Sift
An Efficient Method for Co-residency Detection on Amazon EC2

Kang Chen, Qingni Shen, Cong Li, Yang Luo, Yahui Yang and Zhonghai Wu
School of Software and Microelectronics & MoE Key Lab of Network and Software Assurance, Peking University, Beijing, China

Keywords: Co-residency Detection, Cloud Security, Multi-tenancy.

Abstract: Cloud computing, an emerging computing and service paradigm, where the computing and storage capabilities are outsourced on demand, offers the advanced capabilities of sharing and multi-tenancy. But security has been a major barrier for its adoption to enterprise, as being placed with other tenants on the same physical machine (i.e. co-residency or co-location) poses a particular risk. Former research has shown how side channels in shared hardware may enable attackers to exfiltrate sensitive data across virtual machines (VMs). In view of such risks, tenants need to be able to verify physical isolation of their VMs. This paper presents Sift, an efficient and reliable approach for co-residency detection. Through a pre-filtration procedure, the time for co-residency detection could be significantly reduced. We describe the cloud scenarios envisaged for use of Sift and the accompanying threat model. A preliminary validation of Sift has been carried out in a local lab Xen virtualization experimental platform. Then, using the Amazon's Elastic Compute Cloud (EC2) as the test platform, we evaluate its practicability in production cloud environment. It appears that Sift can confirm co-residency with a target VM instance in less than 5 seconds with an extremely low false rate.

1 INTRODUCTION

Cloud computing has become an essential technology. Commercial third-party clouds allow businesses to avoid over provisioning their own resources and to pay for the precise amount of computing that they require. By placing many virtual hosts on a single physical machine, cloud service providers (CSPs) are able to profitably leverage economies of scale and statistical multiplexing of computing resources. Such as Amazon's Elastic Compute Cloud (EC2) service, it offers a set of virtualized hardware configurations for tenants.

However, this practice of multi-tenancy also introduces the risk of sharing a physical server with an arbitrary and potentially malicious VM, which enables various security attacks in the public cloud. There exist attacks that break the logical isolation provided by virtualization to breach confidentiality (Hund et al., 2013); (Ristenpart et al., 2009); (Wu, 2012); (Xu, 2011); (Yarom and Falkner, 2013); (Zhang et al., 2014); (Zhang et al., 2012) or degrade the performance (Varadarajan et al., 2015); (Zhou, 2011) of the victim. Most notable are the side-channel attacks that steal private keys across the virtual-machine isolation boundary by cleverly monitoring shared resource usage (Yarom and Falkner, 2013); (Zhang et al., 2014); (Zhang et al., 2012). Although, defenses against such vulnerabilities and researches on VM allocation and scheduling policies to mitigate multi-tenancy risks (Bijon et al., 2015); (Han et al., 2014) are already being proposed in the academic literature (Godfrey and Zulkernine, 2014); (Godfrey and Zulkernine, 2013); (Raj et al., 2009), multi-tenancy risk still cannot be ignored.

Venkatanathan Varadarajan et al., (2015) have investigated the problem of placement vulnerabilities and quantitatively evaluated three popular public clouds, including Amazon EC2, Google Compute Engine and Microsoft Azure, for their susceptibility to co-location attacks. The most important pre-condition for this kind of research is a reliable co-residency detection approach. In recent years, feasible detection approaches have been proposed (Bates et al., 2012) (Ristenpart et al., 2009) (Zhang et al., 2011), but some of them are not effective at all since the adoption of stronger isolation technologies such as Virtual Private Clouds

Chen, K., Shen, Q., Li, C., Luo, Y., Yang, Y. and Wu, Z.
Sift - An Efficient Method for Co-residency Detection on Amazon EC2.
DOI: 10.5220/0005742004230431
In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, pages 423-431
ISBN: 978-989-758-167-0

(VPCs).

In this paper, we propose Sift, an efficient and reliable approach for co-residency detection. We find a new way to reliably detect co-residency, and the pre-filtration procedure we have raised can improve the efficiency. Since most VMs could be excluded through a pre-filtration procedure and filtration on VMs consumes very little time.

