan to be extremely effective, requiring only very few packets for discovery of service that runs on some instance, but limits it to specific types of servers (e.g., SMTP). The other technique, interrupt-overloading side-channel, is general and not protocol specific; interrupt-overloading side-channel is based on overloading the server with multiple interrupts due to transmission of a burst of (minimal size) packets. Both of our address discovery mechanisms are limited: they do not apply to cloud providers that block connections from internal addresses, e.g., Microsoft Azure. However, often internal communication has to be allowed, e.g., to reduce costs or to improve quality of service, and there is an increasing amount of tenant-to-tenant communication, [2, 28]; indeed, some popular cloud providers allow communication between internal hosts, e.g., Amazon AWS, Rackspace Cloud Servers.

Our interrupt-overloading side-channel exploits hardware interrupts that I/O devices generate to signal packet arrival to the kernel. Hardware interrupts are known to impose significant overhead [23, 44], and there is research proposing techniques to reduce interrupts, in standard setting, [33, 47], and in virtualised environments [25, 45] (where the interrupt overhead is even increased, see [4]), by reducing the number of interrupts. However, this is not trivial and may have a negative impact on the performance of the system [35].

We also propose techniques for hop-count measuring and co-residence verification, which apply also to cloud providers that block internal communication.

We found that Amazon AWS EC2 and Rackspace Cloud allow (all) communication between internal hosts, and so our IP address deanonymisation attacks apply to them. In contrast, Microsoft Azure blocks (all) communication between internal hosts, and Google Compute Engine restricts internal communication to hosts belonging to the same project. Restricting communication to internal hosts prevents IP address deanonymisation, but does not prevent hop counting. We summarise our attacks against different cloud providers in Table 1.

One of the implications of our work is that internal communication between tenants introduces security vulnerabilities which should be considered when designing defenses.

Our Contributions

We study vulnerabilities in cloud networks, and present techniques that enable attackers: (1) to perform victim IP address deanonymisation, and correlation between internal and external IP addresses, (2) to infer information about the cloud network topology and (3) to learn proximity to the victim and even obtain information about the cloud network topology. We outline the techniques we used to elicit various side channels, and apply them for our attacks, and provide our recommended countermeasures.

We introduce a new technique, interrupt-overloading side-channel, which can be used to elicit side channel allowing to infer placement information. The applications of our technique can be twofold: (1) for benign purposes, to verify the service which the cloud provider guarantees to supply, and (2) for malicious purposes for discovery of victim hosts within the cloud. We show how to apply interrupt-overloading side-channel for address and for co-residence verification of victim hosts.

We also show, that often protocol specific techniques exist, which can be used for address deanonymisation, and design a method enabling location of Email servers hosted in the cloud.

We then propose a TTL scan technique which, once the private address of a victim is known, allows to reconstruct a path to that instance, and can also be used to infer information about cloud network topology.

Cloud Provider	Address Deanonymisation	Hop Count	Coresidence Verification
Amazon (EC2)	√	√	√
Rackspace Cloud	√	√	√
Microsoft Azure	X	X	√
Google Compute Engine	X	same project	√

Table 1: Summary of attack techniques that apply to selected (popular) cloud providers.

2. RELATED WORK

Recently there was a surge of works showing that logical isolation between the virtual machines (VMs) may not prevent side channel attacks and it may often be possible to bypass isolation between virtual machines running on the same physical host, [3, 31, 38, 46, 49, 50]. These attacks exploit shared resources, most notably caches, to elicit side channels, allowing to infer information about sensitive data and computation performed by the coresident victim, and are typically applied to extract the secret keys, [1,10,12,43].

The side channels are inherent in the virtualisation concept of sharing a common physical resource among a number of virtual machines. The hypervisor interacts with the virtual machines, emulates the physical resources to them and coordinates access to the shared resources between the virtual machines. A malicious virtual machine can exploit this interaction to attack the hypervisor or other virtual machines that reside on the same physical host. This may often provide the attacker the ability to attack other virtual machines, and, e.g., breach confidentiality, integrity, and availability of the other tenants' instances services, code or data. Such vulnerabilities make many companies hesitant about moving to the cloud.

There is a wide body of research proposing defenses against attacks that exploit multitenancy, most notably works that attempt to improve virtualisation security, including inter-VM isolation, and security of hypervisor communication with the VMs. There are also mechanisms for detection of anomalies, and proposals for alternative cloud architectures, secure designs of cloud storage, as well as secure cloud software and hardware designs, e.g., [15, 24]. We next survey some of the proposals.

There are many works that attempt to improve the security of the virtualisation systems, e.g., [48] propose a transparent and backward-compatible approach, based on nested virtualisation, to protect the privacy and integrity of customers' virtual machines on commodity virtualised infrastructures. Another approach, [27], proposes to split the hypervisor into smaller components. Preallocation of resources to prevent attacks, was proposed in [42], that devise a mechanism that eliminates vulnerabilities pertaining to communication between the virtual machines and the hypervisor by preallocating memory and processor cores, in order to alleviate the need in hypervisor to allocate resources dynamically.

