Non-interactive OS Fingerprinting through Memory De-duplication Technique in Virtual Machines

Rodney Owens and Weichao Wang
Department of SIS and CyberDNA Center
University of North Carolina at Charlotte
Charlotte, NC 28223
Email: { rvowens@, weichaowang@ }uncc.edu

***Abstract*—OS fingerprinting tries to identify the type and version of a system based on gathered information of a target host. It is an essential step for many subsequent penetration attempts and attacks. Traditional OS fingerprinting depends on banner grabbing schemes or network traffic analysis results to identify the system. These interactive procedures can be detected by intrusion detection systems (IDS) or fooled by fake network packets. In this paper, we propose a new OS fingerprinting mechanism in virtual machine hypervisors that adopt the memory de-duplication technique. Specifically, when multiple memory pages with the same contents occupy only one physical page, their reading and writing access delay will demonstrate some special properties. We use the accumulated access delay to the memory pages that are unique to some specific OS images to derive out whether or not our VM instance and the target VM are using the same OS. The experiment results on VMware ESXi hypervisor with both Windows and Ubuntu Linux OS images show the practicability of the attack. We also discuss the mechanisms to defend against such attacks by the hypervisors and VMs.**

I. Introduction

OS fingerprinting is an essential step for many subsequent penetration attempts and attacks. Only after identifying the type and version of the OS of a system, can an attacker determine the vulnerabilities to exploit. Traditional OS fingerprinting schemes [1], [2], [3], [4], [5] are usually interactive procedures through IP packet analysis, service querying, or chronological exploits. Since these mechanisms usually initiate some interaction with the target system and use the contents and delay of the network packets from the target OS to determine its type and version, the target OS can monitor the network traffic to detect such attempts. It can also disable the responses or generate fake packets to disguise the fingerprinting procedure.

The proliferation of cloud computing environments and virtual machine platforms creates a new path for non-interactive OS fingerprinting. The technique of cloud computing allows end users to outsource the storage, computation, development, and even the complete computing infrastructure to third party service providers through virtual machines [6]. In a VM hypervisor, multiple virtual machines share the same hardware platform. Although perfect isolation among VMs is required by design [7], researchers have identified several mechanisms to break such isolation through the schemes such as side channels [8]. For example, researchers find that the shared cache may become a side channel for the detection of the web traffic access rate or even keystrokes of the co-resident VM instances [9]. As another example, CCCV [10] is a system that can create a covert channel using the CPU loads to secretly transmit information among virtual machines.

In this research we seek to investigate OS fingerprinting in virtual machine hypervisors with the memory de-duplication functionalities enabled (such as VMware ESX and ESXi [11], Extended Xen [12], [13], and KSM (Kernel Samepage Merging) [14] of the Linux kernel). The memory de-duplication technique takes advantage of the similarity among memory pages so that only a single copy and multiple handles need to be preserved in the physical memory, as shown in Figure 1. Here each of the two virtual machines $VM1$ and $VM2$ needs to use three memory pages. Under the normal condition, six physical pages will be occupied by the VMs. If memory de-duplication is enabled, we need to store only one copy of multiple identical pages. Therefore, the two VMs can be fit into four physical pages (note that we have both inter- and intra-VM memory de-duplication). This technique can reduce the memory footprint size of VMs and the performance penalty caused by memory access miss. However, it will break the isolation among VMs and introduce new vulnerabilities of non-interactive OS fingerprinting. The objective of this paper is to exploit the vulnerability by constructing concrete attacks on the VMware ESXi hypervisor, and investigate the mechanisms to defend against such attacks.

The overview of our approach is as follows. When we detect that the target VM has been launched, we will use

978-1-4673-0011-7/11/$26.00 ©2011 IEEE

the mechanisms described in [9] to initiate multiple VM instances using different OS onto the same physical box. Without losing generality, we call the operating system of the target VM as S_t and those of the attacker's VMs as S_{a1}, S_{a2}, $\cdots$, S_{an}. The objective is to determine whether or not S_{ai} and S_t are of the same type. To achieve the goal, we will let the memory de-duplication mechanisms identify and merge those identical pages. Once this procedure is accomplished, we will introduce reading and writing operations to the memory pages that are unique for each different OS type. Since the hypervisor handles the operations differently for those de-duplicated pages and the pages with their own copies [15], we can measure the accumulated differences in the memory access delay to figure out whether or not S_{ai} and S_t are of the same type. During this procedure, the attacker's VMs do not need to directly interact with the target VM.

