# Effects of Memory Randomization, Sanitization and Page Cache on Memory Deduplication

Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, Cyrille Artho
National Institute of Advanced Industrial Science and Technology
{k.suzaki | k-iijima | yagi-toshiki | c.artho}@aist.go.jp

## ABSTRACT

Memory deduplication merges same-content memory pages and reduces the consumption of physical memory. It is a desirable feature for virtual machines on IaaS (Infrastructure as a Service) type cloud computing, because IaaS hosts many guest OSes which are expected to include many identical memory pages. However, some security capabilities of the guest OS modify memory contents for each execution (e.g., ASLR: Address Space Layout Randomization) or uniformly set inactive memory contents to zero (Memory Sanitization). These capabilities have positive or negative impacts on memory deduplication. The severity of the impact depends on the size of the memory (i.e., the number of virtual machines) and update frequency, because most memory deduplications scan and merge the memory at runtime at regular intervals. We evaluated the effects of ASLR, Memory Sanitization, and their related security capabilities (Position Independent Executables, page cache flushing, and dirty page flushing) of the Linux guest operating system on the KVM virtual machine with KSM (Kernel Samepage Merging) memory deduplication. The results indicate ASLR increases physical memory consumption by more than 18% on 4 virtual machines with memory deduplication. The combination of memory sanitization and page cache flushing reduce physical memory consumption about 20 - 35% at a stable state.

## 1. INTRODUCTION

A conventional operating system expects to own and manage the whole available memory itself. This assumption is not true in a virtual machine environment. The memory owned by the guest operating system, which is called guest physical memory, is managed by a virtual machine monitor and shared by other virtual machines. Memory sharing reduces consumption of real physical memory and allows accepting more virtual machines, which is preferred on IaaS (Infrastructure as a Service) cloud computing.

Memory deduplication is a popular technique to merge same-content memory pages between virtual machines. Most virtual machine monitors include this technique [1,2,7,9,16]. To increase the benefit of memory deduplication, the guest operating system should know of the existence of memory deduplication and increase the number of same-content memory pages. Unfortunately, such a collaboration is not as widely accepted as other techniques (e.g., memory ballooning). Therefore, some techniques to modify memory contents affect the performance of memory deduplication.

On the other hand, modern operating systems have security capabilities that modify memory contents dynamically. Address Space Layout Randomization (ASLR)[6,10,12,15,17] changes memory contents for each execution. Memory sanitization[8] sets memory contents to zero. Page cache and dirty pages changes life time of data on memory and are targeted for computer forensics and side channel attacks[13]. These capabilities may have a positive or negative impact on memory deduplication. However, these security capabilities are not taken into account by most memory deduplication systems.

Address Space Layout Randomization (ASLR)[6,10,12,15,17], which is a kind of memory randomization technique, changes the base address of the stack, heap, shared libraries, and the binary. ASLR can protect a system against malware that exploits a fixed address in order to overwrite or reuse code. ASLR is useful but requires changing page contents for each execution. It also requires ELF binaries to be Position Independent Executables (PIE)[10,15], which consume more memory pages. Therefore, ASLR will affect memory contents and thus also memory deduplication.

Memory sanitization sets memory contents to zero, when pages are released from an application or the kernel and become inactive. Sanitization is designed to prevent information leaks from inactive memory pages, but it also helps to increase the effectiveness of memory deduplication. However, memory sanitization on Linux[8] requires all memory managed by the kernel to be initialized at boot time. This may constitute an overhead for memory deduplication.

Page cache and dirty pages are independent of the management of memory sanitization. They are designed to increase I/O performance at the cost of consuming more memory. Furthermore, the page cache and dirty pages may include sensitive information. From the view of security they should be cleared when they are not used anymore. If the memory for page cache and dirty pages is flushed and sanitized in a timely manner, memory deduplication is able to accept more virtual machines. This requires a tradeoff between keeping page cache and dirty pages for the I/O performance of each guest OS, and offering the pages to other virtual machines.

