데스크 탑 가상 머신 라이브 마이그레이션에서 ASLR이 메모리간 중복성에 미치는 영향

(Effect of ASLR to Memory Deduplication Ratio in Desktop Virtual Machine Migration)

박 광 용*, 오 영 섭, 성 백 재, 박 찬 익
포항공과대학교
(Guang-Yong Piao, Young-Sup Oh, Baeg-Jae Sung, Chan-Ik Park)
(POSTECH)

Abstract : There are lots of cloud services available today. One of the services - desktop as a service, make us easily to access desktop environment via smart phone or tablet. We focus on desktop virtual machine live migration problem, in the case, when destination side has already possessed the previous memory status of source VM or when there are many desktop VMs running on destination side that theirs memory can be utilized to memory deduplication. In such scenario, If there is large number of redundant data, between current source side memory and destination side previous memory snapshot or currently running VMs' memory, for these redundant data, we only need to transmit the address and hash value of page, instead of entire content, and consequently we can reduce considerable migration time. However, most of current operating system adopted Address Space Layout Randomization technique, in order to strengthen the security capability, which influence the memory redundant ratio. In our work, we analyze the behavior of ASLR, observe how much degree it decrease the memory redundant ratio, and discuss about it.

Keywords : ASLR, live migration, memory, deduplication

I. Introduction

Reducing the desktop virtual machine (VM) migration time is an important issue 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, in order to increase user experience [1]. Besides, when servers are overloaded, desktop VMs need to be migrated from running servers to reserved back-up servers, to achieve loadbalance. Additionally, other work showed that with efficient migration, we also can reduce energy consumption [5].

One of the efficient migration methods is by utilizing pre-defined cache, which exists only in destination side [2,6] or both in source and destination [3]. Some of these work proved that with the effect of address space layout randomization (ASLR), memory deduplication ratio between two desktop VMs running identical OS decrease 10-20% [4]. However, they had not analyzed more specifically about the ASLR that why and how it happens.

Our purpose is to have a deeper

박광용, 오영섭, 성백재, 박찬익 : 포항공대
※ 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2011-0016972)

comprehension about ASLR by analyzing kernel level flow and implementing monitor process, to find out a solution to detect sub-page level redundant contents.

In this paper, Section2 introduces our analyzed result about ASLR, include how it modify the address of stack top, mmap base and heap base, as well as the change of contents. In section3 we will present our experiment results to illustrate the influence of ASLR to the page level redundant ratio. Finally in section4 we will give an overall summary and describe related and future work.

II. ASLR

Address Space Layout Randomization technique had been implemented into almost every latest version of operating systems (for windows from vista, for linux after kernel version 2.6.12), to improve the security level. In traditional design of operating systems, the layout of process virtual address space was unique for every process. In such scenario, one attacker can easily guess the virtual space layout, by implementing a simple malicious code, because memory space layouts of all processes are the same. Return-to-libc attack is a representative security attack which exploits the weakness of traditional OS. ASLR was designed to eliminate those kinds of security problems.

With the help of ASLR, virtual space layout of processes changed entirely, even when a same process runs for second time. In our experiment, we observed that ASLR mainly shift some sensitive regions. For non-sensitive regions, such as data and bss segment, ASLR never rearrange their space layout. One special case is about read only code segment. The layout of code segment would be changed by ASLR only when the process is independently applied PAX patch or adopted position independent executable (PIE) technique, in other words the “Full ASLR”. However, we had not found out how Full ASLR works, since few applications adopted this mechanism. In this part, I will explain what happened to remaining space layout, the heap, mmbase and stack, when ASLR is enabled.

1. Modifications to heap

Heap defined the virtual memory region that can be dynamically allocated or freed by a process via system call. Since heap randomization had been adopted from later version of ASLR, in current version of linux kernel, unlike other memory regions, it can be controlled independently. When ASLR enabled, the kernel code would execute 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 set the heap start point to a random space within 32MB virtual memory region with period of 4KB. With the help of our monitor process we also found that, except this difference, later malloc() function performs as the same as ASLR disabled case.

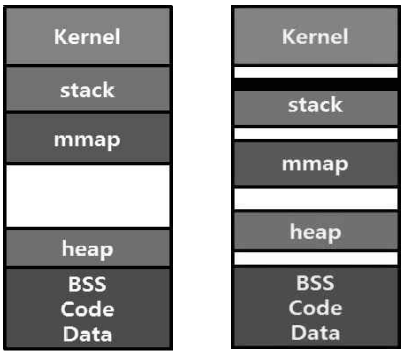
2. Modifications to mmap

Mmap region is used to map device files or shared libraries to virtual memory space of a process. From the virtual memory space layout, we can find that the address of mmap base is just under the stack region. So in the linux kernel, the mmap base is calculated by the maximum end point of stack minus a random value (8 bits in 32-bit system or 24bits in 64-bit system), and then page align it. We traced the address of shared libraries from the maps information from proc folder, extract the contents of shared libraries with ASLR enabled and disabled, and then compared the contents. As we expected, for all of the mapped files, the contents between ASLR enabled and ASLR disables case were the same.