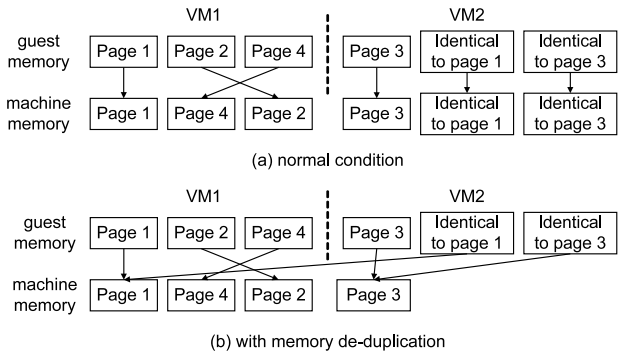


Fig. 1. Memory de-duplication reduces the OS footprint size.

We design several mechanisms to defend against such attacks by the hypervisor and the guest OS respectively. At the virtual machine level, the guest OS can load the unique memory pages belonging to other OS into its memory to obfuscate the fingerprinting procedures. The hypervisor can monitor the behaviors of different VMs and avoid the de-duplication of any memory pages belonging to the OS image files. More details of the defense mechanisms will be discussed in Section IV.

Compared to existing OS detection mechanisms, the proposed approach has the following advantages. First and most importantly, it is a non-interactive fingerprinting procedure since during the attack we only conduct operations on our own VM instances. This non-interactive property will prevent the target OS from detecting the fingerprinting operations. Second, we have analyzed the memory footprint of many different OS types and identified the memory pages that are unique to each type. Our preliminary results show that the number of unique pages is large enough to generate a measurable difference in access delay. Last but not least, our experiment results on VMware ESXi with both Windows and Linux systems show that this attack is practical. Since our approach does not conflict with the interactive OS fingerprinting mechanisms, they can work together to improve the detection accuracy.

The remainder of the paper is organized as follows. In Section II we describe the details of the OS fingerprinting approach. We discuss the generation of the signature files and the procedures to measure the accumulated differences in access delay. In Section III we present the implementation of the attacks and the experimental results when VMware ESXi hypervisor is used. Section IV discusses the problems such as VM co-residence detection and the prevention of the attacks. Finally, Section V concludes the paper.

II. The Proposed Approach

A. System Assumptions and Background

In the investigated scenario, we assume that an attacker can initiate VM instances in the same cloud infrastructure as the target VM. We also assume that through the co-residence detection mechanisms discussed in Section IV we can determine whether or not the target system is running as a guest on the same physical box as the attacker's VMs. Since the target VM could have very sophisticated Intrusion Detection/Prevention Systems in place, it can detect any OS fingerprinting attempts through network interactions. Under this condition, the attacker wants to learn what OS version the target is, without any network scans, in order to exploit known vulnerabilities and conduct a direct one-hit attack before the IDS/IPS can respond.

We assume the attacker has root control over the VM instances that it initiates. We also assume the attacker's VMs have large enough memory (such as 512MB) to avoid very frequent page swapping. We do not assume the attacker can decide how many CPU cycles it is allowed to use, nor do we assume the attacker can decide how much physical RAM its VMs are allowed to consume. These are reasonable assumptions based on current industry practice. Without losing generality, we assume that the host uses $4KB$ memory pages.

Since in our experiments we use VMware ESXi as the hypervisor, below we briefly describe its memory de-duplication operations. A comprehensive description can be found at [16]. To avoid unnecessary delay during page loading, whenever a new memory page is read from the hard disk, ESXi will allocate a new physical page for it. Later, ESXi will use idle CPU cycles to locate the identical memory pages in physical RAM, and remove duplicates by leaving pointers for each VM to access the same memory block. Hash results of the memory page contents are used as index values to locate identical pages. To avoid false de-duplication caused by hash collisions, a byte-by-byte comparison between the pages will be conducted. While the reading operations to the de-duplicated pages will

access the same copy, copy-on-write is used to prevent one VM from changing another VM's memory pages. Specifically, on writing operations a new page will first be allocated and copied. This procedure will incur extra overhead compared to writing to not-shared pages, which will lead to a measurable delay when a large number of shared pages are allocated and copied.

ESXi uses three system wide parameters to adjust how it looks for de-duplicated pages. These are "ShareScanTime", "ShareScanGHz", and "ShareRateMax". ShareScanTime specifies how much time the administrator would like ESXi to scan an entire VM's memory pages for duplicates. ShareScanGHz specifies the maximum number of pages to scan in physical RAM per second. ShareRateMax specifies how quickly the pages should be scanned per-virtual machine. VMware sets these parameters to some default values to offer the least amount of overhead, while still finding identical pages fairly quickly. According to VMware, the algorithm also scans faster if it determines that there is a high likelihood of finding duplicate pages based on the previous scans, and vice-versa.