There is also an effort to design anomaly detection techniques that would identify malicious instances or malicious behaviour, as well as design of systems to identify vulnerabilities in legitimate virtual machines and systems. For instance, [18] devise a security framework for cloud computing that, among others, enables detection of abnormal or anomalous behaviour and detection of vulnerabilities in programs (and patches thereof). CloudER, [7], present an architecture that automatically detects and patches software vulnerabilities in cloud applications at runtime.

A secure storage design is explored in a number of works, e.g., [6, 16, 19, 32] and different architectures and cryptographic techniques are considered.

However, relatively little attention is being paid to network-based attacks, specifically, network-based side channels, which may enable attackers to deanonymise IP addresses of public services. This phase often serves as a stepping stone towards more devastating and targeted attacks, e.g., denial of service, or information theft via shared memory side-channels. The attacker can also apply IP deanonymisation techniques to achieve coresident placement of a malicious instance, on the same physical host as a victim. Indeed, very few proposed defenses address such a threat, and in this work we set forth to investigate, what such techniques can achieve

and whether address deanonymisation and coresidence detection are still feasible attack vectors. We show different techniques that are effective on popular cloud providers' networks, and that cannot be prevented with the defense mechanisms proposed in prior art. In particular our attacks apply to cloud providers that patched their networks following attacks of [31].

3. IP ADDRESS DEANONYMISATION

In this section we present techniques allowing to deanonymise the public services via discovery of their internal IP addresses. IP address deanonymisation is applicable only to cloud platforms which allow communication between internal instances, e.g., Amazon AWS EC2, Rackspace Cloud.

We first present a generic technique, which allows discovery of the victim's IP address regardless of the services that it runs. Subsequently, we show techniques, tailored at a specific service, which allow for a more efficient deanonymisation of internal services.

In contrast to prior art, our approaches do not require acquisition of multiple hosts, [3, 31].

3.1 Address Discovery via Interrupts

We introduce a technique, interrupt-overloading side-channel, which we utilise for discovery of internal IP address of some public host. We first provide a background required to describe our technique and then show how to apply it.

3.1.1 Interrupt Driven Scheduling

The kernels in operating systems (OSes), e.g., Unix and Microsoft platforms, use hardware interrupts for event notification purposes in communication with input/output hardware components. Network interface cards (NICs) generate interrupts to notify the kernel of arrival or transmission of new packets. When a NIC receives a packet it triggers a hardware interrupt. As a result, the CPU suspends its current activity and executes an interrupt handling routine, the kernel processes the packet and invokes the relevant protocol, e.g., UDP, for further packet handling. The kernel processing of packets continues as long as there are packets in the system buffer. The kernel processing can be preempted by interrupt handling procedure, as a result of new packets' arrival.

Hardware interrupts can impose a significant CPU overhead. This is due to the fact that hardware interrupt is associated with context switching of saving and restoring processor state, see discussion in [20], and a typical interrupt handling latency, due to packet arrival, in Linux is in order of 50 μsec, see [34].

After the notification of a new packet arrival the kernel processes the packet, and then invokes TCP/IP protocol processing. Since the operating systems assign higher priority to hardware interrupts, than to other tasks, arrival of a new packet disrupts protocol processing. Under high traffic load, tasks with lower priorities may reach starvation, since the system is busy spending all of its time processing interrupts for arriving packets, see [21, 30].

High packets' arrival rate also has a negative impact on packet transmission, since inbound packets' handling routine has higher priority over packets' transmission routine.

The problem with the overhead of hardware interrupts is further exacerbated in virtualised environments, where the number of interrupts is multiplied, due to host-vs-guest context switches, [17].

Inbound packet handling slightly differs between different operating systems. It was shown, e.g., [34], that Linux significantly outperforms Windows OS. Linux kernel networking supports a Linux New API (NAPI) packet handling mechanism, [36], in its processing and scheduling of incoming packets, which reduces the frequency of generated interrupts. NAPI was designed to handle high traffic rates of fast Ethernet. Therefore, in our experimental evaluation we focus on Linux kernel 3.2.0 OS, and our results also apply (with an even more significant impact) against Windows OS platforms.

We show how to exploit this property to create a side channel, allowing to deanonymise the internal IP address assigned to some public service.

3.1.2 Interrupt-Overloading Side Channels

The idea is to apply interrupt-overloading side-channel to introduce *patterns* into the communication flow. By inspecting the traffic the attacker can learn information whether the burst was sent to a correct address.

Recently, [14] applied socket overloading, to cause packet loss at the receiving end host, for port derandomisation. The attack of [14] required sending packets to different destination ports (sampling each port). When a correct port is sampled, packets' loss occurs. The attacker uses this timing channel (of response latency) as indication of a correct guess. In contrast, in our work, we do not attempt to guess the port at the receiver, and our goal is to discover the internal IP address of the victim host; thus, we can send our packets' burst to any port of the receiver, which adds a delay to all connections that the victim host maintains.

