A Covert Channel Using Event Channel State on Xen Hypervisor*

Qingni Shen[1,2,**], Mian Wan[1,2], Zhuangzhuang Zhang[1,2], Zhi Zhang[1,2], Sihan Qing[1,2,3], and Zhonghai Wu[1,2]

[1] School of Software and Microelectronics, Peking University, Beijing 102600, China
[2] MoE Key Lab of Network and Software Assurance, Peking University, Beijing 100871, China
[3] Institue of Software, Chinese Academy of Sciences, Beijing 100190, China
qingnishen@ss.pku.edu.cn

Abstract. Covert channel between virtual machines is one of serious threats to cloud computing, since it will break the isolation of guest OSs. Even if a lot of work has been done to resist covert channels, new covert channels still emerge in various manners. In this paper, we introduce event channel mechanism in detail. Then we develop a covert channel called CCECS(Covert Channel using Event Channel State) and implement it on Xen hypervisor. Finally we quantitatively evaluate CCECS and discuss the possible mitigation methods. Results show that it can achieve larger bit rate than most existing covert channels.

Keywords: Covert Channel, Virtualization, Event Channel.

1 Introduction

Thanks to the resource sharing, both cloud computing and virtualization give the users an illusion of "occupying resources independently".Nevertheless, resource sharing is a double-edged sword. It makes virtualization have a inherent feature - multitenancy, placing multiple virtual machines(VM) of distinct customers upon the same physical hardware. Therefore, the privacy and security of customers' information may be compromised[1]. Isolation, one important function of virtualization, is designed to eliminate this kind of threat. Due to isolation, a VM cannot access the resources of others. Even if a VM is exploited or manipulated by an attacker, it will not affect the other VMs within the same host.

However, this isolation has been proved not strong enough(e.g. covert and side channels)[2]. Covert Channel[3] is a mechanism that is not intended to transfer sensitive message, violating security policies specified by system . Covert channel is actively sending data, while side channel[4] is passively observing information. At present, several covert channels have been discovered, but the mediums they used are not the same.

* This work is supported by National Natural Science Foundation of China (No. 61232005, 61073156, 61170282).

** Corresponding author.

S. Qing et al. (Eds.): ICICS 2013, LNCS 8233, pp. 125–134, 2013.

Ristenpart et al. [5] first exposed cloud computing to covert channel attacks. They had implemented an L2 cache covert channel in Amazon EC2. The bit rate of it was just 0.2 bps(bit-per-second), a mere fraction above the minimum standard of 0.1 bps [3]. Xu et al. [6] refined the communication model of L2 Cache Covert Channel and the channel arrived at considerably higher bandwidth(3.20 bps) in EC2. On the basis of previous work, Wu et al. [7] redesigned the data transmission scheme and exploited the memory bus as medium. At last, their new covert channel achieved a transmission rate of 107 plus or minius 39 bps in EC2, and 700bps in house. Except for covert channels using CPU cache, there were some other covert channels using other resource(such as CPU load [8], core alternation [9], sharing memory [10],[11], *mfn2pfn* table [12], network [13]) as medium. None of the covert channel above had considered event channel as a vehicle. In fact, the channel via event channel states is feasible and even more dangerous than most of the covert channels in virtualized environment.

In this paper, we analyze the event channel mechanism in Xen and develop a reliable covert channel to transfer information between two virtual machines. We verify the feasibility and effectiveness of the covert channel using event channel state(CCECS) for data leakage through a set of experiments. The bandwidth can achieve about 13Kbps. Then we discuss the method to mitigate the threat.

2 Background

Xen is an open-source VMM(Virtual Machine Monitor), also called *hypervisor*[14], that widely used in the cloud computing industry. For example, Amazon's EC2 (Elastic Compute Cloud)[15] adopts Xen as its virtualization technology. It is a software layer that sits between the host hardware and the VMs(Virtual Machine). It partitions hardware resources and offers them to the VM as virtual resource. Aside from that, it also controls access to I/O devices.

In Xen, VMs is also called *domains*. When Xen boots, it first create a special domain called `Dom0`, which has elevated privileges. It helps Xen to create, destroy, configure, migrate, save and restore VMs, and controls VM's access to I/O devices.

