POSTER: A Covert Channel Construction in a Virtualized Environment

Jidong Xiao
The College of William and Mary, Williamsburg, VA
jxiao@cs.wm.edu

Zhang Xu
The College of William and Mary, Williamsburg, VA
zxu@cs.wm.edu

Hai Huang
IBM T.J.Watson Research Center, Hawthorne, NY
haih@us.ibm.com

Haining Wang
The College of William and Mary, Williamsburg, VA
hnw@cs.wm.edu

ABSTRACT

Memory deduplication has been widely used in various commodity hypervisors. However, while this technique improves memory efficiency, it has an impact on system security. In particular, memory deduplication is usually implemented using a variant of copy-on-write techniques, for which, writing to a shared page would incur a longer access time than those non-shared. By exploiting this artifact, we demonstrate a new covert channel can be built in a virtualized environment.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection

Keywords

Covert Channel, Virtualization, Memory Deduplication

1. INTRODUCTION

Memory deduplication is a technique used in various commercial and open source hypervisors, including VMWare ESX, Xen, and Linux KVM. The key idea is, if multiple memory pages have the same content, then the hypervisor only needs to keep one copy, and such pages are called deduplicated pages. And later, if one of the deduplicated pages is modified, a copy-on-write (COW) technique is used. In other words, the page will be copied, and the write operation will take effect on the copied page. This allows a significant amount of memory space to be saved when there are many identical pages.

However, this technique can also expose unexpected security vulnerabilities. Due to the extra copy operation, a write to a deduplicated page and a normal page (non-deduplicated page) incur different access times [2, 3]. In this poster, by exploiting this artifact of memory deduplication, we demonstrate how to create a covert channel in a virtualized environment. The major contribution of this work is briefly summarized as follows.

CCS'12, October 16–18, 2012, Raleigh, North Carolina, USA.
ACM 978-1-4503-1651-4/12/10.

We develop a reliable covert channel to transfer information between two virtual machines. We validate the feasibility and effectiveness of this covert channel for information leakage through a series of experiments. We run our experiments on top of Linux/KVM with Kernel Samepage Merging (KSM) implemented as a loadable kernel module1. Our experimental results show that the new covert channel can reach nearly 100% accuracy, and even in a system under high computation and memory pressure, it can still achieve a reasonable transfer rate.

2. COVERT CHANNEL CONSTRUCTION

A covert channel allows two isolated entities to communicate with each other. It consists of a sender and a receiver. Typically, in a cloud environment, both entities are virtual machines running on the same physical machine. The sender (of information) is generally the victim of an attack, and we assume that the sender has been compromised by the attacker, therefore, it is under the attacker's control. To bypass traditional monitoring techniques and remain hidden as long as possible, the sender encodes the information the attacker is interested in and injects "signals" onto a covert channel. The receiver is a virtual machine launched by the attacker to be a co-resident with the sender VM. The receiver will probe the covert channel and decode the "signals". In this way, the information on the victim machine can be leaked while minimizing the chance of the attacker being disclosed. Existing works have exploited various shared hardware resources to build a covert channel in virtualization environments, such as L2 cache [4]. However, the existing techniques are not robust against environment noises.

Figure 1 illustrates the framework of our covert channel. In the first step, the sender and receiver load a certain amount of memory with identical content. This can be easily done by having both the sender and receiver opening and reading the same file. Next, the sender encodes the information, e.g., writing to certain pages so that the contents of these pages are different from those launched by the receiver. Once the pages are modified, the sender and receiver sleep and wait for the system to merge these pages. Finally, the receiver should write to all memory pages used by the covert channel and record the write access time. The copy-on-write mechanism makes writing to shared pages cost more time than those that are not. Thus, the receiver can easily iden-

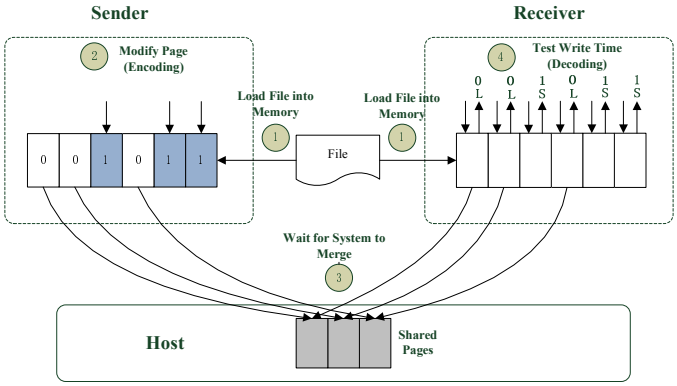

Figure 1: Covert Channel Overview

tify those modified pages. With the knowledge of which pages are modified, the receiver can decode the information. This is basically how the sender and receiver communicate via the memory deduplication covert channel.