The scenario that we consider is illustrated in Figure 1. An attacker, that wishes to discover an internal IP address of some victim service, sets up two types of instances: *malicious client* and *probers*. The client connects to a victim service via a public IP address (alternately a public domain) of the victim, e.g., downloads a file or a web page; the client can reside either on the cloud platform, or can be set up on a host outside of a cloud network. The attacker then instructs the probers to send bursts to different destinations in the address block range, and inspects the communication from the victim server, looking for patterns.

The idea is that upon arrival at the victim host, the burst of packets impacts the transmission and processing times of the victim, introducing delays into the sent packets. If delays, corresponding to the timing and volume of the burst(s), are observable in the traffic, then the correlation between the external and internal addresses of the victim is found. Otherwise, the prober proceeds to the next IP address. In order to increase the effectiveness of the burst, the attacker should increase the concentration of the burst volume. Notice that the host, sending the bursts, (inevitably) adds delays to transmitted packets, e.g., among other factors also due to hardware interrupts, and thus the burst is slightly dispersed. We show how to overcome this issue by distributing a burst among a number of probers.

In our experimental evaluation we set up the attacker, as well as the probers, as instances on a cloud; see Figure 1. We also set up a number of victim hosts, that acted as public servers and served files upon requests. We report on our results of our evaluation in Section 3.1.3.

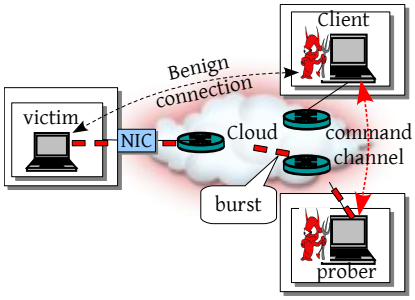

Figure 1: Setting and attacker model. The attacker controls a prober instance and a malicious client, and uses them to deanonymise the IP address of a victim service.

3.1.3 Experimental Evaluation

Our experimental setup consists of a *victim*, a *malicious client* and *probers*, see setting in Figure 1. In this section we report on experiments that were performed on Amazon AWS EC2 cloud network. All our instances run on Ubuntu server 12.04.2 LTS (32 bit) linux kernel OS.

The victim is a legitimate public service, e.g., web, file or email, running on a host on a cloud network. We set up an `http` web server that serves files, of variable sizes, to clients upon request. The victim has a public hostname and address, and also internal hostname and internal address (which are not known to the attacker).

The probers are tenants on the cloud, while the client can be on a host outside of the cloud network, or can also reside on the cloud; in our evaluation, reported in Figure 3 and Figure 4, we placed the client along with the probers on the cloud. Notice that, when the attacker is on the cloud, since no outbound/inbound (to/from the cloud) communication is generated, no costs apply on the attacker.

The client establishes a connection to the victim host via its public hostname, whose internal address it wishes to find, and requests a file, in our evaluation results, reported below, we used 100KB files. The client controls the probers, and invokes them to send the bursts. The bursts are UDP datagrams of variable sizes. The synchronisation between the client and the probers was implemented as follows: the probers listen on an agreed port and once they receive a command from the client with a destination port and IP address, burst size and burst frequency, they start probing the target. The client then inspects the flow from the victim host, searching for patterns that correspond to the burst size and timing, generated by the probers.

We wrote a python file transfer script which the client used to contact the server (over TCP) and to download files. The probers were configured with a user-space C code, transmitting a burst of raw UDP packets of a specified size, in specified intervals.

Two configurations were tested for victim server, m1.small and m1.large, see Table 2; quantities in ECU[1]; our evaluation results, reported in Figure 2 and 4, pertain to configuration using m1.large instance for victim service. The probers and the client all run on micro type (free) instances. In our experimental evaluation below, we set up all our instances on the same cloud availability zone[2].

Hardware	m1.micro	m1.small	m1.large
CPU	< 2 ECU	1vCPU w/1 ECU	2vCPU w/4 ECU
RAM	0.615GB	1.7GB	7.5GB

Table 2: Configuration of instances' types used in our evaluation on Amazon AWS EC2.

We evaluate the impact of (a burst of) inbound packets, on the latency inflicted on outbound packets, as a result of hardware interrupts, and the ability of a victim server to transmit packets during reception of a burst. Our goal is to craft the smallest possible packets' burst that would suffice to introduce noticeable (timing) patterns into the communication. Our technique exploits the fact that inbound packets have higher priority over outbound transmissions, and therefore the burst is expected to introduce delays into the transmitted flow.

The main factors impacting the efficacy of the burst are packets' sizes, number of packets and packets' concentration, i.e., the less dispersed the burst is, the higher the interrupt rate at the receiver, and as a result, also the perceived latency at the attacker's client.

Intuitively, it may seem that large packets should be optimal in their ability to overwhelm the receiving system. However, as we show, in Figure 2, small packets of 100 bytes, among the sizes that we tested, turn out to be best, since they introduce a maximal number of interrupts; notice that smaller packets than 100 bytes could be used, but the difference in latency at the receiver is negligible. In our evaluation of the additional latency inflicted on the receiving host, in Figure 2, we used a single burst of 2000 packets per second.

