논문 2014-09-25

Effect of ASLR on Memory Duplicate Ratio in Cache-based Virtual Machine Live Migration

Guangyong Piao, Youngsup Oh, Baegjae Sung, Chanik Park*

Abstract : Cache based live migration method utilizes a cache, which is accessible to both side (remote and local), to reduce the virtual machine migration time, by transferring only irredundant data. However, address space layout randomization (ASLR) is proved to reduce the memory duplicate ratio between targeted migration memory and the migration cache. In this pager, we analyzed the behavior of ASLR to find out how it changes the physical memory contents of virtual machines. We found that among six virtual memory regions, only the modification to stack influences the page-level memory duplicate ratio. Experiments showed that: (1) the ASLR does not shift the heap region in sub-page level; (2) the stack reduces the duplicate page size among VMs which performed input replay around 40MB, when ASLR was enabled; (3) the size of memory pages, which can be reconstructed from the fresh booted up state, also reduces by about 60MB by ASLR. With those observations, when applying cache-based migration method, we can omit the stack region. While for other five regions, even a coarse page-level redundancy data detecting method can figure out most of the duplicate memory contents.

Keywords : ASLR, Live migration, Memory, Duplication, Desktop as a service

Ⅰ. INTRODUCTION

Reducing the time of desktop virtual machine (VM) migration is very important in many scenarios, especially in cloud-service industry. In the case of desktop as a service (DaaS), desktop VM should be migrated efficiently to a data center near the customer or directly to the desktop the customer currently using, to improve the user experience (Fig.1) [1]. Besides, we also can perform server maintenance, load balancing, recovery and energy saving operations without service disruption [2, 3]. Ample research has tried to reduce the total migration time, because the contents of memory pages would become dirty when the migration process takes long time, and transferring those dirty pages would consume extra network and computing resources.

*Corresponding Author(cipark@postech.ac.kr)
Received: 3 Feb. 2014, Revised: 18 Mar. 2014, Accepted: 30 Apr. 2014.
G.Y. Piao, Y.S. Oh, B.J. Sung, C.I. Park: POSTECH
※ This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2011-0016972).

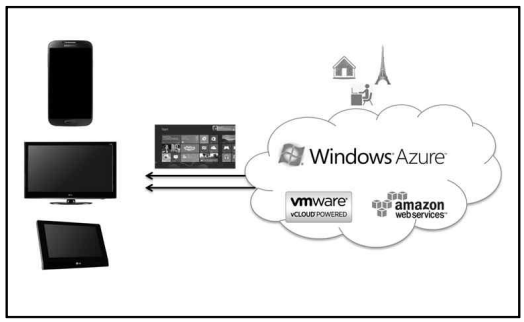


Fig. 1 Desktop as a service environment

ⓒ IEMEK J. Embed. Sys. Appl. 2014 Aug.: 9(4) 205-210
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2014.9.4.205

One of the efficient migration methods is reducing the VM migration time by utilizing a pre-defined migrating cache or freshly booted up memory state [4, 5]. Even the destination side does not have the preloaded cache, many memory pages among VMs running in destination side are duplicate with the source side VM's memory pages, so those duplicate pages can be used by migration cache. However, a security mechanism, Address Space Layout Randomization (ASLR) which is implemented in most operating systems (OS), modifies the virtual memory space layouts, and this modification reduces the memory duplicate ratio between two desktop VMs running identical OS by 10-20% [6, 7].

In this work, we analyzed the ASLR more deeply to learn how and why it influences the duplicate memory ratio among VMs. By analyzing ASLR code and performing experiments, we understood the behavior of the ASLR precisely, and propose possibilities to improve the migration speed.

The rest of the paper is organized as follows. Section II introduces the background of ASLR, Section III explains the detailed behavior of ASLR by implementing a probe code and code-level analysis. Section IV presents experimental results which prove the behavior of ASLR. Finally, Section V gives a brief conclusion and outlines our future work.

II. BACKGROUND

Cache-based live migration method is an efficient way to improve the performance of VM live migration [4, 5]. The cache could be a package of frequently used binary files, a memory state of previous memory or the memory pages of destination host. Comparing the page-level hash value between the cache and target migrating memory is the simplest and most efficient duplicate data detecting method. While ASLR is proved to reduce the data duplicate ratio among VMs running identical OS [6, 7].