This paper makes the following contributions:

- **We put Forward a New Way for Co-residency Detection.** Unlike existing detection schemes, our co-residency detection scheme is based on the physical sharing resources.
- **The Efficiency can be Significantly Improved through a Pre-filtration Procedure.** Through the analysis of those VMs already co-reside on identical physical machine, we have raised an improved co-residency detection scheme.
- **Demonstration on Amazon EC2.** We use multiple customer accounts to launch VM instances under different strategies to simulate the actual deployment of VMs. Then we use Sift to detect co-residency on those VMs. These tests confirm its efficiency and practicality on commodity clouds.

2 RELATED WORK

Tomas Ristenpart et al., (2009) first exposed the co-residency detection of VMs in 2009. Since then, the co-residency detection problems of VMs have become a new research hotspot and the researchers proposed a lot of co-residency detection methods based on different techniques.

Tomas Ristenpart et al., (2009) indicated that, on the EC2 platform, we can judge the co-residency of VMs through simple network-topology-based co-residency checks. This co-residency detection approach is the simplest way to implement, but its accuracy cannot be guaranteed. As it is based on network information, it can be influenced by firewall policies, network traffic flow and so on, so it has a critical limitation. Actually, this simple network-topology-based co-residency check is not usable anymore since the adoption of VPC.

Adam Bates et al., (2012) proposed the co-residency watermark technique based on the network packet delay problem of co-resident VMs. This technique constructed a side channel skilfully based on the network packet delay caused by multiplexing of the physical network card. An adversary can use this side channel to detect co-residency with target server. This technique also has its own limitation. If the service provider restricted the upper limit of bandwidth, this co-residency detection method would fail. If service providers provide each VM with a dedicated network export, this co-residency detection method will also fail.

Yinqian Zhang et al., (Zhang et al., 2011) used L2 memory cache to construct a co-residency detection tool HomeAlone. Different from the previous two methods, the key idea in HomeAlone is to invert the usual application of side channels. Rather than exploiting a side channel as a vector of attack, HomeAlone uses a side-channel (in the L2 memory cache) as a novel, defensive detection tool. By analysing cache usage during periods in which “friendly” VMs coordinate to avoid portions of the cache, a tenant using HomeAlone can detect the activity of a co-resident “foe” VM. HomeAlone has two difficulties. One is how to accurately distinguish Cache behaviour between normal tenant's friendly VM and co-resident VM. Another is how to ensure that the performance of friendly VM will not reduce greatly.

3 SYSTEM DESIGN

3.1 Practical Application Scenarios

3.1.1 Attack Scenario

We assume system administrators are not interfering with the activities of their customers, and will not intervene with customer behaviour unless it is a threat to Service Level Agreements (SLAs) or to the general health of their business. We also assume that our target VM (victim) is trusting of the cloud infrastructure.

As we need to implement communication between target VM and the VMs launched by the adversary, a receiving end has to be established on the target VM. It is mentioned above that co-residency attacks in public clouds involve two steps: a launch strategy and co-residency detection. The focus of this study is to identify if there exists any VMs co-reside with target VM. Here we will explain a reasonable threat model for establishing the receiving end.

In fact, this pre-condition is able to achieve through existing attack technique, e.g., Bundled Software technology. As the operations you are trying have no harm on the system security, it will not be detected by security software. Establishing a receiving end on target VM has nothing to do with any complex or sensitive privileged operation and the entire process from establishing a receiving end to achieving co-residence consumes little time, which ensures a high imperceptibility.

We assume that the adversary is in the disguise of a legal software package source. If the user of target VM accesses this source to update or install any software, a receiving end will be established, benefiting from the bundled software technology. Then, the essential information needed for pre-filtration will be forwarded to the adversary through the normal communication process between target VM and adversary. Once receiving this information, the adversary follows the first step of co-residency attack to launch a large number of VM instances. Then, following the process of Sift, the adversary can find the expected VM in a very short time.

3.1.2 Benign Scenario

We have mentioned in the Introduction section that Venkatanathan Varadarajan et al., (2015) have investigated the problem of placement vulnerabilities and quantitatively evaluated three popular public clouds for their susceptibility to co-location attacks. A key technique for understanding placement vulnerabilities is to detect whether VMs are co-resident on the same physical machine. Thus, a reliable co-residency detection approach is the most important pre-condition for this kind of research. Apparently, Sift is quite a suitable choice for research demand.