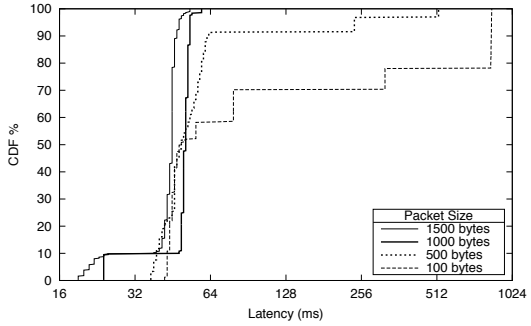

Figure 2: Evaluation of optimal packets' sizes with respect to the additional latency due to inflicted interrupts.

Burst concentration, i.e., the number of arriving packets over some time interval, is another factor which impacts the effectiveness of the attack. An impact of a dispersed burst is lower than that of a condensed burst. In particular, if the inter-packet delay, in arriving traffic, is large enough, to allow the receiving host to process each packet before the arrival of a new one, then, even a large burst may not introduce significant delays into the traffic flow. In contrast, a

[1]ECU is one EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

[2]Notice that since all our communication is within the cloud boundaries, no charges apply on the attacker.

sufficiently concentrated burst, consisting of just a few hundreds of packets, results in a noticeable impact. Notice that each physical host, i.e., the prober that sends the burst, as well as the routers en-route, introduce their own interrupts when forwarding the packets. To address this problem and enhance the effectiveness of the burst, we propose to distribute the burst among a number of hosts, such that each sends a fraction of the volume to the recipient in parallel. A related factor is burst volume, i.e., number of packets comprising a burst: the more packets in a burst, the higher the impact is. We test the burst effectiveness using two burst sizes, of 1000 packets and 2000 packets, and with a single prober instance, two prober instances and three prober instances; in our evaluation we used uniform packets' size of 100 bytes. We report on the results of our evaluation in Figure 3.

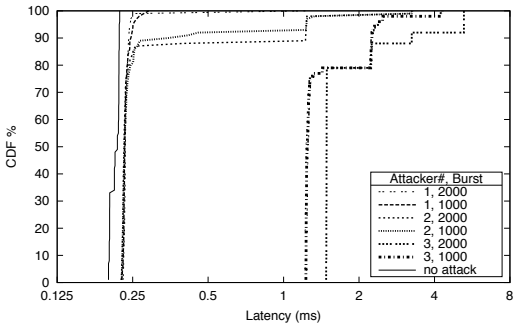

Figure 3: Evaluation of latency with variable burst sizes, of 1000 and 2000 packets, each packet of size 100 bytes, and using a single prober, two probers, and three probers.

Finally, we tested the success rate, of detecting the correct (victim) instance, among a pool of instances, using our techniques of sending bursts to potential victims and measuring the timing patterns in communication to the target victim. In our evaluation, reported in Figure 4, we use uniform packets' size of 100 bytes, and generate a single burst. We tested the effectiveness, of correct identification of the victim, using variable burst size (which we express as a rate per second in Figure 4) and different probers' number, between 1 and 3. As can be seen, with a relatively small burst size, consisting of up to a 1000 packets per prober (using three probers), the attacker can reach above 60% success probability of correctly identifying the victim.

In our experimental evaluation we used the services that we set up to emulate the victim services. Discovery of real public services may be more challenging, in particular, since multiple clients connect to public services and thus will potentially introduce more noise into our measurements. To cope with the noise the attacker may be required to generate larger and more frequent (e.g., more than one) bursts, and can also utilise more probing hosts in an attack to concentrate the burst. Furthermore, the attacker may need to apply different encoding schemes, to filter out noise and identify the timing patterns that correspond to its bursts.

3.2 Email Server Discovery

We present two techniques for discovery of internal address of email servers. One is via a scan of internal address block, while the other is via external. Notice that to pass anti-spam validation that legitimate email servers perform, the attacker should set up the required SPF and PTR records (via the cloud provider). These records also allow discovery of resolvers, which the email servers use, since they will trigger DNS requests to attacker's domain.

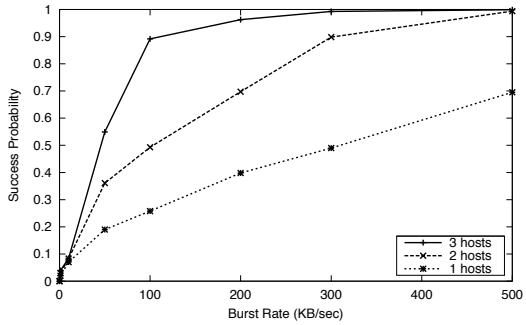

Figure 4: Success rate evaluation of host deanonymisation.

3.2.1 Internal Address Block Scan

The attacker sets up email and name server instances, and sends email to every address in the network block, and monitors responses. Hosts that respond are recorded.

