A Shared Memory based Cross-VM Side Channel Attacks in IaaS Cloud

Ziqi Wang[1], Rui Yang[1], Xiao Fu[1]*, Xiaojiang Du[2], Bin Luo[1]
[1]State Key Laboratory for Novel Software Technology at Nanjing University
[2]Dept. of Computer and Information Sciences at Temple University
{ziqiwang.nju, ruiyang.nju}@hotmail.com, fuxiao@nju.edu.cn, dxj@ieee.org, luobin@software.nju.edu.cn

***Abstract*—Cloud providers usually use virtualization to maximize the utilization of their computing resources, e.g. many virtual machines (VMs) run on a shared physical infrastructure. However co-residency with other VMs will cause high security risks, such as side channel attacks. This kind of attack is difficult to detect and prevent, so it's necessary to study it deeply. Recent research has shown attackers can build up cross-VM side channels to obtain sensitive information. However, due to the features of shared resources (e.g. CPU cache), the sensitive information obtained is usually limited and coarse-grained. In this paper, we present a novel side channel, which is based on shared physical memory and exploits the vulnerabilities of balloon driver. Balloon driver is a very popular mechanism used by current virtual machine managers (VMMs) to balance physical memory among several VMs. Because it is widely used in IaaS cloud, our side channel attack can achieve a high success rate. And compared with current cross-VM side channels, it can transmit more fine-grained data. Using Xen as a case study, we explore how to transmit data by this side channel.**

Keywords—side channel attack; cloud computing; Infrastructure-as-a-Service (IaaS); cloud security

I. INTRODUCTION

According to the NIST [1], cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal cloud service provider's (CSP's) interaction. In other words, cloud providers usually use virtualization to maximize the utilization of their computing resources, e.g. many virtual machines (VMs) run on a shared physical infrastructure. While co-residency with other VMs has brought a lot of advantages, it also causes high security risks, such as side channel attacks. Side-channel attack, which leverages low-bandwidth message channels (e.g., timing, power, cache misses) in a system to derive or leak security-sensitive information, has been proven to be realistic threats to modern computer systems [2]. Recent research has shown attackers can build up cross-VM side channels [3] to obtain sensitive information. However, currently these channels are mostly based on shared CPU cache, networks, CPU loads and so on. Due to the features of these resources, the sensitive information obtained is usually limited and coarse-grained.

In this paper, we present a novel side channel, which is based on shared physical memory and utilizes the vulnerabilities of balloon driver. Balloon driver is a popular mechanism used by current VMMs to balance physical memory among several co-resident VMs. Therefore, if we can put the required information into these pages, and try to actively trigger the balloon drivers of the source VM and target VM, then the information can be "legally" transmitted to the destination without attracting attention from VMM or any other security monitors. Thus it becomes a shared memory based side channel. However, there are two challenges when implementing this channel.

The first one is how to ensure the data in free pages will not be cleared up while transmitting. To achieve high performance, most current operating systems will not clear up the dirty pages immediately after a process's release until a new write operation requires these pages. In addition, to ensure the data confidentiality while balancing memory, the balloon driver also provides a mechanism to clean old data, but this operation only occurs when VM releases idle memory pages to the balloon. Beside this, the balloon and the VMM will not make any inspection to data in the free pages. That is just the vulnerability we can exploit.

The second challenge is how to transmit specific physical pages that contain required data from the source VM to the target VM. The balloon driver cannot adjust memory automatically by itself. All the adjustments are controlled by the VMM. So in order to transmit specific physical pages, some specific measures should be taken to urge the VMM to adjust the shared memory.

We can summarize our side channel attack as following: Firstly, modify the balloon driver to prevent the memory from being cleared up. Secondly, the source VM reads the required information into some memory pages, and then releases them to urge the VMM to recycle these pages into the shared memory pool. Thirdly, the target VM applies for memory as much as possible at the same time to push the VMM to allocate them required memory pages from the shared pool. By this way, attackers can obtain required information from source VM.