B. Generation of OS Signatures

The first step to turn the proposed approach into a practical attack is to identify the memory pages that are unique to each OS type. To accomplish this task, we load the OS image files into a VM instance and then conduct a memory dump. The dump files are then cut into $4KB$ pages. We adopt the mechanism in [12] and use hash results of the memory contents as indexes to locate the identical pages. For each OS type and version, we analyze the memory dump files from different installation sources so that we can remove the impacts of the factors including different hardware drivers, different product keys, and different product IDs. Once we have categorized the memory pages by OS types and versions, we can find out which memory pages are unique to each OS version, but present in all copies of that OS version. These memory pages will hereafter be referred to as *OS signatures*. Please note that not all memory pages can be used as the signatures. For example, the memory dump of Windows XP SP3 contains 59,238 copies of all-zero memory pages. These pages, however, cannot be used for OS fingerprinting since they are not unique to any specific OS type.

The OS signatures created in this fashion give us a real world representation to what can be found in the wild. The off-line memory dump and analysis is also much faster, easier, and only slightly less accurate than calculating what the similar memory pages would be based on the OS's documented loading behavior on essential OS files. Based on these considerations, we believe that an attacker would most likely build OS signatures in the same fashion as we have for our approach.

During the signature generation procedures, we conduct cross-comparison among only the memory pages of different OS types. Therefore, it is possible that some of the signature pages will appear in the memory images of other software applications or data files. This may affect the OS fingerprinting accuracy since the attacker would not be able to identify the sources of these memory pages. Fortunately, because of the large number of diverse memory pages ($2^{4096\times 8}$ if every bit has the same probability to be 0 or 1), the impacts on fingerprinting accuracy will be very limited. Quantitative investigation of this problem will be conducted in future work.

C. OS Fingerprinting Procedures

As we describe in Section I, VMware ESXi uses different methods to handle the writing operations to the de-duplicated pages and pages with their own copies. For the pages with their own copies, the writing operation can be conducted immediately. For the de-duplicated pages, a new copy must be created first. This memory allocation and copy procedure will introduce extra processing delay. Our OS fingerprinting procedure, therefore, is to detect the accumulated difference in access delay caused by the de-duplication between our OS signatures and the target VM. To achieve the goal, we propose to adopt the following schemes.

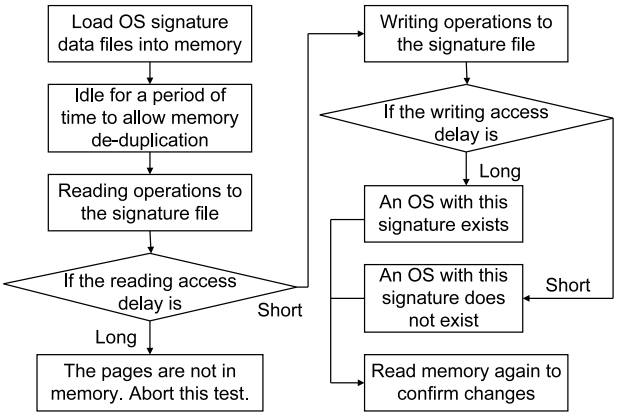


Fig. 2. The proposed OS fingerprinting procedure.

First, the extra processing delay is caused by the writing operations. Since many OS files are read-only, we plan to construct a different signature file for each OS type by chaining its unique memory pages together. Therefore, by loading the corresponding signature file into memory and writing to it, we can force the hypervisor to create a new copy for any de-duplicated pages belonging to that OS.

Second, since the pages that have not been accessed recently will be swapped out by the hypervisor, we need to distinguish the delay of hard disk reading from that of copy-on-write. To accomplish this task, we plan to conduct a reading operation to the OS signature file right before

the writing operation. If a page is in the memory, this reading operation can be accomplished immediately and it will not change the de-duplication status of the page. However, if a page has been swapped out, this reading operation will force the hypervisor to execute a hard disk access. Since VMware ESXi will allocate a new memory page for the newly read contents, the next step of writing can be accomplished immediately and will not provide us useful information about OS fingerprinting. In this way, we can distinguish between the two types of access delay.

With these basic components established, our OS fingerprinting procedure is illustrated in Figure 2. Although here we use the narrative description "short" and "long", the experiments in Section III will help us to determine quantitative thresholds for OS fingerprinting in real systems.