In this paper, we investigate the quantitative impact of these security capabilities on traditional memory deduplication (periodic memory-scan) on a real machine. We evaluate the effects of memory randomization, sanitization, page cache, and dirty page of the Linux guest OS on the KVM virtual machine with KSM (Kernel Samepage Merging) [1] memory deduplication. We evaluate the boot process of guest OSes, because this requires many resources. The results show that


*EUROSEC'12* April 10, 2012, Bern, Switzerland.

- ✓ ASLR increases physical memory consumption by more than 18% on 4 virtual machines, which is more than anticipated.
- ✓ Memory sanitization has to initialize all memory assigned to the guest OS, but the CPU overhead for memory deduplication is low because memory merging is postponed at heavy load. At a stable state the combination of memory sanitization and deduplication reduces physical memory consumption by about 10%.
- ✓ Page cache flushing reduces physical memory consumption by about 10% with memory deduplication. The combination of memory sanitization and page cache flushing yields a 20 - 35% reduction. However, the boot time overhead increases by 30% because of reloading flushed pages.
- ✓ KSM is implemented as a background job and does not affect the performance of guest OS severely. The boot time of the guest OS is delayed by page cache flushing and PIE, because page cache flushing and PIE require more I/O throughput than normal.

# 2. MEMORY DEDUPLICATION

Memory deduplication is a technique to merge same-content memory pages and reduces the consumption of physical memory. It is popular on virtual machines, because the memory images of virtual machines include many same-content pages, especially when the same guest OS runs on several virtual machines.

## 2.1 Taxonomy

Current virtual machine monitors are equipped with memory deduplication. The techniques are divided into two types; content-aware deduplication and periodic memory-scan deduplication.

Content-aware deduplication is used on Disco's Transparent Page Sharing (TPS) [2] and Satori [9] on Xen. TPS reads page data from a special copy-on-write disk and checks whether the same page data is already present in main memory. If a pages matches, TPS creates a shared mapping to the existing page. Satori has a similar policy for duplicate detection, but does not use a special copy-on-write disk. Satori is implemented as para-virtualization on the Xen hypervisor. Content-aware deduplication is useful but cannot treat dynamically created pages, for example when using memory sanitization.

Periodic memory-scan deduplication is used in Content-Based Page Sharing of VMware ESX [16], Difference Engine [7] of Xen, and KSM (Kernel Samepage Merging) [1] of the Linux kernel. Content-Based Page Sharing scans the VM's memory periodically and records fingerprints of each page. When same fingerprints are found, they are merged as an identical page. Difference Engine has not only memory-scan deduplication, but also patching and compressing. When almost identical pages are found, the small difference is taken as a patch. Compression is used when a page has not been active for a long time. KSM (Kernel Samepage Merging) included from Linux kernel 2.6.32 is a general type of memory deduplication. It was developed for its virtual machine (KVM), but it is not limited to a virtual machine. In this paper we use KSM for memory deduplication.

## 2.2 KSM (Kernel Samepage Merging)

Most implementations of periodic memory-scan deduplication use the hash value of a page to check the similarity between pages. The initial implementation of KSM used the same technique, but it was re-implemented with another method to avoid a patent problem. KSM uses a simple 32-bit checksum for rough scanning. After scanning, the exact similarity is determined by a full comparison with memcmp().

KSM manages memory pages with two red-black trees; one is for candidate pages of deduplication (called unstable tree) and the other one is for duplicated pages (called stable tree). Pages are identified by their 32-bit checksum in the trees. When the same content of a candidate page is found in the stable tree, the candidate page is merged with the stable tree. When the same content of a candidate page is found in the unstable tree, the two pages (candidate page and the page in unstable tree) move to the stable tree.

Pages are scanned at an interval, which is defined at /sys/kernel/mm/ksm/sleep_millisecs. The default period is 20 msec. The time is the interval of the kernel daemon called "ksmd". The maximum number of pages that ksmd treats at one period, is limited (the default is 25% of the available memory). Therefore, not all pages are scanned at a time.