This channel can be used as a hidden measure to transmit data between VMs without attracting the VMM's attention. For the wide use of the shared memory and balloon driver in IaaS cloud, this side channel attack can achieve a high success rate and compared with current cross-VM side channels, it can transmit more fine-grained data directly. Furthermore, it can be used to transmit information of any size without limitations. Using Xen as a case study, we introduce two scenarios in detail to explain how to accomplish this attack.

II. Related Work

Side channels have been known for many years. Up to now, many side channels have been demonstrated, but most of them are on the basis of a traditional network environment (e.g. [4][5][6]). Recently with the development of cloud computing, the side channels in cloud have also obtained much attention. They mainly include the following categories:

CPU cache based side channel. This channel can detect the activity of a co-resident VM by analyzing cache usage. Ristenpart et al. [3] shows an attack that identifies whether particular virtual machines are likely to reside on the same physical server in the cloud. Some researchers also use this channel to infer the distribution of VMs [7][8]. In addition, some researchers use it to monitor the keystroke of the victim VM [2] or even extract private keys [9][10].

Timing of computation based side channel. Although the virtualization technology provides strong isolation in cloud, the timing channels can break this isolation by exploring the numerous implicit and high resolution clocks created by the massive sharing resources. In this type of attack, Wang et al. [11] shows a technique for creating a timing channel between VMs, which takes advantage of contention for an arithmetic logic unit on simultaneous multithreading processors. Attackers also can use this side channel to extract confidential information [12], such as keystroke.

Applications or Modules based side channels. One side channel can obtain the browser behavior through tracking changes in the application's memory footprint [13]. Another side channel uses the Kernel Samepage Merging (KSM) module to transmit information between VMs [14].

This paper is motivated by these studies, but is different in several aspects. Our side channel attack is based on shared memory and using the balloon driver to transmit information between co-resident VMs. In our attacks, we can obtain large quantities of fine-gained data rather than limited coarse information, so our attacks do serious harm to the security. Furthermore, the wide usage of the shared memory and the balloon mechanism make our attack have a higher success rate.

III. Background and Threat Model

A. Background

Before introducing our attack, several relevant concepts should be introduced. Firstly, there are three concepts about the configurations of VM's memory: the Current Memory, the Max Memory and the Min Memory.

Definition 1. *Current Memory*: The total memory size in the Domain currently.

Definition 2. *Max Memory*: The maximum memory size the OS can obtain.

Definition 3. *Min Memory*: The minimum memory size the OS owns.

Secondly, to make full use of all kinds of physical resources, the VMM depends on some rules to adjust VMs' memory dynamically. Usually the Adjustment Rule can be described as follows:

Definition 4. *Adjustment Rule*: VMM detects all the memory status of VMs located on the host periodically. For each VM, if the usage rate of its memory is above *the upper limit*, the VMM then will increase its memory size by *the increase unit*, but the total memory size should still be less than the value of the Max Memory after adjustments; oppositely, if the usage rate of its memory is below *the lower limit*, the VMM then will decrease the memory size of it by *the decrease unit*, but the total memory size is still above the Min Memory after adjustments.

In the above rule, the upper limit, the lower limit, the increase unit, and the decrease unit can be defined as follows:

Definition 5. *The upper limit*: The upper bound, above which the VMM will increase the memory size of the VM.

Definition 6. *The lower limit*: The lower bound, below which the VMM will decrease the memory size of the VM.

Definition 7. *The increase unit*: When the VMM increases the memory, the increase unit represents a percentage, by which the memory increases once a time.

Definition 8. *The decrease unit*: When the VMM decreases the memory, the decrease unit represents a percentage, by which the memory decrease once a time.

B. Threat Model

We consider the VMs involved in the side channel attack are all co-resident on a given physical host and they have no control over the functions of the VMM. In our model, we refer to the VM completely under attackers' control as the monitor, the VM partly under attackers' control as the guest and the others as the test. In addition, during an attack, the monitor and the guest can switch between the two different roles: the target and the source VM of the transmission data. We model the cloud provider as neutral and the side channel attack does not need any extra help from the cloud providers.

The preconditions of the threat can be described from four aspects: (1) all VMs must be on the same physical host; (2) all the VMs have installed balloon drivers to support balloon mechanism; (3) the VMM must have adopted some rules to automatically adjust the memory of VMs running on it; (4) at least one of the two VMs must be under the control.