When we confirm that our VM instance and the target VM are located on the same physical box through co-residence detection, we will read the OS signature files into memory. These files will then be left alone for a period of time to allow memory de-duplication algorithms to locate and merge the identical pages. Once the de-duplication procedure is accomplished, we will conduct a reading operation on the signature files. The purpose of this operation is to determine whether or not the files have been swapped out to hard disk. If the reading access delay matches the hard disk loading time, we will abort the fingerprinting procedure since the newly loaded pages all have their own copies. Otherwise, we will conduct a writing operation to the signature files. Since we already know that these pages are in memory, based on the delay of the writing operation, we can determine whether or not they experience the copy-on-write procedures. If so, we know that a VM instance matching this signature file exists in the physical box.

III. Implementation and Experimental Results

Although the basic idea of the proposed approach is straightforward, many issues need to be solved before we can turn the idea into a practical attack. For example, we need to examine the size of the OS signature files to make sure that the accumulated delay is actually measurable. At the same time, we need to examine the timekeeping schemes in hypervisors so that we can measure the delay accurately. In this Section, we present the details of our implementation of the attack and the experiment results.

A. Experiment Environment Setup

Our VMware ESXi server is running on a PC with a dual core 2.4GHz Xeon CPU, 4GB RAM, and SATA hard drives. To simplify the experiment setup and examine the practicability of the attack, during each trial there are only the target VM and our attacking VM instance running on ESXi. OS fingerprinting in more complicated scenarios will be studied in future work.

In order to build the OS signatures, we have generated and examined the memory dump files of different operating systems to find memory pages that would be unique for a specific OS version. We sampled four types of Linux/Unix and eleven types of Windows to cross examine their memory pages. The number of unique pages of each OS type/version is summarized in Table I. From the table, we find that the size of the signature files ranges from several thousands to tens of thousands of pages. The experiment results presented later will show that the accumulated difference in access delay to these pages can be easily detected.

Another issue that we are facing is the accuracy of time measurement. Traditionally a computer provides three schemes to measure the length of a time duration: time of the day, CPU cycle counter, and APIC (advanced programmable interrupt controller) timer. The first method provides the measurement granularity of seconds which is too coarse for our application. The second method will be a good candidate for time measurement if the attacker's OS completely owns the hardware platform. In a VM-based system, however, it cannot accurately measure the time duration. For example, if a page fault happens during our reading operation, the hypervisor may pause the CPU cycle counter while it fetches the memory page. Therefore, the delay caused by hard disk reading will not be measured. Based on these observations, we choose to use the timestamp service provided by masm32 in:

```
\masm32\lib\winmm.lib
```

to access the APIC timer. Specifically, we use the timeGetTime directive because it provides a 1 millisecond resolution [17]. According to [18], VMware has fully emulated the local APIC timer to provide accurate time readings, so the page fault handler built into ESXi to handle de-duplication will not pause the virtual local APIC timer.

Newer operating systems include a technique known as Address Space Layout Randomization, or ASLR. In this technique, operating systems try to prevent code injections from being successful by changing the memory locations of executables. For Windows Vista, this technique was explored in detail by Symantec in [19]. According to Symantec,

> When executing a program whose image has been marked for ASLR, the memory layout of the process is further randomized by placing the thread stack and the process heaps randomly. The stack address is selected first. The stack region is selected from a range of 32 possible locations, each separated by 64 KB or 256 KB...

This technique, as implemented by Microsoft, does not

TABLE I
SIZE OF OS SIGNATURE FILES IN $4KB$ PAGES

OS Type	Fedora		Ubuntu		Windows									
	7	8	10	11	2000	XP SP1	XP SP2	XP SP3	2003	Vista 32 bit	Vista 64 bit	Win 7 32 bit	Win 7 64 bit	2008 32 bit
# of unique pages	4572	5274	8179	5547	4239	4090	3325	3582	3695	5503	20053	9753	23977	5638

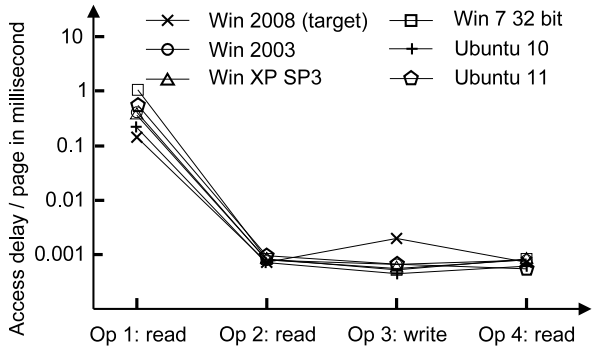
randomize within memory page boundaries. Memory pages will be randomized in their physical locations, but the contents within each page will remain the same. Because our technique does not depend on the physical location of memory pages, only that they exist with their full, undisturbed contents, this technique has no effect on the fingerprinting results. In our experiments we do not turn off the ASLR switch in guest operating systems. The experiment results confirm the discovery by Symantec.

