Memory Deduplication as a Threat to the Guest OS

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 shares 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. Memory deduplication, however, is vulnerable to memory disclosure attacks, which 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. 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 IE6 and Firefox on WindowsXP. It also could detect a downloaded file on the Firefox browser. We describe the attack mechanism in this paper, and also mention countermeasures against this attack.

1. INTRODUCTION

IaaS (Infrastructure as a Service) type cloud computing uses a tremendous number of virtual machines. Even though data centers offer vast resources, the efficiency of a virtual machine is very important, because it is directly linked to the cost of cloud computing. To improve efficiency, *memory deduplication* [1,2,7,9,13] reduces the consumption of physical memory. Memory deduplication merges same-content memory pages on a physical machine, allowing more virtual machines to run on limited resources.

However, memory deduplication is vulnerable to memory disclosure attacks. A shared page has to be copied when a write access is issued to that page. This is called COW (Copy-On-Write). While the sequence of actions is logically valid and behaves consistently, the write access time is different between deduplicated and non-deduplicated pages. An attacker can use the time difference in a memory disclosure attack.

The attack used in this paper depends on a covert channel of COW, which is known exploit to leak information [14]. It does not violate any restriction of SLA (Service Level Agreements) of cloud computing. It only measures the write access times of its own memory, and guesses memory contents on other virtual machines. The attack uses a characteristic of the shared resource of a virtual machine, making it a kind of *cross-VM side channel attack* [11].

This attack has some restrictions. It is limited to exact matches on 4KB-pages, which are aligned on a 4KB boundary. An attacker has to prepare the target 4KB-page contents. Furthermore, the difference of write accesses depends on the environment, and the matching is disturbed by different kinds of noise: file caching, anonymous pages, and the timing of deduplication. This paper describes how to deal with these kinds of noise.

The attack is prevented by some improvements on the virtual machine monitor or guest OS. However, this decreases the performance of the virtual machine. Conversely, when used for live memory forensics, this attack technique increases security. For example, an administrator may detect a prohibited application or illegal file. We describe that as future work.

This paper is organized as follows. Section 2 reviews memory deduplication. Section 3 describes an attack on memory deduplication. Section 4 reports the results of this attack. Section 5 discusses this attack, and Section 6 discusses countermeasures. Related work is listed in section 7. Section 8 summarizes our conclusions.

2. MEMORY DEDUPLICATION

The memory images of virtual machines include many same-content pages, especially when the same guest OS runs on several virtual machines. Memory deduplication merges these same-content pages in physical memory.

Current virtual machine monitors are equipped with memory deduplication. The techniques are divided into two types; disk deduplication, which is content-aware, and memory deduplication, which scans memory periodically.

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 the pages match, TPS creates a shared mapping to the existing page. Satori has a similar policy for duplicate detection, although it does not use a special copy-on-write disk. Satori is implemented as para-virtualization on the Xen hypervisor and requires customization of the guest OS.

Periodical memory-scan deduplication is used in Content-based Page Sharing of VMWare ESX [13], the Differential 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 the same fingerprint is found, it compares the contents of the relevant two pages, and shares them if they are identical. Differential Engine features not only memory-scan type deduplication, but also patching and compressing. When almost identical pages are found, the small difference is taken as a patch and the nearly identical pages are merged. Compression is used when a page is not active for a long time. KSM (Kernel Samepage Merging) included from Linux kernel 2.6.32 onward, is a general 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.

Most implementations of periodical memory-scan type 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 the scanning, the exact similarity is computed by 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 page in unstable tree) move to the stable tree.

Pages are scanned at intervals, 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 can use is limited (the default is 25% of the available memory). Therefore, not all pages are scanned at a time.

A merged page in the stable tree is re-created when a write access is issued to the page; this technique is called Copy-On-Write (COW). The write access is reflected in the new page. When the old same-contents page has no other more buddy pages, it is removed from stable tree.

3. ATTACK ON MEMORY DEDUPLICATION

Memory deduplication may be subject to a memory disclosure attack from the attacker's VM to the victim's VM. The attack guesses a process running on a victim's VM or a file downloaded by a browser.

Memory deduplication shares 4KB pages which have the same contents. When data is written to a deduplicated page, the page is re-created with a copy of its contents (Copy On Write). This causes the write access time to be slower than normal, because it includes the overhead to re-create the same page. An attacker can measure access time to determine whether a page was deduplicated by another VM.