KSM is not applied on all processes automatically. An application which uses KSM, has to declare the memory region to be deduplicated by madvise(). It means the source code has to be rewritten to use KSM. Virtual machine monitor KVM is already customized to use KSM. Memory deduplication is automatically applied on all processes of the guest OS on KVM.

# 3. OS SECURITY FUNCTIONS

Modern operating systems have security capabilities that modify memory contents dynamically. This section introduces these capabilities.

## 3.1 Memory Randomization

Most attackers develop malware on the same environment as the one of the target machine. This allows attackers to know the address layout of a target application. A typical buffer overflow attack works by overwriting the return address on the stack and heap. In order to make this attack work, the attacker must know the exact address at which to overwrite memory. A return-to-libc attack also has to locate the exact code to be executed.

The exact address is easy to get when an application uses the same address mapping of stack and heap. On traditional Linux, the memory layout is fixed: ELF binary code is mapped from 0x08048000, the heap starts from at the end of the ELF binary, shared libraries are mapped from 0x40000000, and the stack starts from 0xBFFFF000. The attacker easily knows the target address. To prevent these attacks, memory randomization was developed.

Address space layout randomization (ASLR) is a popular memory randomization technique[6,10,12,15,17], which involves randomly arranging the base address of key components. It includes the base of the executable, libraries, heap, and stack. ASLR changes the addresses for each process and makes it difficult for an attacker to predict target addresses.

Early ASLR for Linux was implemented as a part of security extensions PaX [12] and Exec Shield [15]. The mainline Linux kernel has enabled ASLR since kernel version 2.6.12 (released June 2005). The kernel gives randomized offsets to addresses of the stack, heap, shared library, and binary code. Most current Linux distributions enable ASLR in the Linux kernel by default.

Even if the kernel supports ASLR, a normal ELF binary is mapped from the preselected address (0x08048000), because the code is compiled to be loaded at that address. Red Hat recognized this weak point and contributed to the GNU Compiler Collection

tool chain (compiler and linker), implementing a technique called Position Independent Executable (PIE). Gentoo Linux has a derivation called "Hardened Gentoo Linux" which builds PIE binaries as default. With Gentoo, we can compare the difference between a normal installation and a hardened one.

## 3.2 Memory Sanitization

Clearing sensitive data after use is commonly accepted practice for secure programming. However, this practice is not widely used in commodity applications. Sensitive data is often scattered through user and kernel memory, and left there for long periods, even if the memory is not used by applications or the kernel [3]. According to the paper [3], sensitive data was reported to be remaining even after 30 seconds without power on IBM ThinkPAD T30 laptops. This feature makes systems vulnerable and increases the risk of exposing the data.

Chow proposed secure deallocation to erase sensitive data [4]. In order to implement secure deallocation on Linux, unconditional page sanitization was developed [8]. That code is largely based on the memory sanitization feature in the PaX project[12]. It writes zero on a memory page when the memory page becomes inactive, i.e., when it is released to the system after use. This avoids leaking sensitive information on inactive memory pages. The software was offered as patch of Linux kernel in 2009, but has not been included in the mainline Linux kernel yet.

Not only at the time of a memory page release, but also at boot time, unconditional page sanitization writes zero for initialization. Namely, this requires the allocation of the physical memory for the guest OS at boot time. This seems to be inefficient, but it does not require taint tracing like secure deallocation. Performance is examined in the evaluation section.

## 3.3 Page Cache

Page cache is a transparent disk-backed buffer of pages in main memory. It reduces the I/O overhead when the same disk pages are accessed repeatedly. However, the page cache mechanism on a normal OS consumes all available memory. It assumes the whole memory is owned by the OS and is unaware of sharing memory with others. Furthermore, the page cache may potentially expose sensitive data. When sensitive data is read from disk, it persists in the page cache and is not purged even after a process terminates [3].