3.2 System Overview

We have analysed those VMs already co-reside with each other from a local lab environment to production cloud environments. The reason we perform an analysis of those VMs is to find the internal relation among them and set the filtration criteria. Then we can conduct a pre-filtration on VM instances we launched before. However, VMs meeting the filtration criteria might not be co-resident with each other, which could lead to false negative. So we still need a reliable approach for co-location test to further detect co-residence after pre-filtration procedure.

In this paper, we present a new way to test by using physical sharing resources. A covert channel for communication can be constructed based on the physical sharing resources. The pre-condition for VMs to use this covert channel for communication is that they must have co-resident relationship. That is to say, as long as the communication succeeds, these two VMs are co-resident. This ensures the accuracy and reliability of co-location test.

Sift consists of three components, **Collecter**, **Communicator** and a **Pre-filter**. A filtration criteria could be set through the analysis of VMs' property. The Collecter on each VM is responsible for collecting essential information and sending the information to the Pre-filter outside the cloud. The Communicator on target VM is to establish a receiving end based on physical sharing resources. The Pre-filter is like the brain of Sift. After receiving the information sent by Collecters, Pre-filter will carry out a filtration on the VMs besides target VM. Then it will make the decision and inform qualified VM to establish a sending end. According to the result of incoming communication process, the sending end will report to the brain whether co-residency is achieved.

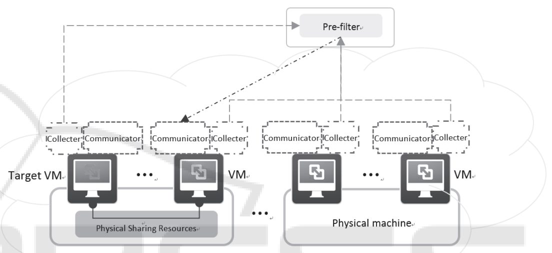

Figure 1: System architecture.

4 IMPLEMENTATION ON AMAZON EC2

4.1 Analysis of VM Instances

We have implemented Sift on Amazon EC2. In our experiment, the domain id is taken as the filtration criteria. Amazon EC2 is based on Xen virtualization technology. In the Xen virtualization technology, each Domain (VM) has a unique identifier called the domain id (domid). Differing from the UUID of each VM, the value of UUID does not change during the lifecycle of a VM, but domid does. When a VM reboots, the domid of a VM will change. Through analysing the source code of Xen (version 4.4.1), we have found that the Domain Us managed by same Domain 0 have adjacent domid.

The code is located in xen/common/domctl.c. As showed in Figure 2, the rover is a static variable. We can see that it reserves the last assigned domid. Then, this value will increase by 1 and be assigned to the next domain, which means when a VM boot, it will be assigned a new domid by increasing 1. When domid reaches the first reserved value, it will be reset to 1.

```
case XEN_DOMCTL_createdomain:
{
   domid_t       dom;
   static domid_t rover = 0;
   …
   for ( dom = rover + 1; dom != rover; dom++ )
   {
      if ( dom == DOMID_FIRST_RESERVED ) //0x7FF0U
         dom = 0;
```

Figure 2: Assignment of domid in Xen 4.4.1.

EC2 uses a customized version of Xen. We are not able to obtain its source code. But Amazon EC2 has provided dedicated service to users, so that we can rent a physical machine. We deployed several VMs on this physical machine and tried to get the domid of these VMs. It was found that the domid of VMs on EC2 presented the same regularity. If Amazon EC2 changes the assignment scheme of domid, for example, using a randomized assignment scheme. Here is a doubt whether this change has an effect on our approach. We will talk about it in the Discussion section. We assume that EC2 would not change the domid assignment scheme, and our work is under this pre-condition.

4.2 Collecter

In general case, domid is not visible to EC2 users. Our Collecter can acquire domid through the utilization of XenStore. XenStore is an inter-domain sharing storage system which is managed and maintained by Domain 0. It stores configuration of all VMs (include Domain 0), e.g., the domain's name, domid and UUID. There are three paths in XenStore:

- /vm directory stores the configuration of domain.
- /local/domain directory stores information of running domain. Each subdirectory represents a running VM like Linux proc file system, e.g., /local/domain/0 represents Domain 0.
- /tool directory stores information of all tools.