`DomU`, VMs created by Dom0, is more restrictive. Since Xen is a kind of paravirtualization, DomU cannot perform any privileged operations. For instance, instead of accessing the devices directly, DomU has to transmit device request through the front-end driver to the back-end driver in Dom0. Then Dom0 access the real devices as an agent. Unlike Dom0, users can have an arbitrary number of domU guests.

As a hypervisor, Xen has to track and control the running situation of each domain. And in some cases such as migration, communication between domains is necessary. Xen hypervisor offers 3 communication mechanisms. They are hypercall, sharing memory and event channel. Since our channel uses event channel, thus we will just introduce event channel mechanism in next section.

2.1 Event Channel Mechanism in Xen

Event channel is an asynchronous notifications mechanism within Xen. Event channels cooperate with ring buffers in shared memory pages to accomplish the message transmission between the front and back ends of split device drivers. In Xen, events are notifications that are delivered from hypervisor to domains, or between domains. Each event's content is one bit of information that indicates its occurrence. Events can divide into 4 categories: *physical IRQ*, *virtual IRQ(VIRQ)*, *interprocessor interrupts(IPIs)*, *interdomain events*. Physical IRQ is used for the communicating with hardware. Guest in Dom0 or domain that is authorized to access device will set up physical IRQs to event channel mapping for diverse devices. Because of its paravirtualization, this is done by a hypercall instead of BIOS calls. Virtual IRQs resemble physical IRQs, but are related to virtual devices. Interdomain events involve two domains and include two stages. First, Domain A allocates a new event channel(port) as an unbound channel and grants Domain B with the permission to bind to it. Then, Domain B allocates a new channel and binds it to the remote domain(Domain A)'s port. When the connection is built, either domain can send an event to the local port. Moreover, the connection is duplex. The fourth kind of events, the interprocessor interrupts, are equivalent to *intradomain events*. It can be viewed as a special case of interdomain events where both local and remote domain are the same one. It can be used to communicate between vCPUs in the same guest.

In Xen, every domain can own its own event channels. when a event channel is allocated, a port is used to identify the channel. Besides port, a event channel has some other attributes, mainly including the state and other variables related to certain state. All the possible states of event channels is listed in Table 1.

Table 1. States and Statuses of Event Channels

State Constant	Status Constant	Comments
ECS_FREE	EVTCHNSTAT_closed	Not in use now
ECS_RESERVED	EVTCHNSTAT_closed	Not in use now
ECS_UNBOUND	EVTCHNSTAT_unbound	Waiting inter-dom connect
ECS_INTERDOMAIN	EVTCHNSTAT_interdomain	Connected to a remote Dom
ECS_PIRQ	EVTCHNSTAT_pirq	Bound to a physical IRQ
ECS_VIRQ	EVTCHNSTAT_virq	Bound to a virtual IRQ
ECS_IPI	EVTCHNSTAT_ipi	Bound for IPC

Like other operations of event channels, building interdomain event channels is done by calling the hypercall `HYPERVISOR_event_channel_op`. When DomA wants to communicate with DomB, DomB initially acquires an unbound event channel by calling the operation "`EVTCHNOP_alloc_unbound`" in the hypercall. During the allocating, its state is set as `ECS_UNBOUND`. And its port is stored in XenStore where DomA can consult. Besides, the channel creator specifies which domain is authorized to bind to it. Next, DomA will also obtain an allocated

channel. After that, DomA accesses XenStore and retrieves the port number, then binds itself by calling the operation "`EVTCHNOP_bind_interdomain`" in the hypercall. Then the two event channels will constitute a duplex connection. If Dom A wants to send events to Dom B, it will just send them to local event channel. Both channels' state changes to `ECS_INTERDOMAIN`. Either of the channels has *remote_dom* and *remote_port* attributes pointing to each other.

As long as interdomain communication is complete, either domain can close its own event channel. Specifically, DomA decides to close its channel by calling the operation "`EVTCHNOP_close`" in the hypercall. During the closing process, DomB's event channel's state restores as `ECS_UNBOUND`. Moreover, DomA's state alters as `ECS_FREE`. The state change of event channel during a connection life cycle between DomA and DomB is described in figure 1.