The page cache is not covered by Linux unconditional page sanitization, because it is owned by the kernel. In order to release the page cache, newer Linux kernels (versions 2.6.16 and later) have a mechanism call "DropCache". When 1 is written at /proc/sys/vm/drop_caches, all page cache pages are released. It means that DropCache requires an explicit write operation. Usually, "cron" is used to flush page cache at certain intervals. Even if the page cache is released from kernel, its contents are not changed. The only effect of DropCache is that the pages are recognized as inactive pages. In order to set the released pages to zero, Linux unconditional page sanitization has to be enabled.

Even if sanitization is not enabled, DropCache changes the behavior of memory, because the released memory pages are reused by other applications instead of requiring extra memory. This means that DropCache reduces the consumption of memory and improves the total performance when a VM's memory is allocated on demand by the virtual machine monitor. The disadvantage of DropCache is the increase of I/O: a flushed page has to be re-loaded from disk when the same contents are accessed again.

## 3.4 Dirty Page

When data of the page cache is updated by an application, its pages have to be written back to disk in order to keep the file consistent. However, data is not flushed to disk instantly when a page is updated, because the page may be updated again. A page containing updated data in memory is called a "dirty Page". This mechanism reduces the number of disk I/O operations.

However, dirty pages may potentially expose sensitive data. They also consume memory. In order to flush data, the sync() system call is used. Even if sync() is issued, a dirty page is not released and zero-cleared. Pages are sanitized when DropCache is issued and memory sanitization is enabled.

# 4. EVALUATION

The impacts of security capabilities (ASLR, memory sanitization, DropCache, and sync for dirty page flushing) on memory deduplication were measured on a real environment.

We made two disk images of the guest OS with Gentoo Linux (1.12.13, kernel 2.6.31) on a 32GB virtual disk (31GB ext3, 1GB swap). One instance of Gentoo Linux was built as normal. Another instance of Gentoo Linux was built using PIE ELF binaries. Both of them could enable or disable ASLR, DropCache, sync, and unconditional memory sanitization, respectively. The disk images were run on KVM with 512MB memory on Ubuntu 9.10 (kernel: vanilla-2.6.32.1), on a 4-core Core2 machine. KVM was used with KSM enabled, and booted with 1, or 2, or 4 virtual machines at the same time. The measurements of booting show the effect of the security capabilities on KSM, especially the initialization of unconditional memory sanitization on Linux.

KSM shows four types of statistical information (sharing, shared, unshared, volatile) at /proc/meminfo/ every second. Sharing indicates the pages to be merged by KSM, which are then eliminated from physical memory. Shared indicates the pages merged by KSM, with one original page existing in physical memory. Unshared indicates candidate memory pages for shared, where no buddy page has been found yet, i.e., unshared indicates unique pages. Volatile indicates the pages that change frequently and are not target of deduplication, i.e., volatile also indicates unique pages. We analyzed the behavior of security capabilities on KSM using these statistics.

## 4.1 ASLR on KSM

ASLR changes the base address of stack, heap, shared libraries and ELF binaries, which increases the number of pages with different contents. Figure 1 shows the memory usage of KSM with PIE Gentoo at a stable state after booting 1, 2, and 4 VMs, respectively. They indicate the same tendencies that we got on normal Gentoo. The left half of the results, (1) – (4), shows the cases with ASLR, while the right of half results, (5) – (8), shows the cases without ASLR. On any results, total (volatile+unshared+sharing) memory with ASLR is larger than without ASLR. The number of shared memory pages are almost the same, but using ASLR increases the number of unshared pages. The results indicate that ASLR increases pages with different contents. The increases were higher than 18% on 4 virtual machines.