Three reasons make us choose Windows 95 as the OS of the attacker's VM instance. First, we want an operating system with a very small memory footprint size so that its contents will not pollute the de-duplication results. Since Windows 95 can easily run on a platform with only $64MB$ RAM, it achieves a good balance between the size and the supported functions. Second, there are not many systems still running Windows 95 so there is a very low probability that our VM instance and the target VM are running the same OS. Last but not least, we have extended experiences in working with the portable executable (PE) file format and Windows 95 is one of the earliest systems supporting this format. To further reduce extra delay caused by high level programming languages, we implemented the memory access and time measurement functions in assembly language.

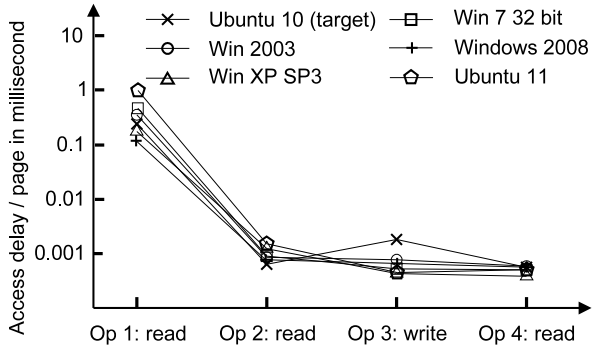
As illustrated in Figure 2, we have divided the OS fingerprinting procedure into four groups of operations. Operation group one will read the first 32 bits of every memory page of the OS signature file and store each result into the EAX register. Our program then sleeps the processor for a duration of several hours to allow de-duplication to occur. After that, it will perform operation groups two through four immediately after each other. Group two does the same operation as group one. Group three writes junk data into the first 32 bits of every memory page for each OS signature. Finally, group four reads back the memory pages to confirm the changes.

B. Experiment Results

We conduct six groups of experiments to evaluate the OS fingerprinting capability of the proposed approach under different levels of computation and memory access workload. The target OS that we try to identify includes Windows 2008 Server 32 bit, Windows 7 32 bit, and Ubuntu 10. In the attacker's VM instance, we load the OS signature files of Windows 2008 Server 32 bit, Windows 2003 Server, Windows XP Service Pack 3, Windows 7 32 bit, Ubuntu 10, and Ubuntu 11. We choose these signatures since they are the favorites of IT staff in our day-to-day lives and would have a high probability of a hit in the real world. The idle periods for memory de-duplication are usually four hours. Our results are shown in Figures 3 through 7. Since the delays span across multiple degrees of magnitude, we use log-scale Y-axis. Since the signature files of different OS types have different sizes, we illustrate the average reading and writing time per memory page in the figures. For a signature file that contains thousands of memory pages, the accumulated difference in access delay will be several to tens of milliseconds, which can be easily measured by the proposed approach. Each node in the figures is the average value of five experiments with the same configuration. All time delays are measured in milliseconds. To help readers to better understand the experiment results, the average access delay to the signature file of the target OS will always be represented by "x".



(a) when Windows 2008 is the target OS



(b) when Ubuntu 10 is the target OS

Fig. 3. OS fingerprinting results with low CPU and memory demands.

In the first experiment, we set up a baseline test case. Here one instance of Windows 2008 is initiated in the host

as the target OS. To reduce the impacts of page swapping on the approach and accelerate the de-duplication calculation procedure, the target VM instance does not activate any other applications. The attacker's VM runs Windows 95 and loads the signature files of Windows 2008, 2003, XP SP3, Windows 7 32bit, Ubuntu 10, and Ubuntu 11 into its memory. As shown in Figure 3.(a), the delay of the first reading operations is relatively long since the pages have to be read from the hard disk. After that, the target VM instance and the attacker's VM are left idle for four hours to give the de-duplication algorithms enough time to scan the memory. Since each VM has enough memory to hold the OS files, we do not expect a lot of page swapping to happen. This is confirmed by the very short access delay of the second group of reading operations. The access delay of the writing operations, however, demonstrates the difference among the signature files. Here the access delay to the signature file of Windows 2008 is about three times longer than those of other OS types because of the copy-on-write operations. Our approach can successfully identify the type of the target OS in this baseline setup. The second experiment has the same configuration except that the target VM is running Ubuntu 10. As the results shown in Figure 3.(b), the long delay will allow us to easily identify the target OS.