As a privileged domain, Domain 0 can read and write all data stored in XenStore. However, as the Domain U, EC2 users only have access to its own data. The first reserved domid is 32752(*0x7FF0U*) as showed in Figure 2, which implies that the domid of Domain U ranges from 1 to 32751.

Collecter tries to access /local/domain/<domid> on a VM. As the directory can only be accessed by the domain which has the corresponding domid (e.g., /local/domain/1000 can only be accessed by Domain 1000). The core pseudo code of Collecter to get domid is shown in Figure 3. *Access* represents access operation to the directory. We can determine VM's domid according to its return value. If access succeeds, then the current *i* is VM's domid.

```
for i =1 to 32751{
       if(Access("/local/domain/i"))
              return domid = i;
}
```

Figure 3: Assignment of domid in Xen 4.4.1.

4.3 Pre-filter

Through the analysis of VM instances, we have found that the Domain Us managed by same Domain 0 have adjacent domid. As the capacity of physical machine has a limit, we can assume that a physical machine can run *x* VMs. Therefore, we can infer from these facts that difference between two co-resident VMs' domid is less than *x* (greater than zero as well). Pre-filter should carry out the filtration procedure according to the following algorithm.

Assume that we have a sample set $X=\{x_1, x_2 \ldots x_n\}$, and we need to find all possible co-resident VMs in this set. Each element represents the VMs' domid. In accordance with the principle of the nearest neighbour clustering, the algorithm is as follows:

- Step 1. Select a distance threshold *x* (i.e. capacity of physical machine), and take a sample as the clustering center of first cluster Z_1 (e.g., x_1).
- Step 2. Calculate the distance to x_1 (i.e. D_n) of all the rest sample, if $D_n < x$, then $x_n \in Z_1$. Choose another unclassified sample (i.e. not belong to any cluster) as the clustering center of the second cluster Z_2 (e.g., x_m).
- Step 3. Repeat until all VMs have been classified. Notice that every clustering center can only belong to one cluster and the other sample can appear in several clusters (i.e. overlap between partitions is allowed for the accuracy and efficiency).
- Step 4. Check every cluster, if a cluster has only one element, it will be excluded. The remaining clusters are possible to be co-resident.

Although when a VM boot, it will be assigned a new domid by increasing 1, when it shutdown, this used domid will not be recycled. So there may be two co-resident VMs with domid differs a lot. For example, assume there are two VMs running on the same host, with VM_1 assigned domid=1 and VM_2 assigned domid=2. Then VM_2 reboot, it will be assigned domid=3. Reboot again, domid=4, 5, 6…. This special case might be missed when we have set up a threshold value, which will lead to false negative. Therefore, we have arranged two experiments. Two groups of VMs are launched in different time. The time interval for first experiment is 12 hours, and the time interval for second experiment is 24 hours. During the time interval, the other VMs on the physical host might reboot which will consume the domid. We set these two experiments to simulate that special case, and test the rate of false negative for Sift.

4.4 Communicator

After the Pre-filtration, the Pre-filter will inform the qualified VMs to establish a communication process using physical sharing resources. We have utilized the physical sharing resources of Xen VM monitor, such as event channel, grant table.

Event channel is an asynchronous notification mechanism provided by Xen VM monitor for VMs to exchange information. We can try to establish event channel between two VMs. These two VMs can be determined as co-resident if the event channel can be established successfully. Further, we can construct a covert channel based on event channel for co-location test, such as the *CCECS* been proposed in our previous work (Shen et al., 2013). If two VMs can use *CCECS* to establish a communication process, the co-residency is achieved.

In addition to covert channel based on event channel, any other covert channels satisfy the following conditions could be applied to co-residency detection.

- It is built on the physical sharing resources of VM monitor.
- It can be carried out on commodity cloud, such as Amazon EC2 to ensure the practicability
- It should be stable enough and have the ability of anti-interference to ensure the reliability

4.5 Experiment Strategies

We use two Amazon EC2 accounts (account A and B) to launch VM instances under different strategies that simulate the actual deployment of VMs. The VMs we launch have the same configuration.