Notice however, that Amazon requires that all tenants apply, via a special procedure, to obtain a permission to make outgoing connections on port 25. Packets to port 25 are monitored, and after a threshold (couple of thousands) of packets, Amazon operators issue an email notifying the account owner of this event.

To circumvent this restriction the attacker should distribute the scan over multiple hosts.

We next present a technique that allows for an effective discovery of email and DNS resolver services, via external network scan.

3.2.2 Server-Bounce (External) Scan

The attacker sets up two email servers: one *external*, i.e., outside of a cloud network, and another *internal*, i.e., set up as instance on a cloud network; we used postfix for both email servers in our experimental evaluation. The attacker scans the external IP address block of the cloud, and sends email to each address that responds to packets sent to port TCP:25.

The emails are sent to a non existing recipient, via the `receipt to` header (instantiated with a random email address), and with a 'spoofed' `mail from` address, indicating the address (or equivalently the internal hostname) of the internal email service that the attacker apriori set up in the cloud. These emails result in delivery sender notification ('bounce') messages (since the destination is a non-existent email address), that are sent by the (victim) mail servers to our internal host. The host runs a packet capture tool (*tcpdump*) and records the identity[3] and addresses of the mail servers that send the notifications.

Notice that email servers that support anti-spam and anti-phishing mechanisms also send requests, via their resolvers,

[3]Email servers identify themselves via the HELO header of the SMTP protocol.

for SPF records of the domain indicated in the `mail from` field; the attacker can set up these records in the zone file of the DNS server authoritative for its domain. The SPF/PTR queries, also expose the address of the caching DNS resolver which the email servers use.

We applied this attack to two address blocks of US East (Northern Virginia), 54.224.0.0/16 and 54.213.0.0/16. Out of 65535 addresses on 54.224.0.0/16, we found 1336 responsive hosts, of which: 1121 sent a RST in response to our connection request, while the remaining 215 responded with a SYN/ACK, and accepted our email. In 54.213.0.0/16 address block, 544 addresses responded with a RST and 109 accepted the email connection.

3.3 Cloud Scan and Evading Detection

While the cloud providers explicitly prohibit scans of their networks, they are still possible. Furthermore, a diligent attacker can scan the entire network staying well under the detection radar. Amazon imposes quotas on scans rate, and on a number of scanned hosts. But these limits can be overcome by distributing the scan among a number of adversarial hosts. Furthermore, to evade detection, instead of running traditional scans of addresses, where the probes are sent to sequentially incrementing addresses, the attacker may use a pseudorandom scan, whereby the internal addresses are selected pseudorandomly; e.g., if a cloud service runs an intrusion detection system, and inspects the traffic volumes[4].

4. HOP-COUNT MEASURING

In this section we present techniques allowing to reconstruct paths to some destination, and even reconstruct the entire map of the cloud. We first describe the technique and then our applications thereof.

After discovering the internal IP address of the victim service[5], e.g., by applying techniques, presented in previous section, the attacker can test proximity to it, by performing a *hop-count measuring* test, described in Figure 5 and illustrated in Figure 6. The idea is to reconstruct the path, between the attacker and the victim, by performing a TTL scan. The idea is to use the *time to live* field in the IP header. The attacker creates a TCP SYN packet, starting with TTL=1, subsequently incrementing the TTL in each transmitted packet, and sends it to the destination. Notice that the receiver cannot use the naive TTL measurement, whereby the receiver learns the route by inspecting the TTL in packets which it receives, since routers may change the TTL value with some default value. Our TTL scan technique is not sensitive to such modification by intermediate routers. Our technique assumes a standard behaviour of the routers, i.e., each router, en-route on the path between the attacker and the destination, decrements the TTL and forwards the packet. If the TTL reaches 0, the router discards the packet, and typically returns an ICMP error. However, following the attack of [31] cloud Amazon EC2 block ICMP messages to prevent attackers from scanning of their network. Thus routers silently discard packets with TTL = 0.

[4]During our experimental evaluation on Rackspace the operators contacted us inquiring on our activity.

[5]Notice that the attacker can also learn information about cloud topology using public IP addresses. However, this information only allows to learn the path of packets that traverse the NAT.

We therefore use timers, and at each timeout event we increase the TTL and retransmit the packet. When the packet reaches the destination, a SYN/ACK packet (or a RST in case the port is closed) is returned. This procedure allows us to reconstruct the path to the victim destination; see pseudocode in Figure 5.

```
# input: ip

for i in range(255):
    # construct TCP SYN packet to dest port 22
    request = IP(dst=ip, ttl=i)/TCP(dport=22,flags="S")
    # send request, receive response
    response = sr1(request, retry=0, timeout=1):
    if response:
     # if response is a TCP packet
        if response.proto == 6:
            print "Victim is "+str(i)+" hops away";
            break
```

Figure 5: Hop-count measuring.