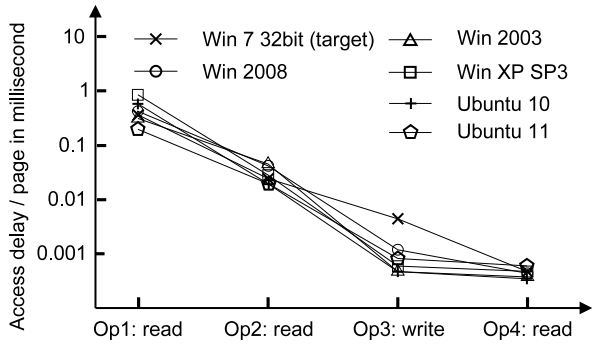


Fig. 4. OS fingerprinting under medium computation workload.

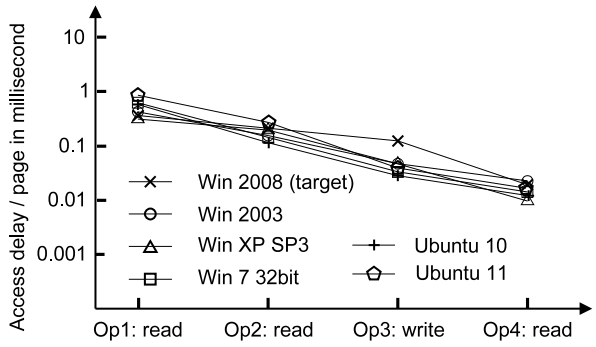


Fig. 5. OS fingerprinting under medium level memory demand.

In the third experiment, we want to investigate the impacts of computation workload in the target VM on the fingerprinting accuracy. Here the basic setup is the same as the first experiment. The only difference is that we use Windows 7 32bit as the target OS. To introduce medium-level computation workload on the target VM, we run DES encryption and RSA encryption algorithms on the VM. The results are shown in Figure 4. Here the second group of reading operations become slower. This could be caused by context switch between the VMs since the target VM is running some applications. The writing delay to the signature files of the target OS is still much longer than those of other OS types (4 to 8 times longer). From this figure, we find that our OS fingerprinting approach will work properly when the computation workload on the target VM is not too heavy.

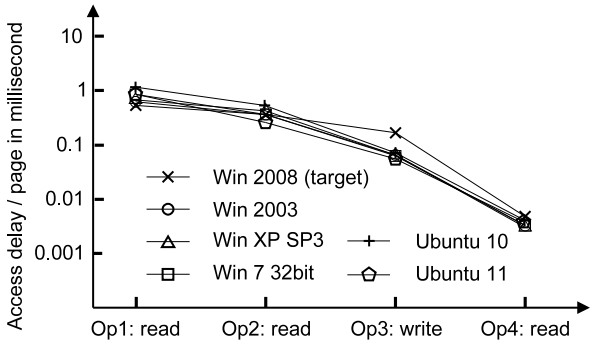


Fig. 6. OS fingerprinting under medium level computation workload and memory demand.

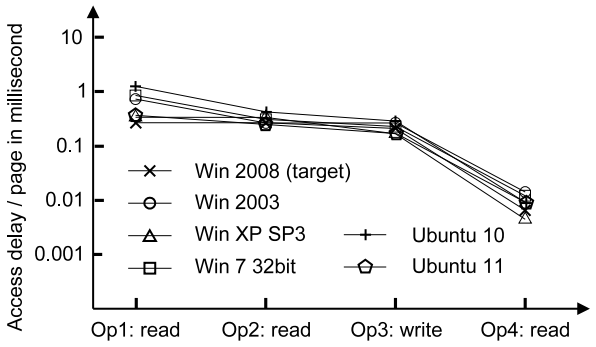


Fig. 7. OS fingerprinting under extremely heavy computation workload and memory demand.

In the fourth experiment, we want to investigate the impacts of memory demands on the target VM on the fingerprinting accuracy. Again we choose Windows 2008 as the OS of the target VM. We run a memory testing software "QA+Win32" [20] to introduce medium-level memory demand on the target VM. The results are shown in Figure 5. Here the second group of reading operations become even slower because of the page swapping. The write delay to the signature files of the target OS is still 2 to 3 times longer than those of other OS types. This figure shows that our approach can work properly under medium

level memory demand on the target VM.

In the fifth experiment, we have medium-level computation workload and memory demand on the target VM. The results are shown in Figure 6. The OS type of the target VM can still be identified based on the slow speed of the writing operations.

