Software Side Channel Attack on Memory Deduplication

Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, Cyrille Artho
National Institute of Advanced Industrial Science and Technology, RCIS, Tsukuba, Japan

1. Introduction

Memory deduplication merges same-content memory pages and reduces the consumption of physical memory. It is effective on environments that run many virtual machines with the same operating system. However, memory deduplication is subject to software side channel attacks, which discloses memory contents. It can be used to reveal the existence of an application or file on another virtual machine. Such an attack takes advantage of a difference in write access times on deduplicated memory pages that are re-created by Copy-On-Write [4]. Previously published exploits were immature. In this work we show untouched problems and refined exploits. Furthermore we show new applications of this technique, which enables secret communication between virtual machines on a processor. A secret marker on memory is used to detect the existence of a VM on a processor in a multi-tenant cloud computing environment.

2. Attack on Memory Deduplication

Memory deduplication is subject to a memory disclosure attack from an attacker's VM to a victim's VM. When data is written to a deduplicated page, the page is re-created with a copy of its contents by COW (Copy-On-Write). This causes the write access time to be slower than normal, because it includes the overhead to re-create the same-content page.

The preliminary idea of our memory disclosure attack is as follows. An attacker allocates same-content pages of a process or file in the memory of the attacker's VM and waits for these pages to be deduplicated. After that, the attacker issues a one-byte write access to the pages. If the pages are deduplicated, the write access time is longer than normal. In order to recognize the write access time difference, the attacker must know the time difference between deduplicated and non-deduplicated pages in advance. The write access time to zero-cleared pages (which are always deduplicated), and random data pages (which are not deduplicated) can be used for comparison.

A successful memory disclosure attack requires a careful implementation, adding issues caused by memory alignment, self-reflection, and run-time modification. These implementation issues cause false-negatives and false-positives.

2.1 Alignment Problem

The memory disclosure attack requires an exact match on memory pages as well as the aligned address of the pages. When a process is created, a binary file is loaded to memory pages by an interpreter. On Linux, ELF binary interpreter "ld-linux" is called to create a process. The contents of the binary file are loaded to aligned memory pages. An attacker has to prepare an identically aligned memory region to recognize the same content. To achieve this, an attacker can use the posix_memalign() function to allocate an aligned memory region in heap memory.

2.2 Self-Reflection Problem

If an operating system and applications create same-content pages on a single VM, memory deduplication works in the same way as on multiple VMs. The feature makes it difficult to execute a memory disclosure attack; this is called the "self-reflection problem".

The self-reflection problem is caused by different memory management of the page cache and the heap. When the attacker's program opens a target file using function open(), the contents of the opened file are stored in page cache memory by the Linux kernel. After that, the attacker's program loads the matching contents in its aligned heap memory using functions posix_memalign() and read(). At that time, the contents in heap memory are deduplicated to the page cache memory contents. In this situation, write access to the contents in heap memory is always delayed by self memory deduplication.

To prevent self-reflection, the images in the heap and page cache must be different. We compress the target file with gzip and expand it at run time. The contents in the page cache are the gzipped image. The contents in heap memory are decompressed by function gzread() and the contents in the heap memory are not deduplicated with the page cache, while being used to detect the same-content memory pages on the other VM.

2.3 Run-Time Modification Problem

Memory pages are modified more often than we expect. We call this phenomenon "run-time modification", which includes memory page swap-out, anonymous pages, ASLR (Address Space Layout Randomization), pre-loading, and self-modifying code. They cause false-negatives and false-positives in a memory disclosure attack.

The most common modification is memory page swap-out on a victim's VM. At that time, memory deduplication is disabled, and an attacker cannot detect it. However, the targeted process or file still exists in the process list (shown by command "ps") or the list of opened files (shown by command "lsof") on the victim's VM. This corresponds to a

false-negative. On the other hand, the attacker's memory is also swapped-out. In this case, access is delayed by the swap-in operation on the attacker's VM and causes a false-positive. This is solved by setting no-swap on the attacker's VM.