In this setting, we consider two attack scenarios: (1) the attackers use a VM to steal confidential information from a co-resident victim VM; (2) the attackers use a VM to implant some malwares into a co-resident victim VM.

Steal confidential information: In this scenario, the victim VM contains some confidential information and the victim supports the balloon mechanism. Then, attackers want to exploit its co-residency to steal information from the victim. All the operations on the victim are legal. In the procedure of stealing, it is the VMM that controls the adjustment of shared memory and triggers the balloon drivers in the victim according to the rules VMM adopts to balance memory between VMs.

Implant malwares: Attackers may want to implant malicious data into the victim VM. They can divide the malicious data into several parts, and then transmit them separately. Before transmitting, attackers tag every part by the index and special marks. Until the victim receives all the parts of the data, it combines all the parts in a certain order and recreates the malware. During the process, all the operations on the victim are legal, and the victim gains the transmitted malware pieces from its memory.

IV. Design and Implementation

Our shared memory based side channel attack can be divided into three modules (see in Fig.1), i.e. the sender

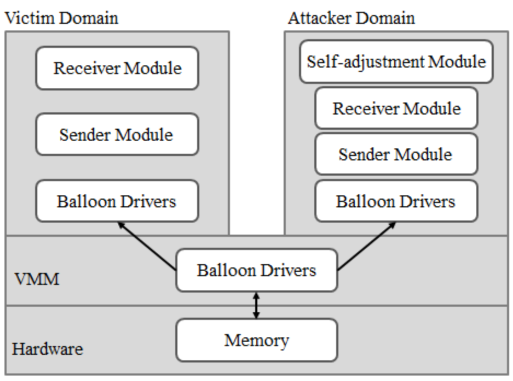

Fig. 1. Structure of the Shared Memory based Side channel Attack

module, the receiver module and the self-adjustment module.

The sender module is in charge of creating the source data to be transmitted and sending it to the shared memory pool, and it consists of the data filling component and the balloon driver's triggering component. The receiver module is responsible for obtaining the transmission data from the shared memory pool, identifying and reconstructing the data. The receiver module also contains two components: the data identification and reception component and the balloon driver triggering component. The self-adjustment module is only adopted in the VMs under attackers' control entirely and this module can complete the memory adjustments by itself in kernel mode. However, there is a premise we should do firstly to carry out our attack. This premise is to make the balloon driver not clean the dirty pages to be transmitted, which can be simply achieved by resetting the configuration of balloon driver. For example, in Linux, the CONFIG_XEN_SCRUB_PAGES should be set to "n".

Therefore, the process of our shared memory based transmission from the source VM to the target VM can be described as the following steps (see in Fig.2). To make it convenient to state, we refer to the target VM under attackers' control as *the target* and the source VM as *the source*.

(1) In the target, attackers should use the sender module to detect the upper limit and the lower limit of the adjustment rule in the VMM. (2) In the target, attackers use the self-adjustment module to release some idle memory pages to the shared memory pool. (3) In the source, attackers use the sender module to write the data to be transmitted into curtain pages. Then, they apply for memory pages constantly to urge the VMM to increase the memory of the source until it reaches the Max Memory. (4) In the source, attackers use the sender module to release the processes space so as to decrease the memory pressure, by which way it urges the VMM to decrease the memory of the source and recycle the memory with transmission data into the shared memory pool. (5) In the target, attackers use the self-adjustment module to reset the memory to the Max Memory to obtain more memory pages from the shared memory pool. (6) In the target, attackers use the receiver module to obtain current memory data, and then scan for specific marks to recreate the original data to be transmitted.