3.1 How to Get Matching Contents

An attacker must know the contents of 4KB memory pages which are used by the victim's application. It is not easy to get the exact memory image which is used as a process fingerprint, because some memory pages are not unique for process invocation of target application.

Some applications, such as sshd and apache2, prevent taking their memory image in the first place, for security reasons. They prevent taking the core dump of a process. While a debugger seems to be a useful tool to take a memory image, it cannot be used to get exactly aligned contents of memory, because a debugger has own memory mapping strategy. For example, "gdb" loads an ELF binary with 8-bytes shifted alignment. To get around the memory protection of sshd and apache2, we used a development version where this protection was disabled.

Moreover, current operating systems have a security mechanism called ASLR (Address Space Layout Randomization). The Linux kernel started to includes ASLR in version 2.6.12, released June 2005. ASLR changes the position of the base of code, libraries, heap, and stack for each process. It prevents malicious code (shellcode) injection attacks. Even though it seems to decrease the effect of memory deduplication, most pages are unchanged by ASLR.

Finally, the page cache prevents the matching of a memory image of a process. An ELF binary is loaded by the ELF interpreter (ld-linux) and assigned to aligned memory. The loaded ELF file is also saved to the page cache. As most pages of a process are shared with the page cache, an attacker cannot recognize if a page is used for a running process or the page cache. Furthermore, an ELF file is cached by copy or move commands, so an attacker cannot decide if the ELF file has been invoked or not. As the page cache still exists after a process terminates, an attacker cannot decide the time of process termination. An attacker only knows that the ELF file has been opened, by comparing the difference of write access times of a same-content page.

We measure the write access times on the pages mapping the contents of an ELF file. The pages which change for each process, are not treated specially, because they are only few. We also ignore file caching, because the binary files are not copied and moved frequently. We guess the moment of write access time difference to be process invocation.

3.2 Implementation Challenges

The memory disclosure attack on memory deduplication requires techniques to take care of the attacker's own memory and the timing of deduplication.

Because a memory disclosure attack may hit the attacker's own memory cache, an attacker must first clean up his memory. Physical memory is not cleared perfectly by rebooting [4,5]. The re-invocation of a VM, however, is the best way for clearing memory contents with zero, because the pseudo-physical memory is zero-cleared for memory isolation. An attacker can use this security countermeasure to his advantage.

When a file of target application is opened, its contents are cached; this leads to spurious matchings (false positives). In order to prevent false positives, the memory image for a disclosure attack is gzipped. A target file is gzipped and then ungzipped on 4KB aligned memory. This technique is similar to a runtime packer.

Memory deduplication takes time to be shared, because candidate pages are examined on being identical during a certain interval. Therefore, an attacker has to wait for a period of time. This period depends on the environment and the size of matching memory. If the period is too short, the prepared pages are not deduplicated by target pages on the victim's VM. If the period is too long, the attack leads to a false positives, due to memory caching or anonymous pages.

4. EXPERIMENTS

In our experiments, the target of our memory disclosure attack consisted of applications on a VM running Linux or Windows. We revealed the existence of executables, and discovered a file downloaded by a browser.

We ran the experiments on a machine with an Intel Core2Quad 3.0GHz processor and 4GB of memory. The host OS was Debian Lenny with the standard Linux kernel 2.6.33.5 being augmented with KSM. The attacker's and victim's VMs were running on KVM (0.12.4).

The write access was issued 5 minutes after a target application was invoked or a file was downloaded. After one byte of data was written on a duplicated page, the write access time was measured and analyzed.

We compared the write access times of pages containing random data, zero data, and target file data. Pages with random data are unique and are not merged with the stable tree of KSM. Their write access time is normal. Pages with zero data exist many times and are merged to stable tree of KSM. The write access time is delayed by COW.

We compared the write access times of pages with target file data before and after the launch of a target application on the victim's VM. The number of pages depends on the number of 4KB pages included in a target file.

4.1 Detected Applications on the Linux VM

We first tried to the disclose existence of secure applications on a victim's VM. In our experiments, these applications are sshd and apache2. The size of the sshd ELF file was 438,852 bytes, and the size of apache2 was 365,308 bytes. The matching target pages of sshd and apache2 were 107 and 89 pages, respectively. The last page which is less than 4KB is not target of matching, because the tail of 4KB are arbitrary contents. We measured the write access times of same-content pages on attacker's VM, before and after the application was invoked.