A TTL = 0 to some destination, is an indication of a potential co-residence, but not a guarantee. This depends on whether the hypervisor reports itself as a hop on the path from the instances on a physical host; for instance, this holds in Amazon EC2, see [31]. To conclude definite co-residence, the TTL hop-count measuring technique should be used in tandem with interrupt-overloading side-channel. Specifically, when TTL = 0 to some destination is reported, the attacker should confirm co-residence by launching bursts at the destination and looking for patterns in its connection to the victim, as described in Section 5.

We tested hop counting on a number of providers, and report on findings in Table 1. We found that Amazon AWS EC2, Rackspace Cloud, and Google Compute Engine are vulnerable to hop counting. While on EC2 and Rackspace, it is possible to perform hop counting to every host, on Google Engine the techique is restricted to hosts belonging to same project (namely, the attacker can only use this technique to learn the topology of the cloud using its own instances). Microsoft Azure block internal communication, and therefore hop counting on the internal network between internal hosts is not possible on their network.

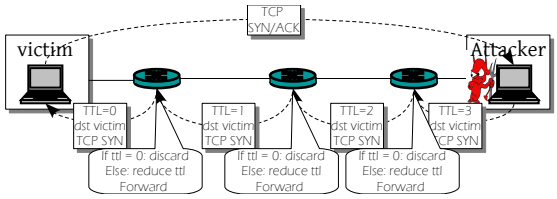

Figure 6: Hop-count measuring via TTL-based scan, to reconstruct the path between the attacker and the victim.

5. CO-RESIDENCE VERIFICATION

In this section we show how to apply our interrupt-overloading side channel, and TTL probing, for coresidency verification. The attacker places the probed instance on different hosts, migrating it to different hosts at each iteration of the attack, attempting to (probabilistically) achieve co-residence.

When sampling a specific host, the attacker performs the following procedure (see Figure 7): the attacker measures distance from the probed instance to the victim, by running the TTL scan. When the TTL, in the scan, between the probed host and the victim is 0, the malicious client establishes a connection to the victim, via the public hostname of the victim, and instructs the probers to send bursts to the (internal IP address of the) probed host. If the probed host resides on the same physical host as the victim instance, the client will observe patterns, in its communication to the victim, that correspond to the bursts sent by the probers.

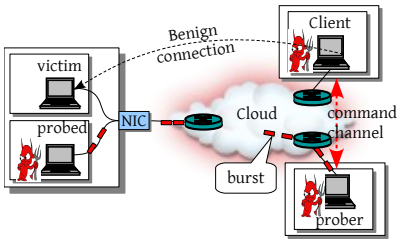

Figure 7: Co-residence verification via interrupt-overloading side channel.

A simple extension of our co-residence testing technique, can be applied to check if two (or more) victims reside on the same physical host: run the procedure above against two (or more) victim services. There are a number of motivations for detecting if two victim services are placed on the same physical host. For instance, if one victim is known to have some vulnerability in its software, e.g., operating system or web application, the attacker may use this information to attack the neighbouring instances.

6. DEFENSES

Clearly preventing co-residence would solve most of the attacks that we presented (it would not solve only the hop-count measuring), but co-residence is one of the core properties of cloud computing paradigm that makes it appealing for cloud providers and approachable, due to its low costs, for customers.

We next propose a number of defenses, some are specific to the cloud while others are related to design of packets' processing and scheduling by network adapters. No single proposed defense prevents *all* of our attacks, but they can make it significantly harder to launch the attacks, thus making it not sufficiently attractive for the attackers.

Cloud-Based Defenses.

We discuss a number of defenses which the cloud provider can employ to prevent some of our attacks.

► A straight-forward defense is to block communication between the internal hosts on the cloud. This is approach is supported by Google Compute Engine and by Microsoft Azure. Blocking internal communication would prevent most of our attacks, including address deanonymisation and hop-count measuring (notice that this would still allow learning some information about the cloud topology, by running the TTL scans to public address of the hosts). The downside of this is that it would also have a negative impact on the connectivity of customers' instances. It would also require appropriate hardware and software on the cloud network, which the cloud provider may not be willing to invest in. Furthermore, blocking internal communication does not prevent co-residence attacks, from Section 5.

► A defense to prevent interrupt-overloading tests can be to use separate network adapters for each instance. This would ensure that traffic, from different instances on the same host, is not multiplexed. However, this countermeasure would also require appropriate hardware support from the cloud provider.

► Cloud provider can use firewall based techniques to rate limit traffic to instances located on its network, which would also prevent, or at least significantly limit techniques that use interrupt-overloading tests.

Design of Network Adapters.

The research which studies efficient design and architectures of packet handling procedures by kernels, focuses on benign network load, e.g., [33, 35]. To enhance robustness to network attacks, systems should be designed to withstand maliciously devised flows, such as those exploited in our interrupt-overloading tests.