We have designed two sets of experiments. The first set consists of three experiments corresponding to three specific availability zones (us-west-2a, us-west-2b, us-west-2c). We use account A and account B to launch 20 VMs at the same time, since EC2 has a limitation that each account can only launch 20 VMs. The reason why we launch VMs at same time is to maximize the likelihood of co-residency, so we can demonstrate Sift better. These three experiments are denoted as experiment 1-a, 1-b, 1-c. Corresponding to the three zones.

It has been mentioned in section 4.3 that a special case might lead to a false negative. Therefore, in the second set of experiments, we have arranged two experiments. Two groups of VMs are launched in different time. The time interval for first experiment (denoted as experiment 2-1) is 12 hours, and the time interval for second experiment (denoted as experiment 2-2) is 24 hours. We set these two experiments to simulate that special case, and test the rate of false negative for Sift.

We have noticed that each account can only launch 20 VM instances on Amazon EC2. Meanwhile, Amazon EC2 has provided a dedicated service to users. So we can infer that Amazon just needs to guarantee that each physical machine could hold 20 VMs of any type. Thus, we set the value of x to 20. We will discuss how to choose a proper value for x in Discussion. The results of experiments will be given in Evaluation section.

5 EVALUATION

5.1 Experimental Data of First Set

The domid of VM instances in experiment 1-a listed in Table 1 is in an ascending order for the convenience of analysing.

Table 1: Domid of VM instances in Experiment 1-a.

No.	domid	No.	domid	No.	domid	No.	domid
A1	437	A11	1039	B1	223	B11	1295
A2	494	A12	1056	B2	314	B12	1317
A3	514	A13	1174	B3	530	B13	1320
A4	635	A14	1176	B4	531	B14	1320
A5	640	A15	1439	B5	548	B15	1517
A6	773	A16	1581	B6	1001	B16	1548
A7	818	A17	1941	B7	1133	B17	1558
A8	952	A18	1942	B8	1196	B18	1569
A9	1000	A19	2117	B9	1240	B19	1615
A10	1308	A20	4948	B10	1262	B20	9009

It is mentioned in section 4.1 that each Domain U managed by same Domain 0 has a unique domid. So it can be sure that VMs have same domid are definitely not co-resident. Besides, VMs launched by same account are not co-resident (Ristenpart et al., 2009), which has been proved in our experiment. We should take these into consideration during pre-filtration procedure.

For example, we can take A3 as the first clustering center. As the VMs launched by same account are not co-resident, we can just search in account B for expected VMs. Here exist B3 and B4 meet the rule. Next, we should pick another clustering center from the remaining 37 unclassified VMs and search for those expected VMs. Then repeat these steps until all VMs have been classified. Remember that overlap between partitions is allowed. Figure 4 illustrates the process of pre-filtration in experiment 1-a.

As VMs launched by same account are not co-resident, we can just do co-location test using covert channel (namely CCECS) between clustering center and other elements. That is to do co-location test between A2 and B3, A2 and B4, A9 and B6, A16 and B18. If the test is successful, then the receiving VM and sending VM can be judged as co-resident. But unfortunately, no VMs are proved to be co-resident in experiment 1-a.

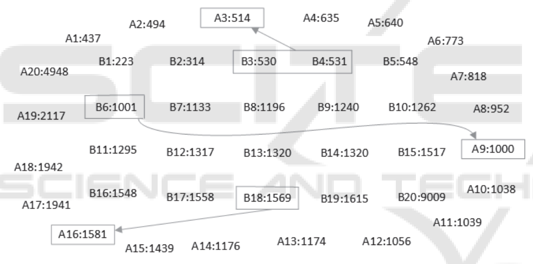

Figure 4: The result of pre-filtration is: A3{B3, B4}(means that A3, B3, B4 belong to same cluster with A3 being the clustering centre), A9{B6}, A16{B18}.