Tables 1 and 2 show the average access times on a page with zero and random contents, and target applications. The access time for zero data was more than 6 microseconds. Access to random data took less than 1 microsecond. These results indicate that the threshold is about 5 microseconds.

Figure 1 shows the difference in access times to each page. The zero and random pages are clearly separated, making deduplication apparent. The average write time of a page of sshd and apache2 increased to 7.50 and 7.56 microseconds, from 0.45 and 0.53 microseconds, respectively (Tables 1 & 2). These results indicate that the pages of the target application were deduplicated, which delayed the write access by COW. Therefore, we successfully detected the invocation of sshd and apache2 on the victim's VM.

Figure 1 also shows noise in write access times. The noise usually increased with the access time. Spikes in access times can be seen in the 55th page of zero data (Fig. 1 (a-2), 23 microseconds) and the 10th page of apache2 (Fig. 1 (b-2), 16 microseconds). These access times exceeded the threshold of COW (5 micro-seconds) and may lead to false positives. Therefore, we remove such outliers for accurate detection.

Another issue of false detection is accidental non-deduplication. Write access times for pages 70–75th of sshd were below 1 micro-second. They indicate the pages that were not deduplicated at measurement time. We do not know if it was caused by the page not yet being deduplicated, not yet loaded on victim's VM, or over-written. Detection has to take care of these behaviors by applying statistical analysis.

Table 1. Average write access times of the contents of sshd (in microseconds).

	zero	random	sshd
Before invocation	6.60	0.55	0.45
After invocation	6.51	0.42	7.50

Table 2. Average write access times of the contents of apache2 (in microseconds).

	zero	random	apache2
Before invocation	6.45	0.37	0.53
After invocation	6.24	0.40	7.56

4.2 Detected Applications on WindowsXP

We applied the same attack on a VM running Windows XP (SP3). The targets were Firefox (3.6.11) and IE6. The size of Firefox and IE6 were 912344 bytes (222 pages) and 93184 bytes (22 pages).

Tables 3 and 4 show average access times on the contents of Firefox and IE6. The results of zero and random accesses were almost the same and indicate that the threshold was 5 microseconds. However, the behavior of the target applications was different. The average write access time before the invocation of Firefox was 1.92 micro-seconds, and 5.68 micro-seconds for IE6.

Figure 2 shows the access times of each page. The results of Figure 2(a-1) and 2(b-1) indicate that some pages already existed before the target application was invoked, especially at the head of the executable file. We guess that *some content pages were preloaded;* common contents are used in other applications or preloaded for a faster application invocation. For IE6, 14 out of 22 pages were preloaded. After invocation, three pages are not deduplicated. We have tried this experiment several times, but the behavior was unstable. The existence of IE6 on a victim's VM is difficult to detect, because IE6 uses only 22 pages and seems to benefit from optimization by Windows XP.

Table 3. Average write access time of the contents of Firefox (in micro-seconds).

	zero	random	Firefox
Before invocation	6.45	0.43	1.92
After invocation	6.49	0.37	7.68

Table 4. Average write access times of the contents of IE6 (in micro-seconds).

	zero	random	IE6
Before invocation	6.59	0.27	5.68
After invocation	6.32	0.68	7.00

4.3 Detection of Downloaded Files
The memory disclosure attack can also be applied to find an opened file on a victim's VM. We have tried to detect a logo file when Firefox shows a home page.

We confirmed that the Google logo file was detected if page caching is enabled on Firefox. When the page cache was set to 0, detection failed. If an attacker leads a victim to a malicious home page which includes an identifiable logo file, the attacker can detect the page view from the victim's VM.

This disclosure attack is dangerous because it detects a page view even if the network is encrypted by TLS/SSL. Especially in a multi-tenant data center, this attack is serious, because it does not violate any SLA statements on cloud computing.

5. IMPACT ON SECURITY
Our memory disclosure attack has limitations, but still important privacy implications. Conversely, this technique may also be used by an administrator to increase security.