7. CONCLUSIONS

In this work we study instances isolation on cloud platforms. We show that given a public address of some victim service, attackers can often find victim's corresponding private address and find its location within the cloud. Our attacks are performed in three steps: (1) we deanonymise the private IP address of some public victim service, then (2) measure distance to the victim from attacker's instance, and finally (3) attempt to achieve co-residence. We propose new techniques to accomplish these three steps. Our techniques are based on timing side channels, and traffic analysis. Our techniques can be extended to perform complete cloud topology discovery and mapping.

Our interrupt-overloading side channel is a generic (protocol independent) technique, which allows to determine the internal IP address of a public service, and can also be used for co-residence verification. Interrupt-overloading technique can be applied for other attacks, and in particular, can be used to launch low-rate degradation/denial of service (D/DoS) attacks on cloud services; low-rate attacks against TCP flows were proposed in prior art, but were launched using different techniques, e.g., see [13, 22]. We leave it as a future work to investigate applications of our interrupt-overloading technique for low-rate attacks against cloud platforms and services.

We also showed how to use protocol specific, e.g., DNS and Email, properties, to discover internal IP addresses of these services.

We suggested a design of hop-count measuring technique, that can be used to reconstruct a path, taken by packets, between two instances, e.g., attacker's and some victim's. This technique can be used to also check for co-residence and to reconstruct the entire topology of the cloud network.

As we show in our work, address deanonymisation attacks and attacks exploiting co-residence, via network channel, are still possible, and efforts should be made to further investigate techniques preventing such vulnerabilities. The main message of our work is that allowing communication between internal hosts is may be exploited for attacks. However, as we also point out, completely blocking internal communication may not be a practical solution. In particular, internal

communication is required for many applications, and allows to reduce communication costs on the cloud, as well as latency for clients. Therefore, preventing our attacks, without adverse impact on cloud applications and tenants, requires further investigation.

Our techniques can also be used for benign purposes, to verify service provided by the cloud to its clients, specifically, topology and placement verification. One of the important questions in cloud computing paradigm, is to enable clients, via a systematic approach, to validate resources and services that the cloud provides; see [37] for a discussion of the challenges in realising such requirements and guarantees. This includes ensuring that: (1) the applications of the customers indeed consumed the physical resources, which the cloud provider charges them for, or that (2) the platform that the cloud provider supplied to the customer reflects the one that cloud provider guaranteed to supply.

Our techniques can be applied to validate the placement guarantees of the cloud provider to its customers. For instance, if the customer wishes ensure that all its instances are placed on a single physical host, or alternately, on separate hosts, the customer can apply the interrupt-overloading side channel in tandem with the hop-count measuring test.

Acknowledgements

We thank the anonymous referees for their feedback, which helped us improve the outline of this manuscript. This research was supported by the Ministry of Science, Technology and Space, Israel, and by COMET K1, FFG - Austrian Research Promotion Agency.

8. REFERENCES

[1] O. Acıiçmez, B. B. Brumley, and P. Grabher. New results on instruction cache attacks. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 110–124. Springer, 2010.

[2] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea. Chatty tenants and the cloud network sharing problem. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, pages 171–184. USENIX Association, 2013.

[3] A. Bates, B. Mood, J. Pletcher, H. Pruse, M. Valafar, and K. Butler. Detecting co-residency with active traffic analysis techniques. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, pages 1–12. ACM, 2012.

[4] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: Design and implementation of nested virtualization. In *OSDI*, volume 10, pages 423–436, 2010.

[5] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest. How to tell if your cloud files are vulnerable to drive crashes. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 501–514. ACM, 2011.

[6] S. Bugiel, S. Nürnberger, A.-R. Sadeghi, and T. Schneider. Twin clouds: An architecture for secure cloud computing. In *Workshop on Cryptography and Security in Clouds (WCSC 2011)*, 2011.

[7] P. Chen, D. Xu, and B. Mao. Clouder: a framework for automatic software vulnerability location and patching in the cloud. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, pages 50–50. ACM, 2012.

[8] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90. ACM, 2009.

[9] A. Coates, A. O. Hero III, R. Nowak, and B. Yu. Internet tomography. *Signal Processing Magazine, IEEE*, 19(3):47–65, 2002.

[10] S. De Capitani di Vimercati, S. Foresti, and P. Samarati. Managing and accessing data in the cloud: Privacy risks and approaches. In *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, pages 1–9. IEEE, 2012.

[11] P. Gasti, G. Ateniese, and M. Blanton. Deniable cloud storage: sharing files via public-key deniability. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 31–42. ACM, 2010.

[12] D. Gullasch, E. Bangerter, and S. Krenn. Cache games–bringing access-based cache attacks on aes to practice. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 490–505. IEEE, 2011.

[13] A. Herzberg and H. Shulman. Stealth DoS attacks on secure channels. In *Proc. Symp. on Network and Distributed Systems Security (NDSS '10)*, San Diego, CA, Feb. 2010. Internet Society.

[14] A. Herzberg and H. Shulman. Socket Overloading for Fun and Cache-Poisoning. In *Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC)*. ACM, 2013.

[15] A. Juels and A. Oprea. New approaches to security and availability for cloud data. *Communications of the ACM*, 56(2):64–73, 2013.