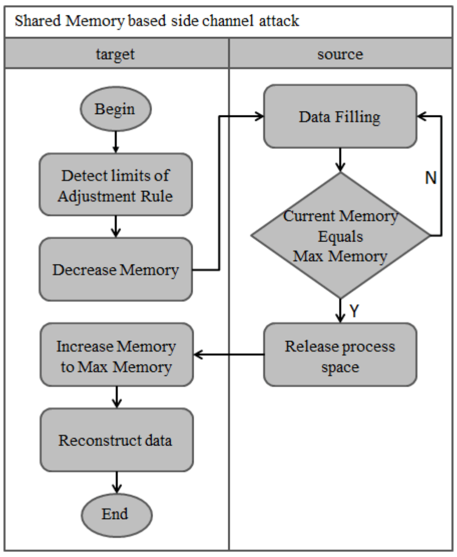
Shared Memory based side channel attack
target
source
Begin
Detect limits of Adjustment Rule
Decrease Memory
Increase Memory to Max Memory
Reconstruct data
End
Data Filling
N
Current Memory Equals Max Memory
Y
Release process space

Fig. 2. The process of the side channel attack

A. The Sender Module

The target of this module is simply creating the source data to be transmitted and sending them to the shared memory. This module can be divided into two components:

1) The data filling component

In this component, it needs to write the data to be transmitted to the memory pages. There are some inputs for this component. One input, called *the address*, refers to the address where the data to be transmitted located in. Another input, named as *the percent*, represents the percentage which the sender wants the memory usage of the data to reach.

It chooses a size as a standard. Then the module divides the data into several small pieces and ensures that the size of every piece is not bigger than the standard. Then reconstruct the data of every small piece. Not only does it need to tag the beginning and the end of every piece, but it also needs to mark the index of each piece with a serial number to make it convenient for the receiver module to recreate the transmission data. The next step is a loop to transmit the data one piece at a time and use the same read program *data_reader* to continuously write a piece of the data to the memory.

After the attackers put the data into memory, what they should do next is ro run another program called *memory_filler* while the program *data_reader* does not close. The *memory_filler* will apply for large numbers of memory to push up the memory usage rate to urge the VMM to increase its memory until the memory size equals the Max Memory.

2) The balloon drivers triggering component

In this component, it offers two functions: *balloon_trigger* and *rule_detector*. The *balloon_trigger* makes the data which is already lying in the memory be recycled into the shared memory pool by the balloon drivers. If the current VM is out of the attackers' control, the measure the function adopts is releasing all the processes apart from the system processes to minimize the memory usage. The function *rule_detector* is in charge of detecting *the upper limit* and *the lower limit* of the *Adjustment Rule* used by the VMM to balance the memory among several VMs. However, the function can only be obtained in VMs under attackers' control completely. The process of the function *rule_detector* can be described as follows, which is displayed in Algorithm 1.

Algorithm 1 Rule Detector Algorithm.

```
1:  initialize the monitor
2:  if memory is changing then
3:    if memory is decreasing then
4:      if Current memory != Min Memory then
5:        return the lower limit = current memory usage rate
6:      end if
7:    else if Current memory != Max Memory then
8:        return the upper limit =current memory usage rate
9:    end if
10: end if
11: if have not got the value of the upper limit then
12:   if Current memory size = Max Memory then
13:     reconfiguration the monitor
14:     return nothing
15:   else increase memory usage step by step
16:     if an increase of the memory occurs then
17:       return the upper limit = current memory usage rate
18:     end if
19: end if end if
20: if have not got the value of the lower limit then
21:   if Current memory size = Min Memory then
22:     increase memory usage step by step
23:     if a decrease of the memory occurs then
24:       return the lower limit = current memory usage rate
25:     end if
26:   else invoke self-adjustment module to increase memory rate
27:    if an decrease of the memory occurs then
28:       return the lower limit = current memory usage rate
29:    else reconfiguration the monitor; return nothing; end if
30: end if end if
```

B. The Receiver Module

The target of this module is identifying and receiving the data transmitted from the Sender Module and then reconstructing the data. This module can also be divided into two components:

1) The balloon drivers triggering component

As our side channel attack is based on shared memory, what we should do firstly in this module is gain the shared memory. This component uses a program named *memory_collector*, which simply invokes the function named malloc, to apply for memory constantly without closing.