In the last group of experiments, we want to investigate the OS fingerprinting accuracy of our approach under very heavy computation workload and memory demand. We use Windows 2008 as the OS of the target VM. The target VM is running Prime95 [21], a CPU and RAM stress test software package. Very frequent memory page swapping is expected. The results are shown in Figure 7. We can see that the writing delay to the signature files of different OS types cannot be distinguished from each other. This experiment shows that memory de-duplication based OS fingerprinting will not work properly under extremely heavy computation workload and memory demand. This is reasonable since under this condition, the hypervisor would not have enough computation power or a relatively stable memory image to identify and maintain the de-duplicated pages.

IV. Discussion

A. The problem of VM instance co-residence detection

Although the experimental results in Section III are very encouraging, one problem is left unsolved: how can we put our VM instances onto the same physical box as the target and determine their co-residence. We plan to experiment with two adversarial strategies to place attacker's VM onto the same physical box as the target. The first strategy is brute-forcing placement. In this mechanism, we will launch numerous instances over a period of time and conduct co-residence test discussed below. Previous research [9] shows that this simple approach has about 10% probability to successfully put at least one VM of the attacker onto the same physical box as the target. In the second attack strategy, we plan to explore the strong sequential and parallel placement locality of VM instances that has been shown in third party clouds such as Amazon EC2 [9]. With this property, if attackers launch VM instances relatively soon after the launch of the target, they have a better chance to achieve co-existence.

We understand that different VM management systems have different mapping policies among the virtual machines and physical boxes. For example, some systems use static mapping between the two groups. Under this condition, we can use the information such as the Dom0 IP addresses and internal IP addresses in the cloud to determine whether or not two instances are on the same physical box. Such management policy will also allow us to launch a new instance immediately after the termination of our previous instance so that the new one will take the slot of the terminated one.

The dynamic mapping between the VMs and physical boxes may even help attackers on their fingerprinting procedures. For example, to maximize the benefits of memory de-duplication, the hypervisors may move all instances with the same OS type onto a single physical box. Under this condition, the attacker only needs to use the co-residence detection schemes to determine whether or not its VM is on the same physical box as the target.

We plan to use two groups of mechanisms to verify co-residence of attacker's VM and the target. In the first group we will examine the similarity of their Dom0 IP addresses and internal IP addresses in the cloud since many third party cloud management systems use static mapping between the addresses of VMs and the physical boxes. In the second group we will investigate load-based co-residence detection schemes through side-channels [9], [22]. The basic idea is to induce different levels of computation and data access loads onto the target and measure the operation delay of attacker's VM instance. If the two sequences of events match very well, the two VMs have a good chance to be located on the same physical box.

B. Porting the attack to other hypervisors

Although we implement and evaluate the proposed attack using only the VMware ESXi hypervisor, the technique can be easily ported to other hypervisors that support memory de-duplication. For example, researchers [15] have examined information leakage caused by memory de-duplication in Linux KSM (Kernel Samepage Merging). In their experiments, attackers construct memory pages that have the same contents as some specific applications. They will then measure the write access time to these pages to determine whether or not the applications are initiated in the target VM. In [12], memory harvesting using de-duplication is implemented in Xen. Therefore, the similar attack can be conducted in this environment with minor changes.

C. Prevention of de-duplication based OS fingerprinting

Special mechanisms can be designed to defend against the de-duplication based OS fingerprinting attacks at both the hypervisor and the target VM. As the analysis in Section II shows, the differences in access delay will allow attackers to detect only the existence of specific pages in the main memory. However, they cannot identify to which applications or data files these pages belong. Therefore, a VM can obfuscate the attack by intentionally loading the signature files of other OS types into its memory. Since most signature files contain only a few thousand unique memory pages, we can easily fit multiple signatures into the main memory of a VM. The hypervisor could also intentionally load multiple VMs with different guest OS onto the same physical box. This scheme, however, will reduce the benefits brought by memory de-duplication.

Two mechanisms can be adopted by the hypervisor to defend against the studied attacks. First, the hypervisor can label the memory pages so that the de-duplication operations can be conducted on only the pages of data files but not OS images. The disadvantages of this mechanism include the required modification to existing hypervisors and the reduced memory usage efficiency. In the second mechanism, physical isolation among the VMs will be enforced so that only mutually trusted VMs will be positioned in the same physical box. For example, Amazon introduces a new service with physically isolated, tenant-specific hardware so that NASA will join its cloud infrastructure [23]. In [24], researchers have designed a scheme to help end users to verify the physical isolation among VMs.