Table 1 shows the statistics of KSM on 4 VMs with and without ASLR, DropCache and Sanitization on PIE Gentoo. The upper four results are with ASLR and the lower four results are without ASLR. At a stable state, sharing memory usage with ASLR is lower than without ASLR, and unshared+volatile

**Table 1. Statistics of KSM on 4 Virtual Machines with/without ASLR, DropCache and Sanitization on PIE Gentoo. On Peak Memory, Virtual is the maximum memory requested by the guest OS, and Physical is the maximum memory offered by KVM with KSM. The stable state shows the memory status after booting. Total Physical indicates used memory, Sharing indicates deduplicated memory, and Unshared+Volatile indicates non-deduplicated memory. The guest OS boot time is the time in which X Window boots on VMs.**

| ASLR | DropCache | Sanitize | Peak Mem (MB) Virtual/Physical | Stable State Total Physical Mem (MB) | Stable State Sharing (MB (%)) | Stable State Unshared+ Volatile (MB (%)) | Guest OS Boot Time (sec) |
|---|---|---|---|---|---|---|---|
| ○ | | | 574/458 | 234.9 | 106.4(45.3) | 128.5(54.7) | 62 |
| ○ | ○ | | 431/332 | 206.9 | 70.7(34.1) | 136.3(65.9) | 83 |
| ○ | | ○ | 2063/1661 | 204.6 | 82.1(40.1) | 122.5(59.9) | 61 |
| ○ | ○ | ○ | 2063/1616 | 186.5 | 39.4(21.1) | 147.1(78.9) | 83 |
| | | | 574/455 | 199.0 | 120.1(60.4) | 78.9(39.6) | 62 |
| | ○ | | 429/316 | 169.5 | 83.1(49.0) | 86.5(51.0) | 82 |
| | | ○ | 2063/1661 | 171.2 | 94.0(54.9) | 77.2(45.1) | 62 |
| | ○ | ○ | 2063/1161 | 129.9 | 50.4(38.8) | 79.5(61.2) | 85 |



memory usage with ASLR is higher than without. This indicates that ASLR decreases deduplication and increases unique pages. It means that ASLR reduces opportunities for memory deduplication.

The increases of non-deduplicated pages (unshared+volatile) between 2 – 4 VMs with ASLR were higher than without ASLR. This result is to be expected, but will have a significant impact when a large number of VMs is run.

## 4.2 Position Independent Executable on KSM

Gentoo Linux has 1,469 ELF binary files in /bin, /sbin, /usr/bin, and /usr/sbin. The total volume of the original ELF binaries is 88.4M; for PIE ELF binaries, it is 94.6M (7% more). The biggest change is in "pampop9", which increases from 5,396B to 9,440B (75% more). The smallest change is in "wall", which decreases from 9,624B to 9,392B (2% less).

The PIE image increases the consumption of memory at any security capability setting. However, the increases were less than 10% on 4 VMs in stable states. The effects of other security capabilities are small, and experiments using normal Gentoo show the same tendencies. The subsequent sections discuss the results of PIE Gentoo.

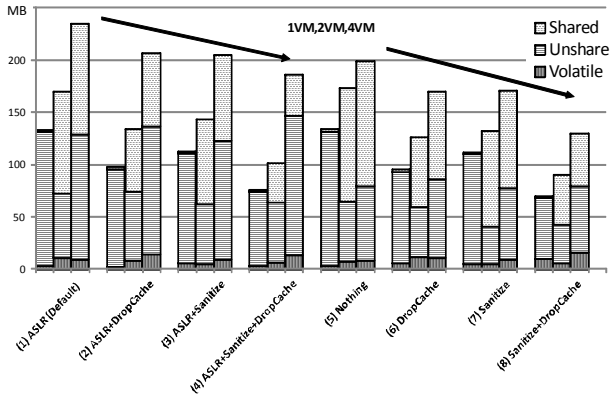


**Figure 1. Memory usage of KSM with PIE Gentoo at stable states after booting 1, 2, and 4 VMs, respectively. The Y-axis indicates consumed memory (MB).**

## 4.3 DropCache on KSM

On experiments with DropCache enabled, DropCache was issued every second. DropCache takes 10 - 20 microseconds, thus it does not affect performance severely. However, the boot time with DropCache increases by about 20 seconds (30%); see Table 1, last column. This change is caused by re-reading data from disk. Without DropCache, the boot procedure reads 65MB of data from disk. DropCache increases this to 99MB. Table 2 summarizes the results.