The encoding mechanism is further detailed as follows. Since we can detect memory deduplication at the granularity of a page, we make each page represent one bit of information. At the sender side, an unmodified page indicates a 0 and a modified page denotes a 1. For instance, as illustrated in Figure 1, if we want to transmit 001011 through the covert channel, after the sender and receiver both read six identical pages, the sender should modify the 3rd, 5th, and 6th pages. After sleeping for a period of time, the receiver will write to these six pages and record access time. Since pages 1, 2, and 4 remain unchanged, the memory deduplication mechanism should have merged them with their counterparts. On the other hand, pages 3, 5, and 6 have been modified by the sender, and hence they are not deduplicated. Thus, the receiver takes much more time (according to our observation, at least six times more) to write to pages 1, 2, and 4 than to pages 3, 5, and 6. At the receiver side, a long access time indicates a 0 and a short access time denotes a 1. Therefore, in our example the receiver can infer that the sender is sending 001011.

3. EXPERIMENTAL EVALUATION

The experiments are conducted on an Intel Xeon 3.07GHz, Quad-Core processor with 4GB memory. Each Guest is assigned 1GB memory. The host OS is Linux(2.6.37) with Kernel Samepage Merging (KSM) implemented as a loadable kernel module.

We first validate the feasibility of the memory deduplication based covert channel. We boot two virtual machines on the same hypervisor. One virtual machine is selected as the sender and the other is set as the receiver. We load a file of size 1088KB (i.e., 272 4KB pages) into memory. In order to ensure the reliability of deduplication, we set the sleeping time to 250 seconds. We set up four sets of experiments. In each experiment, the sender modifies different pages to transfer different messages to the receiver, and we record all the write access time to these 272 pages at the receiver side to verify whether the information has been correctly delivered.

Figure 2 shows the experimental results. In the first experiment, the sender wants to send a 272-bit data beginning

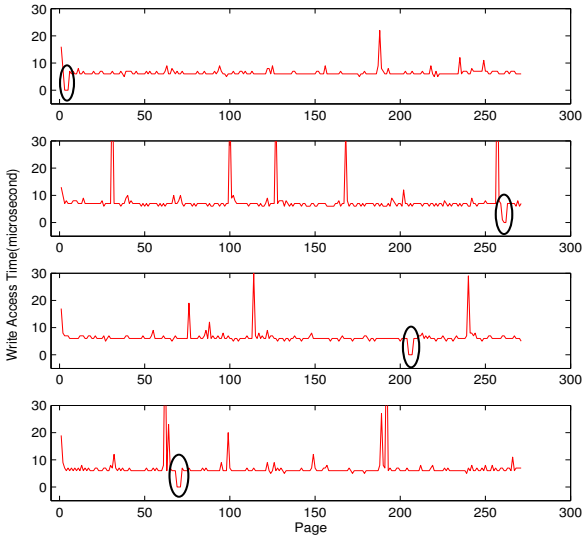

Figure 2: Information Transferring in Covert Channel

with 00111 as the first 5 bits followed by all 0s. Between the sender and receiver, if a modified page denotes a 1 and an unmodified page denotes a 0, the sender would need to modify the 3rd, 4th, and 5th pages to encode such data. For the other three experiments, the sender modifies pages 260, 261, and 262 in the second experiment, pages 205, 206, and 207 in the third experiment, and pages 69, 70, and 71 in the last experiment. From Figure 2, we can see although different experiments demonstrate different write access spikes, the sender-modified pages always incur much less write access time. This is because once the sender modifies a page, the page will become a non-deduplicated page, and hence less time is required for the following write access issued by the receiver. We also observe that, the time of writing to a deduplicated page is always at least 6 times longer than writing to a non-deduplicated page. This strong signal-to-noise ratio implies that the covert channel can be reliably established.