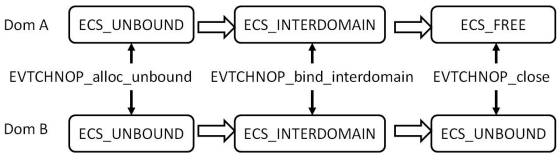

Fig. 1. State Change of Event Channel during Connection Life Cycle

During the use of event channels, it might be necessary to retrieve the state of them. However, the *state* variable cannot be accessed by guests directly. Guests have to inquire the *status* by calling the operation "`EVTCHNOP_status`" in the hypercall. From Table 1, we can see the differences between state and status.

3 Covert Channel Using Event Channel State

3.1 Threat Scenario

In a host of cloud environment where Xen is installed, any domain may store some sensitive information such as the user's password and identity information. We assume that an attacker is capable of injecting a spyware such as Trojan into the guest in a domain. According to the isolation between domains, other domains do not have the ability to access it or eavesdrop it. Thus, the user or the VMM administrator usually adopts network traffic monitoring software that monitor how the data stream transmit, specific-configured firewall or IDS(Intrusion Detection Systems) that inspect the data packets transport in the network, VPN(Virtual Private Network)that encrypts the data, to prevent data leakage. Since the presence of these facilities, the spyware could not send the sensitive data through Internet. Once there is abnormal data stream sent to outside, the VMM administrator will notice or receive an alert. However, we find that there is still a special information transmission path the attacker can make use of. In our scenario, this domain owns sensitive information is called *Sender Domain*. Like the normal users, the spyware owner can rent(having not controlled

Dom0) or create(having controlled Dom0) a domain beside the *Sender Domain* on the same hypervisor, that is to say, co-resident with the *Sender Domain*. We presume the co-residence is always possible[16]. In most cases, *Receiver Domain* is controlled by the attacker, so it doesn't enable the additional facilities like *Sender Domain*. The spyware in *Sender Domain* sends user's private or secret information to an unrecognized covert channel(e.g. CCECS, covert channel using event channel state), through which a program in *Receiver Domain* can acquire the data and send it to the attacker's host through Internet.

3.2 A New Kind of Covert Channel

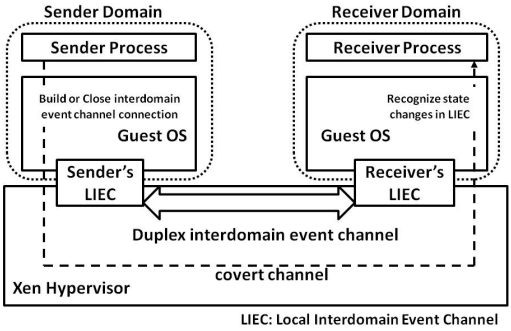

Fig. 2. Structure of CCECS

As depicted in Figure 2, the new CCECS(Covert Channel using Event Channel State) is different from duplex interdomain event channel(see section 2.2).

According to the event channel mechanism mentioned above, the sender process runs a spyware program in the *sender domain* which can read the sensitive information that attackers are interested in. But the receiver process in the *receiver domain* of the attacker is not granted to share the secrets, though the event channel is permitted between guests. In order to construct the CCECS, the sender process would send sensitive information by building or closing interdomain event channel connection to local end of the connection, which will change the state of local end. Meanwhile, the receiving end of the connection could change its state synchronously. So the receiver process could acquire the secrets by recognizing the changes of the local end's state(see figure 1).This covert channel exists between guests independently of the domain(dom0 or domU).

3.3 Request Hypercall from DomU

In most cases, OS kernel runs in Ring 1 can use the hypercall services directly. However, the guest application running in Ring 3 may need to request hypercalls.

Fortunately, Xen kernel offers a privileged command driver `/proc/xen/privcmd`[1] to invoke hypercalls. Applications in user space can request hypercalls by calling *ioctl* function with necessary parameters. Thus, it is feasible for process in DomU to operate event channels.