The ASLR technique had been applied to almost every latest version of OS (for Windows from Vista [8], for Linux from kernel version 2.6.12) to improve the security level. In traditional design of OS, the layout of virtual address space is unique for every process. In such a scenario, an attacker can easily guess the virtual space layout, by implementing a simple malicious code. Return-to-libc attack [9] is a representative security attack which exploits the weakness of traditional OS. With the help of ASLR, virtual space layout of processes changed entirely, even when the same process runs for second time. We observed that ASLR mainly shifts some sensitive regions, and for non-sensitive regions like data and bss segments, the ASLR never rearrange the virtual space layout. There is only a special case about read-only code segment. The layout of code segment would be changed by ASLR, only when the process has independently applied PAX patch or adopted Position Independent Executable (PIE) technique. But there are few programs written with this special technique. So in this paper, we consider only three sensitive virtual memory regions, heap, mmap and stack.

III. EFFECT OF ASLR TO MEMORY SPACE

1. Modifications to Heap

Heap defines the virtual memory region which can be dynamically allocated or freed by a process using system calls. Because heap randomization had been adopted from later version of ASLR, in current version of Linux kernel, the heap randomization process can be controlled independently, unlike the space randomization processes in other memory regions. When ASLR is enabled, the kernel code executes an additional randomization function randomize_range(), with the original brk_start as the first parameter and brk_start + 0x2000000 as the second parameter. As a

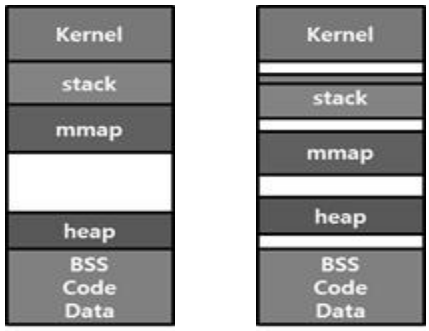
result, this function shifts the heap start point to a random space within 32MB virtual memory region with period of 4KB. Our monitor process showed that except this difference, the later malloc() functions perform as the same as ASLR disabled case.

2. Modifications to Mmap

Mmap region is used to map device files or shared libraries to virtual memory space of a process. In the virtual memory space layout, the address of the mmap base is just under the stack region. So in the Linux kernel, the mmap base is calculated by the maximum end point of the stack minus a random value (8 bits in 32-bit system or 24 bits in 64-bit system), and then aligned by page size. We traced the address of shared libraries from the maps information from proc directory, extracted the contents of shared libraries with ASLR enabled and disabled, and then compared those two contents. As for all mapped files, the contents between ASLR-enabled and ASLR-disabled cases were the same.

3. Modifications to Stack

The stack is the most sensitive region of process virtual memory space. The stack stores all of the local variables and return addresses of function, so only a small modification to this region could cause fatal memory leak and application collapse. The ASLR rearranges the stack layout much more complexly than the previous two regions. At first, the process copies the necessary argument and environment strings into the stack. This process is similar to the ASLR-disabled case. Then the randomize_stack_top function randomly sets the stack top address in an area of 8-MB virtual space, it then aligned by one page. Finally, arch_align_stack() function additionally decreases the stack start address with a random value in a range of 8KB aligned by 16 byte. With this mechanism the start point of stack has 1,048,576 possible



(a)ASLR disabled (b)ASLR enabled

Fig. 2 Address space layout of process with and without ASLR (after ASLR enabled the stack top is shifted by sub-page size)

positions, so attackers cannot guess the address space of stack region.

When ASLR turned off, the heap start address points to the end of BSS, the stack top starts from end of kernel space (0xC0000000), and mmap just followed by stack. All start points of the three memory regions are aligned by a page size (Fig. 2-a). Otherwise, in ASLR enabled case, all start points of mmap, heap and stack can be changed. For mmap and heap the start points are aligned by a page size, but for stack, it additionally shifted by a randomized sub-page level size (Fig. 2-b).

IV. EVALUATION

1. Environment

Our host machine has 12 Intel Xeon E5-2530 CPUs, 48 GB of RAM and a 4-TB disk, which runs KVM 1.2.0 on 64-bit 12.04 OS. Each guest had been assigned to one vCPU, 2 GB memory and 20GB virtual storage and runs 64 bit 12.04 OS. To simulate the real desktop environment, we applied a desktop workload, which randomly selected one of three tasks; these include editing libreoffice or viewing PDF files, playing local video streams, and searching random amount of web pages.