When considering memory usage, DropCache is effective, because the released memory is reused for other processes. DropCache reduces the consumption of physical guest memory on each VM, thus reducing total real physical memory. This effect is visible in Figure 2, which shows the maximum guest physical

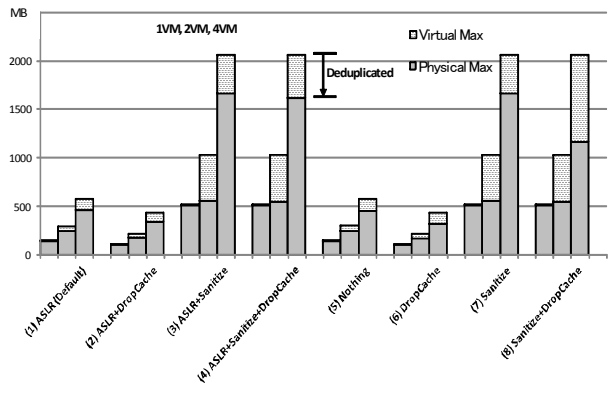


**Figure 2. The maximum guest physical memory requested by PIE Gentoo and maximum real physical memory offered by KVM with KSM on 1, 2, and 4 VMs, respectively. The Y-axis indicates consumed memory (MB).**

memory requested by PIE Gentoo and maximum real physical memory offered by KVM with KSM on 1, 2, and 4 VMs, respectively. The results of DropCache, (2) and (6), show the smallest physical memory usage, regardless of ASLR being enabled or not. At a stable state, DropCache can reduce memory consumption by about 10% compared to the case without memory sanitization (Table 1, Total Physical Mem). The results indicate that DropCache has advantages on IaaS type cloud computing, even if sanitization is not active. When sanitization is active, the memory released by DropCache is set to zero and deduplicated.

**Table 2. Disk read data on PIE Gentoo**

| | Disk Read (MB) |
|---|---|
| Normal | 65 |
| DropCache | 99 |
| Sanitize | 65 |

## 4.4 Sync on KSM

On experiments with sync enabled, sync was issued every second. We measured the effects with and without DropCache and sanitization. The results show no significant difference in any cases. We guess this result comes from out choice to measure boot time. If we used an application updating data frequently, such as a data base, sync might have an effect. Such experiments constitute future work.

## 4.5 Memory Sanitization on KSM

Unconditional memory sanitization in Linux sets memory contents to zero on all memory pages at the beginning of booting. The whole memory is assigned at that time, even if it is not used by applications or the kernel. KSM is a periodic memory-scan deduplication and cannot catch up with this large memory assignment. This is confirmed by the case when several guest OSes sanitize the memory in Figure 3, which shows a trace of memory deduplication with and without memory sanitization. At peak memory usage, memory merging is postponed and pages are treated as volatile memory. Furthermore, the effect on the guest OS is small, exemplified by minor differences in boot time.

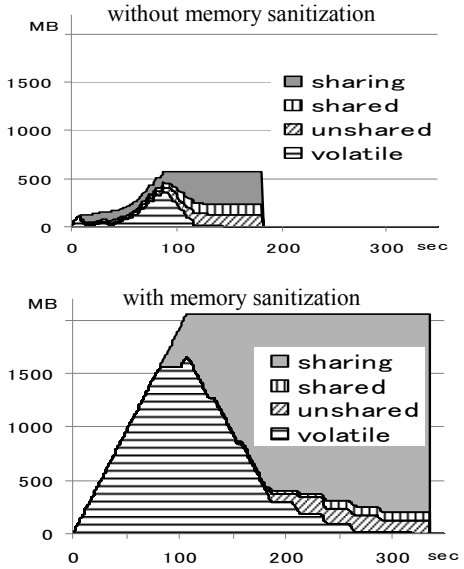


**Figure 3. Trace of Memory Deduplication on ASLR of PIE Gentoo with ASLR on 4 VMs. The upper graph shows the case without sanitization, and the lower graph shows the case with sanitization. The X-axis indicates time (sec) and the Y-axis indicates consumed memory (MB).**

However, memory sanitization causes another problem. About 1600MB physical memory is required by the memory spike caused by the sanitization on 4 VMs, though only 500MB physical memory is required for the case without sanitization. If the VMM can recognize memory sanitization of the Guest OS, the extra physical memory is not required.