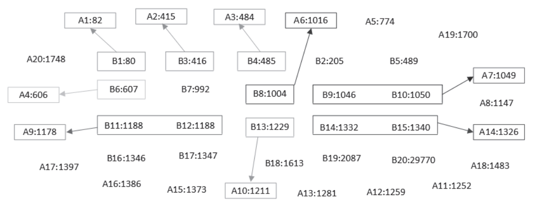

Figure 5: The result of pre-filtration is: A1{B1}, A2{B3}, A3{B4}, A4{B6}, A6{B8}, A7{B9, B10}, A9{B11, B12}, A10{B13}, A14{B14, B15}.

Without listing the domid of VMs in Table again, we directly illustrate the other two experiments through graphs. Figure 5 illustrates the process of pre-filtration in experiment 1-b. There were 12 co-location tests need to be carried out in total. After that, we got 4 pairs of VMs that were co-resident. They were A2-B3, A3-B4, A4-B6, A7-B10.

Figure 6 illustrates the process of pre-filtration in experiment 1-c. There were 73 co-location tests need to be done in total. After that, we got 17 pairs of VMs that were co-resident. They were A1-B3, A2-B4, A3-B5, A4-B6, A5-B7, A6-B8, A7-B9, A8-B10, A9-B11, A10-B12, A11-B13, A12-B14, A13-B15, A14-B16, A15-B17, A16-B18, A20-B20.

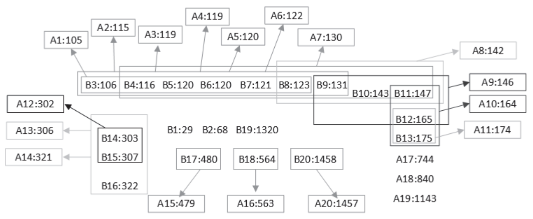

Figure 6: The result of pre-filtration is: A1{B3, B4, B5, B6, B7, B8}, A2{B3, B4, B5, B6, B7, B8, B9}, A3{B3, B4, B5, B6, B7, B8, B9}, A4{B3, B4, B5, B6, B7, B8, B9}, A5{B3, B4, B5, B6, B7, B8, B9}, A6{B3, B4, B5, B6, B7, B8, B9}, A7{B4, B5, B6, B7, B8, B9, B10, B11}, A8{B8, B9, B10, B11}, A9{B9, B10, B11, B12}, A10{B11, B12, B13}, A11{B12, B13}, A12{B14, B15}, A13{B14, B15, B16}, A14{B14, B15, B16}, A15{B17}, A16{B18}, A20{B20}.

In order to prevent the false negative, we have carried out exhaustive co-location tests for all possible VM combination in all three experiments, in other words, 400 co-location tests for each experiment. It turns out that no false negative has been produced. In terms of efficiency, the number of co-location test has been greatly reduced with the help of pre-filtration. The efficiency is improved indeed.

5.2 Experimental Data of Second Set

The time interval for experiment 2-1 is 12 hours. We first use account A to launch 20 VM instances, and use account B to launch 20 VM instances after 12 hours. The time interval for experiment 2-2 is 24 hours. During the time interval, there might happen a lot of reboot operations, which will consume the domid. We set these two experiments to simulate that special case, and test the rate of false negative for Sift.

Figure 7 illustrates the process of pre-filtration in experiment 2-1. There were 5 co-location tests need to be done in total. After that, we got 5 pairs of VMs that were co-resident. They were A1-B3, A3-B7, A4-B11, A5-B13, A18-B18.

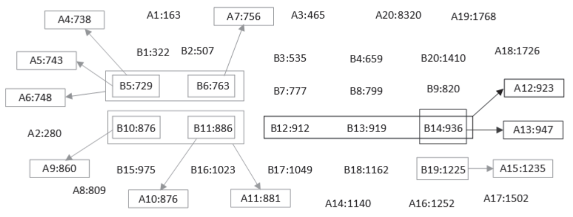

Figure 7: The result of pre-filtration is: A1(B3), A3(B7), A4(B11), A5(B13), A18(B18).

Figure 8 illustrates the process of pre-filtration in experiment 2-2. There were 14 co-location tests need to be carried out in total. But unfortunately, there were no VMs proved to be co-resident in experiment 2-2 as well.