3.4 Communication Protocol

In this section, we describe the communication protocol of CCECS. The sender and the receiver need to act in a synchronized manner. Therefore, both processes must synchronize the timing of communication before sending bit-streams. When an interdomain connection is constituted by calling hypercall, both end's state is `ECS_INTERDOMAIN`. When one end closes, remote end is set to `ECS_UNBOUND` and its own state is `ECS_FREE`. After analyzing the source code of Xen, we know that the state-setting instructions are separated by a few lines of instructions. Thus, in terms of the powerful computing capability of host CPUs, the time interval is so short that it is negligible. Once the sender end's state changes, the receiver's alters correspondingly. We can see the process in Figure 3.

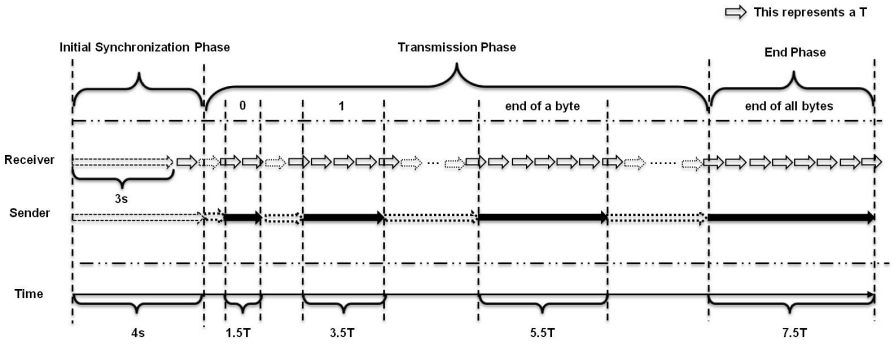

Fig. 3. Communication Protocol of CCECS

Initial Synchronization Phase. In Figure 3, the initial synchronization phase shows our timing setting for the sender and receiver. Because building interdomain event channel is done by hypercall, so it takes longer time than normal function calls, which execution time is about `1s`. Empirically, we let the receiver dectect its own state every `ns`(e.g. `3s`) to ensure that the synchronization request from the sender is always not missed. And we let the sender wait `(n+1)s`(e.g. `4s`) before sending data to ensure enough time to switch receiver's console to the process of receiving data. During the phase, it includes three operations in order: First, the receiver domain should request an event channel and let the sender domain be its remote end, and then its initial state is

[1] The driver have been re-engineered as /dev/xen/privcmd since the Linux 3.3 was released.

ECS_UNBOUND,that is the EVTCHNSTAT_unbound for the process in the end. Second, the sender should request a event channel for the interdomain connection with the remote end of receiver domain. When the new event channel is allocated, it will retrieve the remote port number and bind itself to the local port. Once the binding is finished, both end's state becomes ECS_INTERDOMAIN, that is the EVTCHNSTAT_interdomain for the processes in both ends. Third, the receiver would check whether the state of its local event channel is EVTCHNSTAT_interdomain. If that is the case, it indicates both ends share the same timing of the following operations and the transimission phase starts out.

Transmission Phase. After synchronization, the sender and receiver enter the communication phase, in which they communicate $8n(n \geq 1)$ bits information. To stimulate the common situation, the transmission includes file operations. Apart from that, to be flexible, we let both entities share a value T(T μs is configured by an attacker, we test some minimal value of T in section 4). The protocol require a short timespan to convey one bit. The timespan is determined by how long some state of event channel lasts. As long as the local end does the operation EVTCHOP_close(or EVTCHOP_bind_interdomain) of HYPERVISOR_event_channel_op succesfully, it means the sender has sent the bit data '1'(or '0'). Meanwhile, if sending bit '0', the sender suspends its process for 1.5T μs, which is more than 1 time but less than 2 times of T. Otherwise, if sending bit '1', the sender suspends its process for 3.5T μs, which is more than 3 times but less than 4 times of T. We let the receiver measure the state of local event channel every T μs. If the detected times is 1(supposing the previous state has emerged) or 2 for the "EVTCHNSTAT_interdomain" state, it receives bit '0'. If the detected times is 3 or 4 for the "EVTCHNSTAT_UNBOUND" state, it receives bit '1'. Thus, during the transmission phase, no extra waiting time or synchronization time is needed between two bit-sending.

To observe the transmission result more directly, we collects the bits and combine them into text printed on the console. Thus, when the sender finishes the transmission of one byte, we let it suspend 5.5T μs, which is more than 5 time but less than 6 times of T. Afterwards the receiver will detect one state more than 4 times. If the condition is met, the receiver will print the 1 byte data.