[16] S. Kamara and K. Lauter. Cryptographic cloud storage. In *Financial Cryptography and Data Security*, pages 136–149. Springer, 2010.

[17] P. A. Karger and D. R. Safford. I/O for Virtual Machine Monitors: Security and Performance Issues. *IEEE Security & Privacy*, 6(5):16–23, 2008.

[18] A. D. Keromytis, R. Geambasu, S. Sethumadhavan, S. J. Stolfo, J. Yang, A. Benameur, M. Dacier, M. Elder, D. Kienzle, and A. Stavrou. The meerkats cloud security architecture. In *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, pages 446–450. IEEE, 2012.

[19] B. H. Kim, W. Huang, and D. Lie. Unity: secure and durable personal cloud storage. In *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*, pages 31–36. ACM, 2012.

[20] S. R. Kleiman. Apparatus and method for interrupt handling in a multi-threaded operating system kernel, May 7 1996. US Patent 5,515,538.

[21] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[22] A. Kuzmanovic and E. W. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks: the Shrew vs. the Mice and Elephants. In *SIGCOMM*, pages 75–86, New York, NY, USA, 2003. ACM.

[23] S. Larsen, P. Sarangam, R. Huggahalli, and S. Kulkarni. Architectural breakdown of end-to-end latency in a tcp/ip network. *International Journal of Parallel Programming*, 37(6):556–571, 2009.

[24] R. B. Lee. Hardware-enhanced access control for cloud computing. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 1–2. ACM, 2012.

[25] J. Liu. Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe nics with sr-iov support. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.

[26] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: the case of aes. In *Topics in Cryptology–CT-RSA 2006*, pages 1–20. Springer, 2006.

[27] W. Pan, Y. Zhang, M. Yu, and J. Jing. Improving virtualization security by splitting hypervisor into smaller components. In *Data and Applications Security and Privacy XXVI*, pages 298–313. Springer, 2012.

[28] L. Popa, M. Yu, S. Y. Ko, S. Ratnasamy, and I. Stoica. Cloudpolice: taking access control out of the network. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 7. ACM, 2010.

[29] H. Raj, R. Nathuji, A. Singh, and P. England. Resource management for isolation enhanced cloud services. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 77–84. ACM, 2009.

[30] K. Ramakrishnan. Performance considerations in designing network interfaces. *Selected Areas in Communications, IEEE Journal on*, 11(2):203–219, 1993.

[31] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[32] A.-R. Sadeghi, T. Schneider, and M. Winandy. Token-based cloud computing. In *Trust and Trustworthy Computing*, pages 417–429. Springer, 2010.

[33] K. Salah. To coalesce or not to coalesce. *AEU-International Journal of Electronics and Communications*, 61(4):215–225, 2007.

[34] K. Salah, K. El-Badawi, and F. Haidari. Performance analysis and comparison of interrupt-handling schemes in gigabit networks. *Computer Communications*, 30(17):3425–3441, 2007.

[35] K. Salah and A. Qahtan. Boosting throughput of snort nids under linux. In *Innovations in Information Technology, 2008. IIT 2008. International Conference on*, pages 643–647. IEEE, 2008.

[36] J. H. Salim, R. Olsson, and A. Kuznetsov. Beyond softnet. In *Proceedings of the 5th annual Linux Showcase & Conference*, volume 5, pages 18–18, 2001.

[37] V. Sekar and P. Maniatis. Verifiable resource accounting for cloud computing services. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 21–26. ACM, 2011.

[38] J. Shi, X. Song, H. Chen, and B. Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, pages 194–199. IEEE, 2011.

[39] N. C. A. System. Vulnerability Summary for CVE-2007-4993. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-4993`, September 2007.

[40] N. C. A. System. Vulnerability Summary for CVE-2007-5497. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-5497`, December 2007.

[41] N. C. A. System. Vulnerability Summary for CVE-2010-2240. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-2240`, March 2010.

[42] J. Szefer, E. Keller, R. B. Lee, and J. Rexford. Eliminating the hypervisor attack surface for a more secure cloud. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 401–412. ACM, 2011.

[43] E. Tromer, D. A. Osvik, and A. Shamir. Efficient cache attacks on aes, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[44] D. Tsafrir, Y. Etsion, D. G. Feitelson, and S. Kirkpatrick. System noise, os clock ticks, and fine-grained parallel applications. In *Proceedings of the 19th annual international conference on Supercomputing*, pages 303–312. ACM, 2005.

[45] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, and W. Zwaenepoel. Concurrent direct network access for virtual machine monitors. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 306–317. IEEE, 2007.

[46] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 29–40. ACM, 2011.

[47] M. Zec, M. Mikuc, and M. Zagar. Estimating the impact of interrupt coalescing delays on steady state tcp throughput. In *Proceedings of the 10th SoftCOM*, volume 2002. Citeseer, 2002.

[48] F. Zhang, J. Chen, H. Chen, and B. Zang. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 203–216. ACM, 2011.

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

[50] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 305–316. ACM, 2012.