Table 1. page-level duplicate contents among fresh booted up VMs

	Total Size(MB)	Duplicate ratio
Disable ASLR	640	64.46%
ASLR without heap random	639	54.20%
ASLR with heap random	640	53.70%

2. Effect on Freshly Booted up Memory

In the first experiment, we booted up VM three times, once in each of the three scenarios: (1) disable ASLR; (2) enable ASLR without heap randomization; (3) enable ASLR with heap randomization. After fetching three sets of memory snapshots, we utilized the SHA-1 hash value of every page to calculate the page-level memory duplicate ratio in each case, so we obtained three comparisons (between 1 and 2, 1 and 3, 2 and 3) for each set.

We detected a slight difference between enable ASLR with heap randomization and without randomization case (Table 1). This means that the heap randomization does not influence the number of duplicate pages among VMs memory. However, the duplicate data ratio increases significantly when we disabled the ASLR. Based on the observation in section 2, we conclude that most of different pages among those 60 MB different pages are stacks.

3. Effect on Long-lived Memory

The previous experiment proved that the heap randomization does not influence the page-level duplicate ratio among VMs, so in this experiment we only consider the ASLR with heap randomization as well as ASLR disabled cases. For each set, after freshly booted up, we performed input replay to each VM, and calculated the page-level duplicate data among each set (the average value of three times).

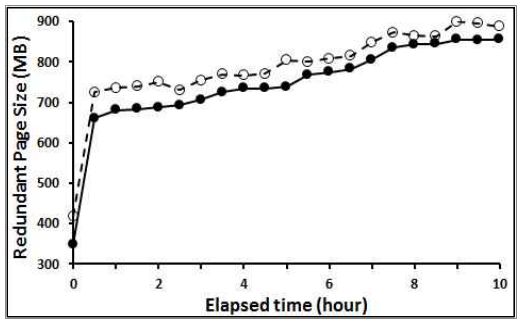


Fig. 3 Size of duplicate pages among identical VMs perform input replay. (_●_: ASLR with heap randomization, __O__: ASLR disabled)

In the first half hour, the size of duplicate pages increased rapidly (Fig. 3). This was caused by loading the code and data of workload into page cache. From the half hour to 8th hour, the duplicate page size of ASLR-disabled case always larger than the ASLR-enabled case around 60MB. Although, this gap was reduced to 20MB after 7.5 hours later, it increased again around 40MB. We consider the reasons for this phenomenon as follows. When the VM runs for long time, the allocated physical page would be a dirty one, which had been used before. Although in ASLR disabled case, the stack start from the start point of one page, if the stack does not overflow one page size or ended up in non-end position of a page, those stack pages could not be detected.

4. Duplicate Ratio with Fresh Booted-up State

Freshly booted up memory contains a considerable number of duplicate data with memory state of long lived VMs. So with page-level hash value comparison, memory pages which duplicate with fresh booted up state can be rebuilt independently on destination side. The purpose of this experiment is observing how ASLR influences that duplicate ratio.

In this experiment we booted up VM and saved the memory state after login, for each set. If the memory snapshots were taken

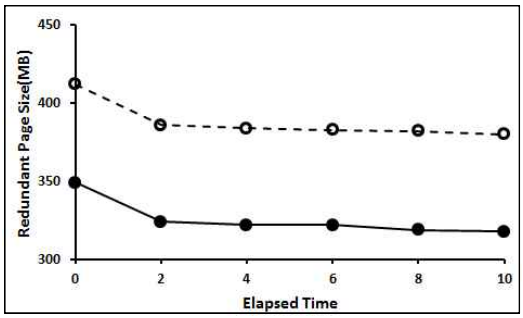


Fig. 4 Size of duplicate pages between long lived VM memory and freshly booted up memory state. (-●-: ASLR with heap randomization, --○--: ASLR disabled)

contiguously from that moment, the result would contain duplicate page-level contents of stack. To eliminate that effect, we booted up a new VM, and took snapshots.