End Phase. When communication of all bits finishes, the sender simply stops changing the state of the event channel without doing extra operations. Considering recycling the event channel resources, if the channel is not closed, the sender closes the channel and last the state for 7.5T. Afterward, the receiver will detect the state more than 6 times. And they end the data communication. After that, if both entities try to communicate more information, they have to resume from the initial synchronization phase.

4 Evaluation

Our experimental platform is a HP Compaq 8100 Elite CMT PC with an Intel Core i7-870 running at 2.93 GHz and We use Xen version 4.1.1 as the base

hypervisor. The domain0 in Xen's terminology runs 32-bit Ubuntu 12.04.2 Linux and the two guest domains(DomUs) that make use of the covert channel run para-virtualized 32-bit Ubuntu 12.04.2. Xen assigns two virtual CPUs and 512 MB RAM to each guest VM while domain0 owns four virtual CPUs and 2048 RAM. Every domain uses the kernel 3.5.0-23.

And we conduct the experiments based on the CCECS ommunication protocol mentioned above. There are 1023 Byte secret data stored as a file in the sending end. Our first test target is to show that if the choice of T changes, the receiving end could always get these information at very high bit rate. That is to say, we will test if minimal T exists. Our second test target is to show that if the usage of vCPU changes, the minimal T will increase and the maximal transmission rate will decrease.

First, to get to know the minimal value of T at the error rate less than 5%, we tested the value of T from 100-20000μs in 100μs intervals. Second, we developed a small tool to simulate different execution environment with different usage of vCPU. This tool will run two processes in each DomU to occupy their vCPUs, each process binding itself to one vCPU in the local DomU using CPU affinity. Considering the existed preempted core scheduling algorithm, we also assign each process the highest priority to run in the individual vCPU and then keep them staying on the core as long as possible. With the help of this tool, we simulate the usage of vCPUs from 0%-100% and test the minimal T and maximal bit rate.

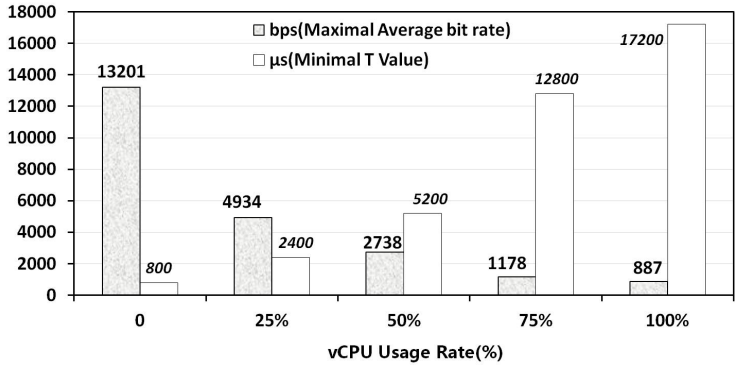

Fig. 4. Maximal Bit Rate and Minimal T at Different Usage Rate of vCPU

The execution results with the error bit rate limited to 5% are shown in figure 4. From 4, we can see that the minimal value of T is directly proportional to vCPU usage. When the vCPU usage is about 0%, the T should not be smaller than 800 μs. But when the usage is about 100%, the T should not be smaller than 17200 μs. Correspondingly, the average bit rates is inversely proportional to vCPU usage. When the vCPU usage is about 0%, the average rate is up to 13201bps, and when the usage is about 100%, the rate is also about 887bps.

As a result, if the T is not smaller than the minimal T, two guests could always share large information using CCECS even though the vCPU is busy.

5 Discussion

Since the exclusiveness of interdomain event channels, the accuracy rate is steadily high if the minimal T is adequate. To lower the danger of the covert channel, two possible solutions are discussed here. First, adding security checks to the operations of creating, binding, closing and quering event channels between domains. As we know, if the XSM(Xen Security Module) is enabled in Xen hypervisorte[14] and the restriction policy is adequately enforced, this covert channel can be controlled to some extent. The second solution is to reset all event channels randomly in each domain. This will affect the time of state change and state query in CCECS, and thus the accuracy of bits transimission will be decreased.