V. Conclusion

In this paper we propose a new OS fingerprinting mechanism for VM instances on hypervisors that enable the memory de-duplication functionality. The analysis shows that the reading and writing delay of the memory pages will demonstrate a measurable difference when they do not have their own copies in the memory. Experimental results on multiple OS types show that each of these OS contains a large number of unique pages that can be used as its signature. We can use the co-residence detection schemes to launch VM instances onto the same physical box as the target and determine its OS type. Different from previous approaches that need interaction with the target, our approach is more difficult to detect by IDS or network traffic monitor.

Immediate extensions to our approach consist of the following aspects. First, we plan to experiment with more complicated scenarios to determine whether or not such attack can be used for OS fingerprinting in them. We will also experiment with other hypervisors such as extended Xen and Linux KSM to generalize the attacks. Second, we want to extensively study the prevention mechanisms. We plan to implement the mitigation mechanisms at both the VM level and the hypervisor level. We will also evaluate their effectiveness and overhead. The research will provide new information to determine the tradeoff between memory management strategies and security in hypervisors.

Acknowledgment

This research is supported in part by NSF CNS award 1143602.

References

[1] P. Auffret, "Sinfp, unification of active and passive operating system fingerprinting," *Jour. Comp. Virology*, vol. 6, no. 3, pp. 197–205, 2010.

[2] Fyodor, "Remote os detection via tcp/ip stack fingerprinting," http ://www.insecure.org/nmap/nmap-fingerprinting-article.html, 1999.

[3] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting," in *Proceedings of the first USENIX workshop on Offensive Technologies*, 2007, pp. 6:1–6:10.

[4] D. Richardson, S. Gribble, and T. Kohno, "The limits of automatic os fingerprint generation," in *ACM workshop on Artificial intelligence and security (AISec)*, 2010, pp. 24–34.

[5] G. Taleck, "Ambiguity resolution via passive os fingerprinting," in *International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2003, pp. 192–206.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, April 2010.

[7] A. Verma, P. Ahuja, and A. Neogi, "Power-aware dynamic placement of hpc applications," in *Proceedings of the annual international conference on Supercomputing*, 2008, pp. 175–184.

[8] A. Aviram, S. Hu, B. Ford, and R. Gummadi, "Determinating timing channels in compute clouds," in *Proceedings of ACM workshop on Cloud computing security workshop*, 2010, pp. 103–108.

[9] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proc. of ACM Conference on Computer and Communications Security (CCS)*, 2009.

[10] K. Okamura and Y. Oyama, "Load-based covert channels between xen virtual machines," in *Proceedings of the ACM Symposium on Applied Computing*, 2010, pp. 173–180.

[11] VMWare, "Esxi configuration guide," VMware vSphere 4.1 Documentation, 2010.

[12] D. Gupta, S. Lee, M. Vrable, S. Savage, A. Snoeren, G. Varghese, G. Voelker, and A. Vahdat, "Difference engine: harnessing memory redundancy in virtual machines," *Commun. ACM*, vol. 53, no. 10, pp. 85–93, 2010.

[13] X. Zhang, Z. Huo, J. Ma, and D. Meng, "Exploiting data deduplication to accelerate live virtual machine migration," in *IEEE International Conference on Cluster Computing*, 2010, pp. 88–96.

[14] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using ksm," in *Linux Symposium*, 2009, pp. 19–28.

[15] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Memory deduplication as a threat to the guest os," in *Proceedings of the Fourth European Workshop on System Security*, 2011, pp. 1:1–1:6.

[16] VMWare, "Understanding memory resource management in vmware esx 4.1," VMware vSphere 4.1 Documentation, 2010.

[17] Microsoft Developer Network, "timegettime," http://msdn.microsoft .com/en-us/library/ms713418(VS.85).aspx, 2010.

[18] VMWare, "Timekeeping in vmware virtual machines," http://www. vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf, 2010.

[19] Symantec, "An analysis of address space layout randomization on windows vista," http://www.symantec.com/avcenter/reference/ Address_Space_Layout_Randomization.pdf, Tech. Rep., 2007.

[20] Eurosoft, "Qa+win32-diagnostic software," http://www.eurosoft-uk .com/qawin32.html, 2010.

[21] G. Woltman, "Prime95," a component of Great Internet Mersenne Prime Search (GIMPS), http://www.mersenne.org/, 2009.

[22] H. Raj, R. Nathuji, A. Singh, and P. England, "Resource management for isolation enhanced cloud services," in *ACM Cloud Computing Security Workshop*, 2009, pp. 77–84.

[23] B. Stone and A. Vance, "Companies slowly join cloud computing," New York Times, 18 April, 2010.

[24] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, "Homealone: Co-residency detection in the cloud via side-channel analysis," in *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*, 2011, pp. 313–328.