2) The data identification and reception component

For VMs attackers cannot fully control, it needs to take advantage of some tools, such as DD, to obtain the memory data, because all the operations in these VMs should be legal and proper. For VMs under attackers' control, the attackers modify the source codes of the OS kernel. In Linux, there is a strategy named Demand Fetch to deal with the pages that do not reside in the current memory. Thus, attackers insert some codes into the physical page allocation algorithm to dump the original memory data into files, by which way the attackers not only obtain the memory data but also increase the memory pressure. We create a function, named *memory_dumper*, to realize this method. In the design of the function *memory_dumper*, we dump all the pages belonging to a certain process into files, and add a lock to avoid the deadlock caused by the function known as vfs_write used in our codes. The logic is displayed in Algorithm 2.

Algorithm 2 Memory Dumper Algorithm.

```
1:  lock = 1
2:  in function page_alloc
3:  pages are cleared up or replaced with new data
4:  if the process = "dumpFile" && lock = 1 then
5:    count = 0, lock = 0
6:    obtain current timestamp to generate the file name
7:    for count = 0 to PAGE_SIZE do
8:      invoke vfs_write to write to file
9:    end for
10: end if
11: lock = 1
12: other page_alloc codes go on working
```

After the module has gained the memory data, it scans the files for special marks to find the transmission data. If the data has been transmitted successfully, the next step is to reconstruct the data.

C. The Self-adjustment Module

This module only applies to the VMs under attackers' control completely. This module only has one input which refers to the target size of pages the user wants to set. In this module, it can dynamically adjust the memory size by itself with the help of a self-adjustment module, which invokes the function known as balloon_set_new_target, provided by the balloon driver.

V. Case Study

In order to present how to use the side channel to conduct

TABLE I. DOMAINS ON XEN

Domain Name	OS	Kernel
Dom0	Ubuntu12.0.4	3.13.4
guest	Ubuntu12.0.4	3.15.1 (modified configuration of balloon driver)
monitor	Ubuntu12.0.4	3.15.1 (modified configuration of balloon driver and added codes of dumping data to files into Linux Kernel)
test	Ubuntu12.0.4	3.13.4

attacks, we choose Xen as the testbed and Ubuntu 12.0.4 as the OS running on it. Xen is an X86 VMM which allows multiple commodity operation systems to share conventional hardware in a safe and resource managed fashion without sacrificing either performance or functionality. The host's total memory size is 4GB, but for Xen, the available total memory size is 3.5GB. The VMs running on Xen are listed in TABLE I. From the table, we can recognize that the guest only modifies the configuration of CONFIG_XEN_SCRUB_PAGES in Linux Kernel and resets it to "n", followed by recompiling and reinstalling the Linux Kernel. Furthermore, we carry out a detection experiment firstly using the function *rule_detector* in the sender module in the monitor to obtain the *upper* and *lower limits* of the *Adjustment Rule*. The result is that the *upper limit* is 83% and the *lower limit* is 32%.

A. Scenario I

In this scenario, the target is to steal some information from a guest using the monitor. The configurations of the VMs are displayed in TABLE II.

TABLE II. CONFIGURATION OF VMS IN SCENARIO I

Domain Name	Current Memory	Max Memory	Min Memory
guest	1GB	1.5GB	1GB
monitor	1GB	1GB	512MB
test	512MB	512MB	400MB

For example, a typical attack can be described as follows:

Company A has set up a Linux server on an IaaS cloud to store the source codes of a project. The VM belonging to the company is named as the guest. During the development of the project, the codes of the project are always added, modified, deleted and so on. Thus, the memory of the guest contains many physical pages with detail data of the codes. Thus, after one day's work, the memory usage of the guest may meet the requirements of the Adjustment Rule used in the VMM. Then, the VMM will recycle idle memory pages of the guest. Furthermore, an attacker can use the side channel to conduct an attack to steal the source codes of the project from the guest.

The process of the experiment is displayed as follows.