Furthermore, intradomain threat scenario using CCECS may exist when two processes running in the same domain. In this case, the guest OS does not permit them to communication with each other because of the mandatory policy enforced in the operating system. But if the Xen hypervisor does not restrict the event channel between them, the CCECS communication between them may happen and will result in the data leakage from one process to another process. We had evaluated the bit rate also, which is also very high but obviously lower than the interdomain scenario. This is because the interferences between two processes in the same domain is more often than those in the different domains.

6 Conclusion and Future Work

In this paper, we first demonstrate the danger of existing covert channels. Then we introduce the event channel mechanisms of Xen. After that, we design a protocol for CCECS(Covert Channel using Event Channel State) and implement it on Xen hypervisor. Then we do a series of experiments to evaluate CCECS, which can arrive at average 13210bps with error rate less than 5%. Additionally, the state event channels is only impacted by either end of the channel, so the channel is high noise-tolerant. In a word, CCECS is a perilous threat to Xen, its bit rate is larger than most existing covert channel till now.

Our future job is mainly to do some experiments in Amazon EC2 and check if the CCECS still works in the real world. The second aspect is to optimize the transmission scheme because there may be new encoding method to improve the transmission rate of CCECS. The third aspect is to analyze some new communication method introduced in Xen's latest release(e.g. Xen4.3), check if some other factors will downgrade the threat of CCECS.

References

1. Chen, Y., Paxson, V., Katz, R.H.: What's New About Cloud Computing Security? Technical report, UCB/EECS-2010-5, EECS Department, University of California, Berkeley (2010)

2. Reuben, J.S.: A survey on virtual machine security. In: Security of the End Hosts on the Internet, Seminar on Network Security Autumn 2007. Helsinki University of Technology Telecommunications Software and Multimedia Laboratory (2007)
3. U. D. of Defense: Trusted Computing System Evaluation Criteria. DoD 5200.28-STD, Washington (1985)
4. Wang, Z., Lee, R.B.: Covert and Side Channels Due to Processor Architecture. In: Proceedings of the 22nd Annual Computer Security Applications Conference, Washington, pp. 473–482 (2006)
5. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: 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, New York, pp. 199–212 (2009)
6. Xu, Y., Bailey, M., Jahanian, F., Joshi, K., Hiltunen, M., Schlichting, R.: An exploration of L2 cache covert channels in virtualized environments. In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, New York, pp. 29–40 (2011)
7. Wu, Z., Xu, Z., Wang, H.: Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In: Proceedings of the 21st USENIX Conference on Security Symposium, Berkeley, p. 9 (2012)
8. Okamura, K., Oyama, Y.: Load-based covert channels between Xen virtual machines. In: Proceedings of the 2010 ACM Symposium on Applied Computing, New York, pp. 173–180 (2010)
9. Li, Y., Shen, Q., Zhang, C., Sun, P., Chen, Y., Qing, S.: A Covert Channel Using Core Alternation. In: Proceedings of the 2012 26th International Conference on Advanced Information Networking and Applications Workshops, Washington, pp. 324–328 (2012)
10. Wu, J., Ding, L., Wang, Y., Han, W.: Identification and Evaluation of Sharing Memory Covert Timing Channel in Xen Virtual Machines. In: Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, Washington, pp. 283–291 (2011)
11. Xiao, J., Xu, Z., Huang, H., Wang, H.: POSTER: A covert channel construction in a virtualized environment. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, New York, pp. 1040–1042 (2012)
12. Salaün, M.: Practical overview of a xen covert channel. J. Comput. Virol. 6, 317–328 (2010)
13. Ranjith, P., Priya, C., Shalini, K.: On covert channels between virtual machines. J. Comput. Virol. 8, 85–97 (2012)
14. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield., A.: Xen and the art of virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, New York, pp. 164–177 (2003)
15. Amazon Elastic Compute Cloud (EC2), `http://aws.amazon.com/ec2/`
16. Zhang, Y., Juels, A., Oprea, A., Reiter, M.K.: Homealone: Co-residency detection in the cloud via side-channel analysis. In: Proceedings of the 2011 IEEE Symposium on Security and Privacy, Washington, pp. 313–328 (2011)