Even if a process terminates or a file becomes unused, the memory image still exists as anonymous pages for a long time. Anonymous pages do not change the contents in memory; instead the status of process or file is changed. Memory deduplication merges anonymous pages and causes a false-positive in a memory disclosure attack. This means that an attacker cannot always know the status of an application or file.

ASLR is a computer security technique to make it difficult for shellcode injection attacks to predict target addresses. It changes the position of the base of the code, libraries, heap, and stack for each process. Current operating systems have this mechanism enabled by default. Even though it seems to decrease the effect of memory deduplication, the contents in the most pages are unchanged by ASLR. An attacker need not care about ASLR.

Some operating systems have a pre-load mechanism in order to start applications quickly. Linux has a "readahead" system call which populates the page cache with contents of a file before it executes. The target process and file must not yet be listed on victim's VM, but an attacker detects the memory image. This causes a false-positive.

Self modifying code also causes false-negatives, because the code alters its own instructions while it is executing. In general, an attacker does not know the status of self-modifying code and cannot successfully carry out a memory disclosure attack.

Run-time modification makes an attack on memory deduplication difficult. However, the vulnerability and potential for an exploit still exist. Furthermore, the success of an attack also depends on the number of deduplicated pages. When an attacker gets many deduplicated pages, the confidence in the result increases.

3. Experiments

In our experience on KSM (kernel samepage merging) with the KVM virtual machine, the attack could detect the existence of "sshd" and "apache2" on Linux, and "Firefox" on WindowsXP. The results included some noise potentially causing false positives or false negatives. However, the results generally showed clear detection.

The attack also could detect a downloaded file on the Firefox browser when the caching is enabled. Even if the network is encrypted by TLS/SSL, a downloaded file is detected. However, when caching is disabled, the file cannot be detected because of the alignment issue.

4. Countermeasures

Countermeasures against memory disclosure attacks are considered in advanced virtual machine monitors. For example, Overshadow [1] encrypts the VM's memory and prevents the attack. However, this style hampers the effectiveness of memory deduplication. Another countermeasure is to use a sandbox which encrypts the memory of a process. This measure allows memory deduplication for other processes.

5. New Application: Searching a VM on Cloud

Our memory disclosure attack is able to detect a VM on a multi-tenant cloud computing environment. A VM can set a secret marker on its memory in order to announce its existence to other VMs unseen by administrator. The other VMs detect the secret marker using the memory disclosure attack. This secret channel is not easily detected by an administrator, because an attacker can cause spurious memory deduplication.

Such a marker must be used to prevent a conflict of interest between VMs on a processor. This implements a user-level exclusive allocation control, which is offered by sHype [3] as an administrator control. A marker may also help to collect VMs on a processor. This is also a kind of user level optimization. When a new VM does not find the marker, the user terminates the VM and recreates it till the VM is allocated on the same processor running the target VM. This allows VMs to reduce communication overhead.

Cloud storage deduplication exhibits the same vulnerability as described in paper [2]. This attack takes advantage of the elimination of traffic, when data is deduplicated. The method has the same restrictions. As in our work, the alignment problem occurs if an attacker tries to exploit block-level deduplication. The run-time modification problem occurs when a target file is deleted but the storage service retains its copy for a certain period.

6. Conclusions

We present a disclosure attack based on memory deduplication. Countermeasures and other applications of this technique are also mentioned. Administrators of multi-tenant cloud computing should be aware of them.

References

[1] Chen, X., et.al, Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems, ASPLOS'08.

[2] Harnik, D., Pinkas, B., and Shulman-Peleg, A., Side Channles in Cloud Service Deduplication in Cloud Storage, IEEE Security & Privacy, Vol. 8 Issue.6, 2010. Nov.

[3] Sailer, R., et.al, Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor, ACSAC'05.

[4] Suzaki, K., Yagi, T., Iijima, K., and Artho, C., Memory Deduplication as a Threat to the Guest OS, EuroSec'11.