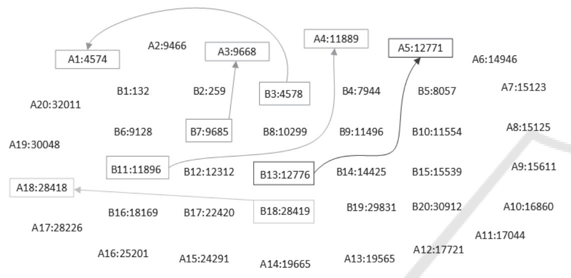

Figure 8: The result of pre-filtration is: A4(B5), A5(B5), A6(B5, B6), A7(B6), A9(B10), A10(B11), A11(B10, B11), A12(B12, B13, B14), A13(B14), A15(B19).

We can learn from the experimental data, the biggest difference of domid between co-resident VMs is 13, these two VMs are A3 and B7 in experiment 2-1. The reason why we arrange this set of experiment is to simulate the special case we discussed in previous section. We have carried out exhaustive co-location tests for all possible VM combination in both two experiments. But this exceptional case has not occurred as we expect, and no false negative has been produced. We speculate that this special case is a small probability event. It might happen, but its impact on Sift is negligible.

5.3 Performance Analysis

We can find that the number of co-location tests in five experiments have been greatly reduced from 400 to 4, 12, 73, 5, 14 respectively. While the filtration on VMs consumes very little time, close to the time needed for a co-location test.

Suppose there are *M* accounts, each account can launch *N* VMs. Considering the fact that VMs launched by same account are not co-resident, we have $C(M,2) \bullet N^2$ possible VM combinations for co-location test. So the time complexity is $O(M^2 \bullet N^2)$, e.g., we use two accounts and each account can launch 20 VMs, as a result, we have 400 possible VM combinations for co-location test. In contrast, if we implement a pre-filtration procedure before co-location test, we can improve the time complexity. Considering the extreme case that each cluster has $2(x-1)$ elements, we need to do $M \bullet N(x-1)$ co-location tests at most. As x can be considered as a constant, so the time complexity is $O(M \bullet N)$. It can be seen by comparing the time complexity that the adoption of a pre-filtration procedure can significantly reduce the time of co-residency detection.

6 DISCUSSION

The value of *x* depends on the following factors: the performance of physical machine, instance's type, and the number of VMs that a physical machine should hold defined by CSP. In general case, the capacity of physical machine for different types of instance is diverse, for example, the capacity for medium type VM on Amazon EC2 is 8 (Ristenpart et al., 2009). Furthermore, the value of *x* we set could directly affects the efficiency and accuracy of co-residency detection. When the selected value of *x* is greater, the accuracy is higher, but the efficiency would be lower. When the selected value of *x* is smaller, the accuracy would be relatively reduced, and it may appear some omissions. You could adjust the value of *x* according to the result of the experiment to find the most accurate value of *x*.

The reason why we choose domid as the filtration criteria during pre-filtration procedure on Amazon EC2 is that a VM will be assigned a new domid by increasing 1 when it boot. So co-resident VMs have adjacent domid. If Amazon EC2 changes the assignment scheme of domid, for example, using a randomized assignment scheme. It is a problem what impact there will be. Actually, the output number sequence is fixed when the seed of pseudo random number generator is certain. So it is possible for us to define a mapping from this random number sequence to a linear growth sequence.

We have mentioned that the domid will not be recycled when a domain reboot, which could lead to a special case that two VMs co-reside on same host with domid differs a lot. We have to admit that this special case does exist, so we have arranged experiment 2-1 and 2-2 to simulate this special case. However this special case has not occurred as we expect, and no false negative has been produced. We speculate that this special case is a small probability event. It might happen, but its impact on Sift is negligible.

7 CONCLUSIONS

In this paper, we proposed Sift, an efficient and reliable approach for co-residency detection. A detailed introduction of this detection scheme was presented, and the threat model for Sift was explained as well. Through an extensive series of tests, we have implemented Sift on Amazon EC2. Through the analysis of experimental data and the computation of complexity, we have proved its practicality and efficiency. Finally, we made a discussion about how to select a proper value for *x* and several problems of Sift.