We observed that the duplicate memory size between long lived VM and freshly booted up memory states decreases with time elapse. But from second hour, the size did not change too much (only reduced by 6MB during 8hours) in both case. Besides, the difference of duplicate memory size between ASLR enabled and disabled case also remains stable (Fig. 4).

V. CONCLUSION

We analyzed the behavior of ASLR, which influences the performance of cache-based live migration. To understand the behavior of ASLR correctly, we analyzed the ASLR code, and conducted experiments to prove how it influences the page-level memory duplicate ratio between two freshly booted-up virtual machines (VMs) running identical workloads. With code level analysis, we found that, among six virtual memory regions, only the stack decreased the memory page-level memory duplicate ratio. However, in most cases, the stack occupies only a small portion of the entire memory region. This means that ASLR does not much influence the page-level duplicate ratio among operating systems. This result is very valuable, because when live migrate a VM with a cache, we need not perform data deduplication to the stack region. But for the other five memory regions, even a coarse granularity duplicate data detecting method can correctly figure out most of the duplicate memory content.

References

[1] A. Kochut, H. Shaikh. "Desktop to cloud transformation planning," Proceedings of IEEE International Symposium on IPDPS, 2009.

[2] T. Das, P. Padala, V.N. Padmanabhan, R. Ramjee, K.G. Shin, "LiteGreen: Saving Energy in Networked Desktops Using Virtualization," Proceedings of USENIX Annual Technical Conference, 2010.

[3] J. Reich, M. Goraczko, A. Kansal, J. Padhye, "Sleepless in Seattle No Longer." Proceedings of USENIX Annual Technical Conference, 2010.

[4] R. Chandra, N. Zeldovich, C. Sapuntzakis, L.S. Lam, "The Collective: A Cache-Based Systems Management Architecture," Proceedings of USENIX Symposium on Networked Systems Design and Implementation, 2005.

[5] C. Clark, K. Fraser, S. Hand, J. Hanseny, E. July, C. Limpach, I. Pratt, A. Warfield, "Live Migration of Virtual Machines," Proceedings of USENIX Symposium of on Networked Systems Design and Implementation, 2005.

[6] A. Rai, R. Ramjee, A. Anand, "MiG: Efficient Migration of Desktop VMs using Semantic Compression," Proceesing of USENIX Annual Technical Conference, 2013.

[7] S. Barker, T. Wood, P. Shenoy, R. Sitaraman, "An Empirical Study of Memory Sharing in Virtual Machines," Proceesings of USENIX Annual Technical Conference, 2012.

[8] O. Whitehouse, “An Analysis of Address Space Layout Randomization on Windows Vista”, Symantec Advanced Threat Research, 2007.

[9] H. Shacham, M. Page, B. Pfaff, E.J. Goh, N. Modadugu, D. Boneh, "On the Effectiveness of Address-space Randomization," Proceedings of ACM Conference on Computer and Comm. Security, pp. 298-307, 2004.

Biographies

Guangyong Piao

2010: Received B.S. degree in department of computer science and technology from JiLin University, China.

Current: M.S. candidate in the department of computer science and engineering in Pohang University of Science and Technology.

Research Interests: include operating system, system virtualization.

Email: tmipyiong@postech.ac.kr

Youngsup Oh

2011: Received B.S. degree in department of computer science and engineering from Soongsil University.

Current: Ph.D. candidate in department of computer science and technology in Pohang University of Science and Technology.

Reseach Interests: include operating system, system virtualization, system security.

Email: youngsup@postech.ac.kr

Baegjae Sung

2006: Received a B.S. degree in department of computer science and engineering from Inha University.

2009: Received a M.S. degree in department of computer science and engineering from Pohang University of Science and Technology.

Current: Ph.D. candidate in Department of Computer Science and Engineering in Pohang University of Science and Technology.

Research Interests: include storage system, web service, operating system.

Email: jays@postech.ac.kr

Chanik Park

1983: Received a B.S. degree in department of electronics and engineering from seoul national university.

1985: Received a M.S. degree in department of electronics and electrical engineering from KAIST.

1988: Received a Ph.D. degree in deparment of electronics and electrical engineering from KAIST.

Current: Professor in the department of computer science and engineering in Pohang University of Science and Technology.

Research Interests: include operating system, storage system, embedded system, system virtualization, system security.

Email: cipark@postech.ac.kr