(1) Initialize all the VMs on Xen. The Domain will occupy about 350MB of memory at the beginning. (2) In the monitor, we adjust the monitor's memory size to 512MB through the self-adjustment module. At this time, the memory usage rate is below 83%. (3) In the guest, we write the transmission data into the memory via the program *data_reader* continually. In this way, we increase the memory usage rate up to 83% to trigger VMM's adjustment. Then, the memory size of the guest will increase until it reaches the Max Memory. (4) In the guest, we use the sender module to cancel all processes apart from systems'. Then the OS occupies only 378MB, at which time the usage rate falls below 32% and Domain0 will decrease the memory size of guest. While the guest memory remains stable, the total memory size of the guest is 1112MB and the memory usage rate is about 34%. Thus, the guest has released 424MB free pages into the shared memory pool. (5) In the monitor, we reset the size of memory to 1GB. Then we trigger the function *memory_dumper* to dump data into files to obtain the data transmitted from the guest. We have gotten 2682 files and then we insert all files into a database and use the SQL, "select * from FILE where CONTENT like '%begin>>%' ", to search for the transmission data.

B. Scenario II

In this scenario, the target is to transmit malicious data from monitor to guest. For example, this type of attack can be described as follows:

Company A owns a Linux server on an IaaS cloud to store some internal information about the company, while the information is updated day by day. An attacker wants to steal the internal information once a day. He has obtained the username and password of the server, but he does not want to obtain the information manually every day taking into account the changes of the password. Therefore he wants to install a malware on the server, but if he transmits the malware through the network, some security tools may check the data being transmitted. Thus, a transmission without attracting the VMM and other security tools' attentions is needed. Under the circumstances, the attacker can use this side channel to transmit the malware from his co-resident VM to the server.

There are several differences between this scenario and Scenario I we should pay attention to: (1) As the size of the file he transmits is 221KB, the attacker divides the malware into 56 pieces and transmit them several times between the two VMs. (2) Every time the attacker reads the small malware piece, he insert some special tags, "begin>>" and "<<end", at the beginning and end of the piece. In addition, he inserts a transmission serial number just following the beginning symbol. The process of transmitting is similar to Scenario I. (3) The attacker uses a tool, named as DD, to dump current memory into a file. Then, he extracts the data between "begin>>" and "<<end" to a file. Finally, he gets 56 files in total and the filename is the index of each file gotten from the bit following the beginning symbol "begin>>". (4) He reconstructs the transmission file by combining these ordered files. Then he can do what he wants with the file.

VI. PERFORMANCE EVALUATION

To evaluate the performance of our attack, we choose the *transmission success rate* as the target. We define the *success rate* as follows:

Definition 8. *Transmission success rate*: After a complete process of an attack, if the data transmitted from the victim is totally received and reconstructed, we treat this attack as one time of success. The transmission success rate can be calculated by equality (1):

$$\text{Success rate} = \text{success times} / \text{total times of attack} \qquad (1)$$

The *success rate* may be related with two aspects: (1) The percentage of data to be transmitted in memory. It is obvious that if the data to be transmitted occupies most of the memory, the success times will be bigger. (2) The *lower limit* of the Adjustment Rule. The lower limit of the rule controls the release of the memory, so if the lower limit is very small, all the VMs will not release memory pages to the shared memory pool again.

Thus, we carry out the evaluation from these two aspects. And the size of the data to be transmitted in our evaluation is smaller than a physical page size.

A. *The percentage of data to be transmitted in memory*

In this evaluation, the configuration and process are all the same as Scenario I and the process of evaluation is the same as Scenario I. We divide the evaluation into four groups: 10%, 30%, 50% and 70%. Each value represents the percentage of the memory with transmission data in total memory. We conduct the experiment 20 times with each group and count the success times. The result is displayed in Fig.3.

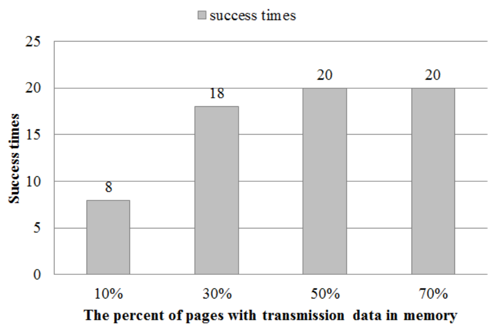

Fig. 3. Result of Evaluation A

The percent is related to the existence of data sources of our attack. According to Fig.3, it indicates that the more the percent is, the more times it succeeds. When the percent reaches a certain value, about 35% or so, the transmission will always succeed.