3. Modifications to Stack

Stack is the most sensitive region of

process virtual memory space. It stores all of the local variable and function return address, since a small modification to this region would cause fatal memory leak and application collapse. So ASLR rearranges the stack layout much more complex than the previous two regions. At first, the process copies the necessary argument and environment strings into the stack, this process is similar with ASLR disabled case. After that the function randomize_stack_top randomly set the stack top address in an area of 8MB virtual space, then aligned it by one page. Finally, arch_align_stack() function additionally decrease the stack start address in a range of 8KB, aligned by 16 byte. With this mechanism start point of stack has 1048576 possible positions. So it is mealy impossible for attackers to guess the stack address space.



(a) ASLR disabled (b)ASLR enabled

Fig. 1. Address Space layout of process with and without ASLR

Figure1. shows the process memory space layout in two different cases. Figure1.a is the case when we turn off the ASLR. Here the heap start address is equal with end address of BSS, stack top is started from end of kernel space (0xC0000000), and mmap is just followed by stack, and all of the contents are aligned by one page. In Figure1.b, although mmap and heap change the space layout, both of them are aligned by a page. However, when we look at stack, all contents are shifted by black amount of bytes, in sub-page level.

In the next section, we will discuss about the effect of ASLR to page level memory redundant ratio, with some experiments.

Ⅲ. Evaluation

Our Host machine has 12 Intel Xeon E5-2530 CPUs, 48GB RAM and 4TB disk, which runs KVM 1.2.0 on 64bit Ubuntu 12.04 OS. Each guest had been assigned to 1 vCPU, 2GB memory and 20GB virtual storage runs ubuntu12.04 64 bit. In order to simulate the real desktop environment, we made our own desktop workload, which randomly select one of three tasks. They include simple document operation, editing libreoffice or viewing PDF files, play local or online video streams and searching random amount of web pages.

1. Effect to fresh booted up memory

In the first experiment, we booted up VM for three times in each three scenarios, disable ASLR, enable ASLR without heap randomization and enable ASLR with heap randomization. After fetched three sets of memory snapshots, we utilized the SHA-1 hash value of every page to calculate the page level redundant ratio in each case. So we can get three results (between 1 and 2, 1 and 3, 2 and 3) for each set. The data presented in Table1 is the average value of our results.

Table1. page-level identical contents among fresh booted up VMs

	Total Size(MB)	Redundant ratio
Disable ASLR	603	64.46%
ASLR without heap random	603	54.20%
ASLR with heap random	602	53.70%

As we expected, there is slight difference between enable ASLR with heap randomization and without randomization case. It means that

the heap randomization does not influence the identical pages among VMs memory. However, the redundant data ratio increases significantly when we disable the ASLR. Based on the observation in section2, we could estimate that most of different pages among 60MB pages are stacks.

2. Effect to long lived VM memory

Previous experiment proved that heap randomization does not reduce the number of identical pages, since in this experiment we only consider disable ASLR and enable ASLR with heap randomization cases. At this time, we run our mixed desktop workload for 12 hours and choose a memory snapshot, which runs less than 3 hours and include large percentage of dirty pages, as a standard snapshot. After that we calculate the de-duplication ratio between this snapshot and consequent snapshots.

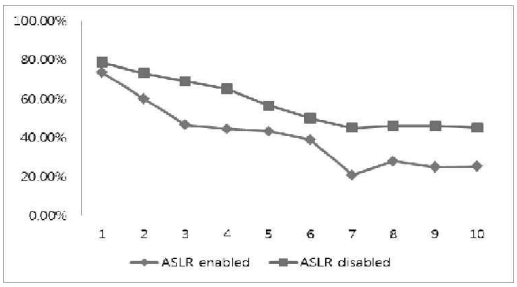


Figure2. Redundant ratio among VMs memory with time elapse

In the Figure2 we can observe that with the time elapse, ASLR disabled case has 20% more redundant data than ASLR enabled case, and in the 10th hour the redundant data size difference is about 300MB.

IV. Conclusions

In sum, we have analyzed behavior of ASLR more specifically and showed how it modifies the layout of process virtual space. Our experiment results shows that, in order to extract more redundant data from previous memory snapshot, we should do sub-page level data comparison, which would generate significant overhead. However, we understood that large proportion of subpage-level identical contents is belonged to stack. Despite ASLR another technique preload also influence the redundant ratio between VM memories. Our future work is analyzing behavior of preload, and find out an efficient sub-page level duplicate data detection method to improve the migration performance.

V. Reference

[1] A. Kochut and H. Shaikh. "Desktop to cloud transformation planning. " In IEEE IPDPS, May 2009.

[2] R. Chandra et al., "The Collective: A Cache-Based Systems Management Architecture," Proc. Symp. Network Systems Design and Implementation, USENIX, 2005, to appear.

[3]. Anshul Rai, Ramachandran Ramjee, Ashok Anand. "MiG: Efficient Migration of Desktop VMs using Semantic Compression", In USENIX ATC2013.

[4]. Sean Barker, Timothy Wood, Prashant Shenoy, Ramesh
Sitaraman. "An Empirical Study of Memory Sharing in
Virtual Machines", In USENIX ATC, 2012

[5]. T Das, P Padala, VN Padmanabhan, R Ramjee, KG. Shin. "LiteGreen: Saving Energy in Networked Desktops Using Virtualization." In USENIX ATC, 2010.

[6] C. Clark, K. Fraser, S. Hand, J. Hanseny,E. July, C. Limpach, I. Pratt, A. Warfield, "Live Migration of Virtual Machines", In NSDI, 2005.

[7]. J Reich, M Goraczko, A Kansal, J Padhye. "Sleepless in Seattle No Longer." In USENIX ATC, 2010.