5.1 Scope of this Attack
An attacker does not know which VM contains the same contents, because KSM merges all pages used by VMs running on the machine. Because an attacker only knows that the contents exist on at least one VM which runs on the same physical machine, the exact VM cannot be decided on unless only two VMs exist.

An attacker does not know whether an application is running or not, because a cached binary file would not be sanitized and may exist in an anonymous page [4,5]. Furthermore, an attacker does not know if the binary was actually executed, because a binary file is also cached by copy and move commands.

Still, we consider the ability to detect the existence of files and downloaded contents to be an important privacy risk. Users of cloud computing should be aware that the partition between VMs is not a full protection against all types of attacks.

5.2 Security Enhancement
Papers [4,5] mention that data life time on memory is longer than we expect. Most users do not realize that memory keeps important data after an application terminates. The technique of memory deduplication attack is used to warn the remaining data on memory, especially it is useful for education because we confirm the effect using virtual machines.

For an administrator of cloud computing, memory deduplication attack may be used for live memory forensics. It can detect prohibited applications and files on other VMs. For example, insecure applications, a P2P downloader, or illegal data can be detected. The benefit of our method is that it only measures the write access time on the attacker's own VM. If an administrator can use live migration on any VM, he may move suspicious VMs to another CPU to narrow down the search for a particular VM containing the prohibited application or file.

6. COUNTERMEASURES
Our attack depends on the difference in write access times. It is prevented, if the target of memory deduplication contains only read-only pages, because they do not change during the life time of a page. In order to apply this solution, a virtual machine monitor has to know which pages are read-only. Read and write permissions are defined in the page table entries of virtual memory on a VM. Memory deduplication may use this information to implement this countermeasure. We also have to know the number of same-page read-only pages in the target operating system.

This attack is also prevented, if the victim's OS uses obfuscation code to change every runtime memory image. The binary loader may also change the cached file image. Then, an attacker cannot prepare an identical page for a target application.

It is not good idea to include a delay in write access, because that destroys the merit of deduplication. It is not clear how much delay time is suitable and how one randomly inserts a delay.

Memory sanitization seems to prevent a memory disclosure attack. However, it also works the other way. Memory sanitization helps an attacker to know that an application is running on a victim's VM. Because the cache and anonymous page are cleaned up, an attacker knows the exact launch and termination time of an application.

7. RELATED WORK
A different type of cross-VM side channel attack [11] also discloses applications running on the victim's VM. It monitors the behavior of a physically shared cache of CPUs, which runs the attacker's and victim's VMs. That cross-VM side channel attack also has strong restrictions, but it only accesses the physical cache assigned to a VM. It does not violate any SLA statements on cloud computing, like our memory disclosure attack.

A covert channel of memory deduplication is mentioned in Satori [9]. Satori can prevent this attack because Satori is content-aware disk deduplication and can recognize illegal matching. Satori also has a mechanism to refuse memory deduplication to prevent the exploit. These are strong defenses, but Satori requires para-virtualization on guest OS and is not applied to any OS. Currently widely-used memory deduplication scans memory periodically and is independent of the guest OS. This paper treats memory-scan-type memory deduplication and shows a real vulnerability and exploit.

A *cold-boot attack* [8] is a kind of physical side channel attack, which freezes physical memory and scans the data. The attack discloses any data in memory. It has no restriction on alignment and page size, but it requires elaborate equipment. Our memory disclosure attack assumes a virtual machine monitor that uses memory deduplication, which we think will be commonly used on next-generation cloud computing.

The IEEE 1394 attack is also physical side channel attack for memory disclosure. IEEE 1394 enables us to read from and write to physical memory by accessing a DMA controller, while an operating system owns the memory. It also has alignment and page size restrictions, but it does not require exact pattern matching. While that attack has weaker restrictions than ours, it requires physical equipment.

Countermeasures against memory disclosure attacks between VMs are considered in advanced virtual machine monitors. Overshadow [3] and SP3 [12] make difficult to access pseudo physical memory from another VM. While that security architecture is effective, it does not go together with memory deduplication, because the memory of each VM is encrypted by a different key.

SLINKY [6] uses memory deduplication to increase security. SLINKY shows that memory deduplication reduces the increased memory caused by statically linked shared libraries, which protects against the vulnerability of dynamically linked shared libraries. Unfortunately this technique is not applied on current Linux distributions, because many applications use dynamically linked shared libraries to avoid license contamination problems. Paper [10] uses a self-contained binary translator to integrate shared libraries into an ELF file. These results show that the contents of a guest OS should consider the existence of memory deduplication.