At a stable state, memory sanitization can reduce the consumption of memory by about 10%, compared to without memory sanitization (Table 1, Total Physical Mem). The combination of DropCache and sanitization shows the best memory performance (20 - 35% reduction) at a stable state regardless of ASLR being enabled, as shown in Figure 1, (4) and (8). The drawback of this combination is the time overhead caused by DropCache and the large amount of guest physical memory required by sanitization.

## 4.6 Ratio of Deduplication at a Stable State

Figure 1 shows the ratio of memory deduplication at a stable state, which has the smallest number of volatile pages after booting the guest OS.

The ratio of shared pages is almost the same using any security capabilities on 2 and 4 VMs, except in the case that combines DropCache and Sanitization. This result comes from flushing the page cache, memory reuse, and the deduplication of sanitized pages. When considering memory usage, this combination is the best one.

The number of shared pages on one VM is small, because a single VM does not have many identical pages by itself. Unshared (unique) pages, on one VM turn into shared on 2 and 4 VMs (see Figure 2). It also shows that the number of pages merged by deduplication do not increase beyond 2 VMs. However, physical memory usage increases beyond 2 VMs, which is caused by unshared memory, indicating unique memory pages. This increase comes from different memory pages that were created on each guest OS.

## 4.7 Boot time of the Guest OS

The boot time of the guest OS (Gentoo) is affected by PIE and DropCache. The other security capabilities do not affect the boot time significantly. The number of VMs also has little effect, because the experiments were executed such that the number of VMs was not more than the number of physical cores.

Table 3 summarizes the effects. Time was measured by a physical clock, and the results include the overhead of the VMM. We expected that the effects come from I/O, because DropCache increases the I/O from 65MB to 99MB, and PIE increases the size of ELF binaries (the average increase being 7%). The time difference is also related to the increase of I/O.

The results are interesting, because memory behavior does not affect boot time severely, even if memory sanitization causes extra load on the guest OS and KSM. The most severe impact comes from extra I/O.

**Table 3. The boot time guest OS**

| | Normal Gentoo (seconds) | PIE Gentoo (seconds) |
|---|---|---|
| No DropCache | 57—58 | 61—62 |
| DropCache | 72—79 | 82—85 |

## 5. RELATED WORK

Memory deduplication can offset overheads introduced by increasing the security of an OS. SLINKY[5] shows that memory deduplication reduces extra memory usage caused by statically linked shared libraries, which protect against the vulnerabilities of dynamic linking. In our past work [14], a self-contained binary translator is used to integrate shared libraries instead of static linking; extra memory usage is also reduced by memory deduplication. These results show that redundant memory contents caused by security-strengthened OSes can be reduced by memory deduplication.

Memory deduplication is vulnerable to side channel attacks. The vulnerability is caused by different write-access times between deduplicated and non-deduplicated memory pages, because a deduplicated memory page has to be re-created by Copy-On-Write when the page is updated. The vulnerability is used to disclose contents on other VMs[13] and fingerprint the Guest OS[11].

## 6. CONCLUSIONS

This paper shows how memory deduplication is affected by the security capabilities of the guest OS. Different security capabilities (ASLR: address space layout randomization, position independent executables, memory sanitization, page cache flushing, dirty page flushing) are equipped on a guest OS (Gentoo Linux), and their effects are evaluated on the KVM virtual machine with KSM (Kernel Samepage Merging) memory deduplication, which is a kind of periodic memory-scan deduplication.