B. *The lower limit of the Adjustment Rule*

This aspect not only affects the Adjustment Rule but also affects the size of memory the guest can release. In this evaluation, the configuration and process are all the same as Scenario I. The percentage of data to be transmitted in memory we choose is 50%. We also divide the evaluation into four groups: 20%, 25%, 30% and 40%. Each value refers to the value of *the lower limit* of the *Adjustment Rule*. We conduct the experiment 20 times with each group and count the success times. The result is displayed in Fig.4.

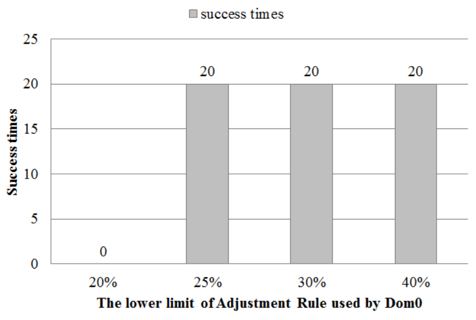

Fig. 4. Result of Evaluation B

From Fig.4, it suggests that if the transmission channel has been established, it will be likely to succeed. Therefore, this aspect controls the transmission of the attack.

VII. CONCLUSION

In this paper, we present a novel side channel, which is based on shared physical memory and utilizes the vulnerabilities of the balloon driver. This attack can directly obtain fine-gained information from another co-resident VM without the limitation of data size. Because of the wide use of shared memory and the balloon in current IaaS, this attack can achieve a high success rate.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (61100198/F0207).

REFERENCES

[1] Mell P, Grance T. Draft NIST working definition of cloud computing[J]. Referenced on, 2009, 53(6):50-50.

[2] Shi J, Song X, Chen H, et al. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring[C]// IEEE/IFIP International Conference on Dependable Systems & Networks Workshops. IEEE Computer Society, 2011:194 - 199.

[3] Erdal T, Onculer A. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds[C]// Acm Conference on Computer & Communications Security. 2009:199-212.

[4] Handel T G, Ii M T S. Hiding Data in the OSI Network Model.[J]. Proceedings of International Workshop on Information Hiding, 1996, 1174:23-38.

[5] Hu W M. Reducing timing channels with fuzzy time[C]// Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on. IEEE, 1991:8-20.

[6] Osvik D A, Shamir A, Tromer E. Cache Attacks and Countermeasures: The Case of AES[M]// Topics in Cryptology – CT-RSA 2006. Springer Berlin Heidelberg, 2006:1-20.

[7] Zhang Y, Juels A, Oprea A, et al. HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis[C]// 2011 IEEE Symposium on Security and Privacy. IEEE Computer Society, 2011:313-328.

[8] Sarda S, Kotecha R, Shetty P, et al. Secure Co-resident virtualization in multicore systems by VM pinning and page coloring[C]// Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on. IEEE, 2012:1-7.

[9] Zhang Y, Juels A, Reiter M K, et al. Cross-VM side channels and their use to extract private keys[C]// Acm Conference on Computer & Communications Security. ACM, 2012:305-316.

[10] Yarom Y, Falkner K. FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack[C]// Proceedings of the 23rd USENIX conference on Security Symposium. USENIX Association, 2014:719-732.

[11] Aviram A, Hu S, Ford B, et al. Determinating Timing Channels in Compute Clouds[C]// Proceedings of the 2010 ACM workshop on Cloud computing security workshop. ACM, 2010:103-108.

[12] Charles W. Practical Timing Side Channel Attacks against Kernel Space ASLR[C]// 2013 IEEE Symposium on Security and Privacy. IEEE, 2013:191-205.

[13] Jana S, Shmatikov V. Memento: Learning Secrets from Process Footprints[C]// Security and Privacy (SP), 2012 IEEE Symposium on. IEEE, 2012:143 - 157.

[14] Harnik D, Pinkas B, Shulman-Peleg A. Side channels in cloud services, the case of deduplication in[J]. IEEE Security & Privacy Magazine, 2011, 8(6):40-47.