Our future work will focus on improving Sift. We will solve the leftover problems first and then implement it on other cloud platforms to assure whether Sift is still feasible.

ACKNOWLEDGEMENTS

This work is supported by the National High Technology Research and Development Program (“863” Program) of China under Grant No. 2015AA016009, the National Natural Science Foundation of China under Grant No. 61232005, and the Science and Technology Program of Shen Zhen, China under Grant No. JSGG20140516162852628.

REFERENCES

Bijon, K., Krishnan, R., & Sandhu, R. (2015, June). Mitigating Multi-Tenancy Risks in IaaS Cloud Through Constraints-Driven Virtual Resource Scheduling. In Proceedings of the 20th ACM Symposium on Access Control Models and Technologies (pp. 63-74). ACM.

Bates, A., Mood, B., Pletcher, J., Pruse, H., Valafar, M., & Butler, K. (2012, October). Detecting co-residency with active traffic analysis techniques. In Proceedings of the 2012 ACM Workshop on Cloud computing security workshop (pp. 1-12). ACM.

Godfrey, M., & Zulkernine, M. (2014). Preventing Cache-Based Side-Channel Attacks in a Cloud Environment. Cloud Computing, IEEE Transactions on, 2(4), 395-408.

Godfrey, M., & Zulkernine, M. (2013, June). A server-side solution to cache-based side-channel attacks in the cloud. In Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on (pp. 163-170). IEEE.

Han, Y., Chan, J., Alpcan, T., & Leckie, C. (2014, June). Virtual machine allocation policies against co-resident attacks in cloud computing. In Communications (ICC), 2014 IEEE International Conference on (pp. 786-792). IEEE.

Hund, R., Willems, C., & Holz, T. (2013, May). Practical timing side channel attacks against kernel space ASLR. In Security and Privacy (SP), 2013 IEEE Symposium on (pp. 191-205). IEEE.

Ristenpart, T., Tromer, E., Shacham, H., & Savage, S. (2009, November). 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 (pp. 199-212). ACM.

Raj, H., Nathuji, R., Singh, A., & England, P. (2009, November). Resource management for isolation enhanced cloud services. In Proceedings of the 2009 ACM workshop on Cloud computing security (pp. 77-84). ACM.

Shen, Q., Wan, M., Zhang, Z., Zhang, Z., Qing, S., & Wu, Z. (2013). A covert channel using event channel state on xen hypervisor. In Information and Communications Security (pp. 125-134). Springer International Publishing.

Varadarajan, V., Zhang, Y., Ristenpart, T., & Swift, M. (2015, August). A placement vulnerability study in multi-tenant public clouds. In 24th USENIX Security Symposium (USENIX Security 15)(Washington, DC (pp. 913-928).

Varadarajan, V., Kooburat, T., Farley, B., Ristenpart, T., & Swift, M. M. (2012, October). Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In Proceedings of the 2012 ACM conference on Computer and communications security (pp. 281-292). ACM.

Wu, Z., Xu, Z., & Wang, H. (2012, August). Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In USENIX Security symposium (pp. 159-173).

Xu, Y., Bailey, M., Jahanian, F., Joshi, K., Hiltunen, M., & Schlichting, R. (2011, October). An exploration of L2 cache covert channels in virtualized environments. In Proceedings of the 3rd ACM workshop on Cloud computing security workshop (pp. 29-40). ACM.

Yarom, Y., & Falkner, K. E. (2013). Flush+ Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. IACR Cryptology ePrint Archive, 2013, 448.

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

Zhang, Y., Juels, A., Reiter, M. K., & Ristenpart, T. (2012, October). Cross-VM side channels and their use to extract private keys. In Proceedings of the 2012 ACM conference on Computer and communications security (pp. 305-316). ACM.

Zhang, Y., Juels, A., Reiter, M. K., & Ristenpart, T. (2014, November). Cross-tenant side-channel attacks in paas clouds. In Proceedings of the 2014 ACM

SIGSAC Conference on Computer and Communications Security (pp. 990-1003). ACM.

Zhou, F., Goel, M., Desnoyers, P., & Sundaram, R. (2011). Scheduler vulnerabilities and attacks in cloud computing. arXiv preprint arXiv:1103.0759.