The security capabilities have a positive or negative impact on memory deduplication. ASLR increases physical memory consumption by more than 18% on 4 virtual machines, which is more than anticipated. Memory sanitization reduces physical memory consumption by about 10%. Even though memory sanitization has to initialize all memory assigned to the guest OS, the time overhead for memory deduplication is low. Page cache flushing (DropCache of Linux) reduces physical memory consumption by about 10%, but it requires pages to be re-read from disk, which reduces the performance of the guest OS. The combination of memory sanitization and page cache flushing reduces physical memory consumption by 20 - 35%, but takes over disadvantages of them. Fortunately, KSM is implemented as a background job and does not affect the performance of the guest OS severely, because memory merging is postponed. The impact on the performance of the guest OS is caused by page cache flushing and PIE, which increase I/O.

Some of this impact is due to the security capabilities not being recognized by the VMM. We propose that a page cache for dynamically created pages should be co-designed with the memory deduplication of the VMM. In future work we plan to implement the co-design between security capabilities and memory deduplication on a VMM.

## References


[1] Arcangeli, A., Eidus, I,. and Wright, C., Increasing memory density by using KSM, Linux Symposium, 19–28, 2009.

[2] Bugnion, E., Devine, S., and Rosenblum, M., Disco: Running Commodity Operating Systems on Scalable Multiprocessors, Symposium on Operating Systems Principles (OSDI), 143–156, 1997.

[3] Chow, J., Pfaff, B., Garfinkel, T., Christopher, K., and Rosenblum, M. Understanding data lifetime via whole system simulation, USENIX Security, 321–336, 2004.

[4] Chow, J., Pfaff, B., Garfinkel, T., and Rosenblum, M., Shredding your garbage: reducing data lifetime through secure deallocation, USENIX Security, 22–22, 2005.

[5] Collberg, C., Hartman, J.H., Babu, S., and Udupa, S.K., SLINKY: Static Linking Reloaded, USENIX Annual Tech, 309–322, 2005.

[6] Drepper, U., Security Enhancements in Red Hat Enterprise Linux (beside SELinux), http://www.akkadia.org/drepper/nonselsec.pdf, 2004.

[7] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A.C., Varghese, G., Voelker, G.M., and Vahdat, A., Difference Engine: Harnessing Memory Redundancy in Virtual Machines, Operating Systems Design and Implementation (OSDI), 309–322, 2008.

[8] Linux Sanitization Patch: http://lwn.net/Articles/334919/ 2009.

[9] Miłoś, G., Murray, D., Hand, S., and Fetterman, M.A., Satori: Enlightened page sharing, USENIX Annual Tech, 2009.

[10] Murphy, F., Position Independent Executables, http://blog.fpmurphy.com/2008/06/position-independent-executables.html?output=pdf, 2008.

[11] Owens, R., and Wang, W., Non-interactive OS Fingerprinting through Memory De-duplication Technique in Virtual Machines, IEEE International Performance Computing and Communications Conference (IPCCC), 2011.

[12] Pax project: http://pax.grsecurity.net/

[13] Suzaki, K., Yagi, T., Iijima, K., and Artho, C., Memory Deduplication as a Threat to the Guest OS, The Fourth European Workshop on System Security (EuroSec) 2011.

[14] Suzaki, K., Yagi, T. Iijima, K., Quynh, N.A., Artho, C., and Watanabe, Y., Moving from Logical Sharing of Guest OS to Physical Sharing of Deduplication on Virtual Machine, USENIX Hot topics in Security (HotSec), 2010.

[15] Ven, A., New Security Enhancements in Red Hat Enterprise Linux v.3, update 3, http://www.redhat.com/f/pdf/rhel/WHP0006US_Execshield.pdf , 2004.

[16] Waldspurger, C.A., Memory Resource Management in VMware ESX Server, Symposium on Operating Systems Principles (OSDI), 181–194, 2002.

[17] Xu, J., Kalbarczyk, Z., and Iyer, R.K., Transparent Runtime Randomization for Security, Reliable Distributed Systems, 2003.