8. CONCLUSIONS

This paper describes a memory disclosure attack on another VM that uses memory deduplication. The attack measures write access times on pages that are re-created by COW (Copy On Write) of memory deduplication.

The attack has some restrictions (only matching on aligned 4KB pages, noise, wait time of merging, etc.), but it can detect running applications and downloaded files on a victim's VM. In our experiments, the attack detected sshd and apache2 on Linux and Firefox on WindowsXP on a victim's VM. The detection of IE6 of WindowsXP was difficult, because it contains few pages, some of which are preloaded.

Our attack also successfully detected the Google logo file download by Firefox, if page caching is enabled on Firefox. This suggests that an attacker can detect a page view, if that page includes an identifiable logo file. It works even if the network is encrypted by TLS/SSL. This attack is serious, because it does not violate any SLA statements on cloud computing.

Fortunately, the attack can be prevented by some countermeasures: restriction of deduplication to read-only pages, or obfuscation of the guest OS. These countermeasures, however, decrease the performance of deduplication. The analysis of the trade-off between security and performance constitutes future work.

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] Chen, X., Garfinkel, T., Lewis E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J., and Ports D. R. K., Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems, Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2–13, 2008.

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

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

[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] Halderman J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., and Felten, E.W., Lest We Remember: Cold Boot Attacks on Encryption Keys, USENIX Security, 45–60, 2008.

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

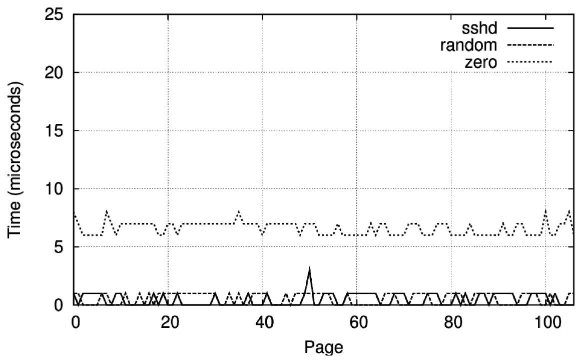
[10] Suzaki, K., Yagi, T. Iijima, K., Quynh, N.A., Artho, C., and Watanabe, Y., Moving from Logical Sharing of Guest OS to Physical Sharing of Deduplication on Virtual Machine, USENIX Workshop on Hot topics in Security (HotSec), 2010.

[11] Ristenpart,T., Tromer, E., Shacham, H., and Savage, S., Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds, Proceedings of the 16th ACM conference on Computer and Communications Security, 199–212, 2009.

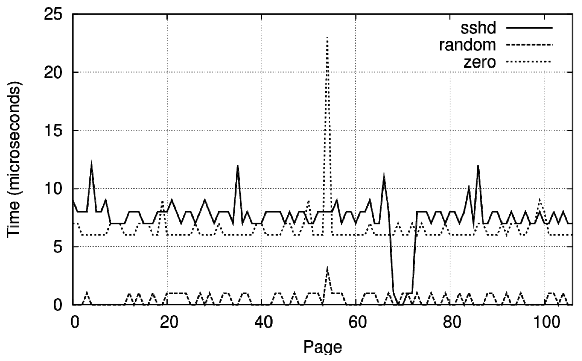
[12] Yang, J., and Shin, K., Using Hypervisor to Provide Application Data Secrecy on a Per-Page Basis, Conference on Virtual Execution Environments (Vee), 2008.

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

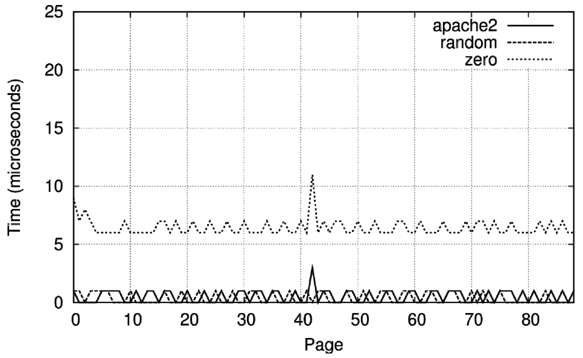
[14] Warner, A., Li, Q., Keefe. T.F., and Pal S., The impact of multilevel security on database buffer management, 4th European Symposium on Research in Computer Security (ESORICS), 1996.



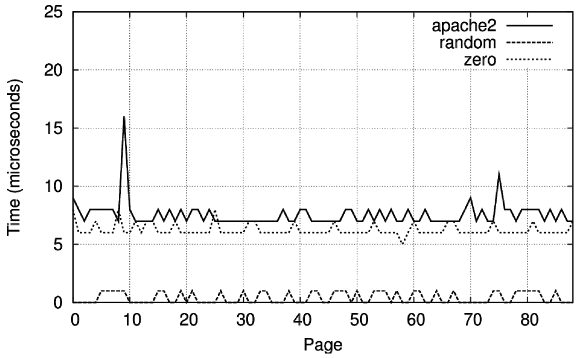
(a-1) Before sshd launches on victim VM



(a-2) After sshd launches on victim VM

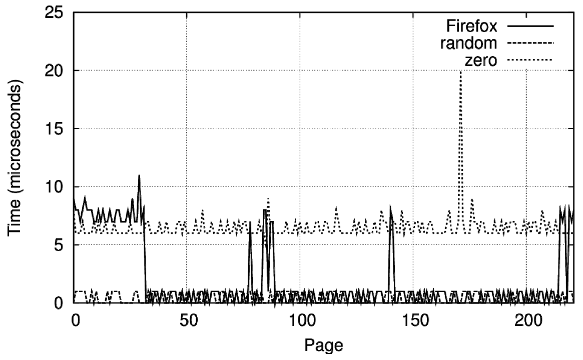


(b-1) Before apache2 launches on victim VM

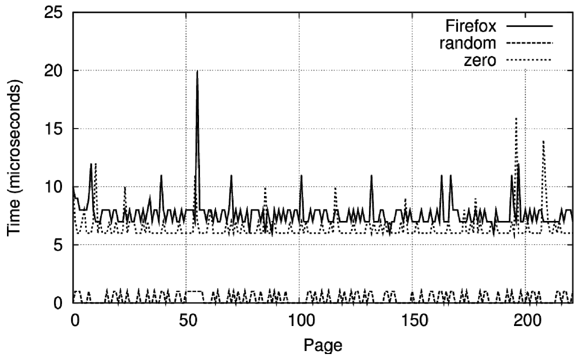


(b-2) After apache2 launches on victim VM

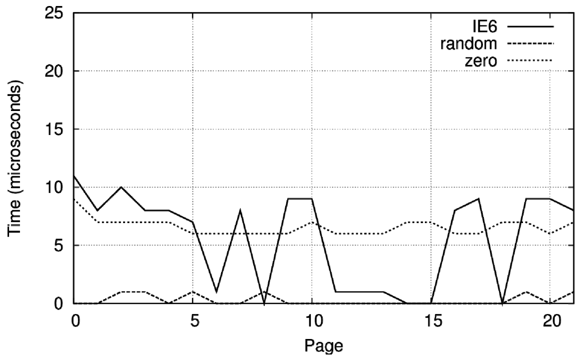
Figure 1. Write access time on attacker's VM before and after sshd/apache2 is launched on victim's VM of Linux.



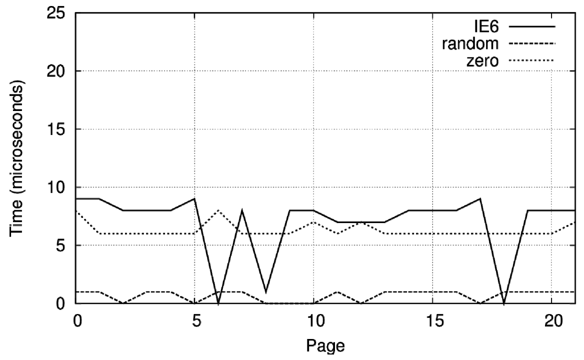
(a-1) Before Firefox launches on victim VM



(a-2) After FireFox launches on victim VM



(b-1) Before IE-6 launches on victim VM



(b-2) After IE-6 launches on victim VM

Figure 2. Write access time on attacker's VM before and after FireFox/IE-6 is launched on victim's VM of Windows XP.