Next we evaluate the channel's bit rate and robustness to see whether it is practical. The bit rate (R) is determined by the time needed to complete one transmission (T), and the volume (V) of information that can be delivered in one transmission: R=V/T. Moreover, since the transmission time is dominated by the sleeping time (i.e., the transmission time is almost equivalent to the sleeping time), we can use the sleeping time as the time required to complete one transmission without losing much accuracy. In this set of experiments, we also set up two virtual machines as the sender and the receiver, respectively. In each round, the sender and the receiver load a file with a certain size into memory to build the channel. We gradually increase the sleeping time until it reaches a threshold such that the information can be transferred through the covert channel without any errors. We call such a threshold stable sleeping time. Then, we gradually increase the size of the loaded file and record the change of the stable sleeping time. After obtaining the stable sleeping time for each different memory size, we can calculate the bit rate using the formula of R=V/T.

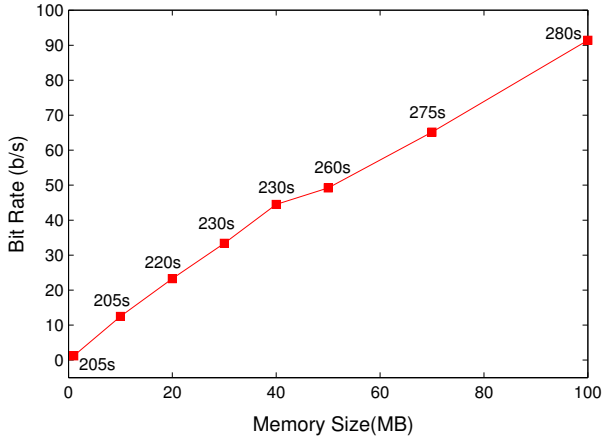

Figure 3: The relationship between memory size and bit rate. The stable sleeping time is also marked.

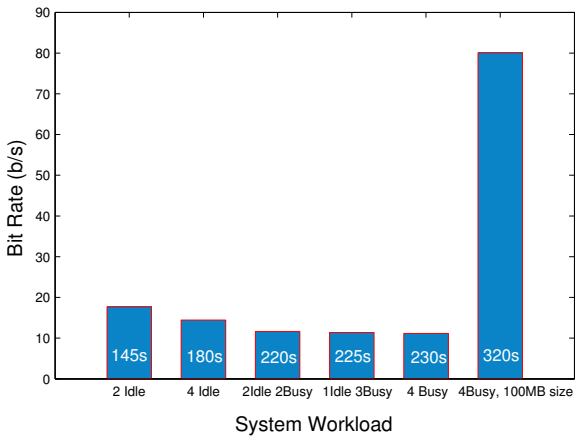

Figure 4: The relationship between computation workload and bit rate. The stable sleeping time is also marked.

Figure 3 illustrates the dynamics of achieved bit rate with the change of memory size, in which the stable sleeping time in each case is also marked. When the memory size increases from 1MB to 100MB, we can easily observe that the bit rate significantly increases but the stable sleeping time only modestly increases. When the memory size is 1MB, or 256 4K pages, it takes our channel 205s to complete the transmission. The bit rate under this case is merely 256b/205s=1.24bps. However, when the file size reaches 100MB, namely 25,600 pages, the covert channel only takes around 280s to complete the transmission. In this way, the bit rate can surge to above 90bps. This means that we should select relatively large memory size for the covert channel construction. Unfortunately, using too much memory might easily expose our covert channel. Thus, a memory size in the range of 80MB to 100MB can be a good choice.

The performance of the covert channel is also dependent upon the system workload. Given a certain memory size, we run a set of experiments to study how the bit rate changes when the system workload increases. In the experiments, we first boot two virtual machines as the sender and receiver, then we gradually increase the system workload by launching more virtual machines and running computational benchmarks on virtual machines.

Figure 4 illustrates the dynamics of the bit rate under different system workloads, which can be divided into six cases. For the first five cases, the size of the leveraged memory is fixed at 10MB. Initially, only two virtual machines are launched and they collude with each other to construct the covert channel. In this first case, the stable sleeping time is 145s and the bit rate is 17.66bps. In the second case, we boot another two virtual machines (we call them the *irrelevant VMs*) with the same configuration and keep them idle. Now the stable sleeping time is increased to 180s and thus the bit rate is reduced to 14.44bps. To further increase system workload, in the third case, we run CPU intensive benchmark Cuadro [1] on the two *irrelevant VMs*. Again, the bit rate drops to 11.64bps. Then we also run the benchmark on the sender, i.e. the victim VM. This fourth scenario is close to a real world one: the victim machine and other co-resident machines are busy running services while the attacker-controlled receiver can keep idle. Such a system configuration in the fourth case yields the stable sleeping time of 225s with the bit rate of 11.38bps. In the fifth case, when we also run the benchmark on the receiver, the bit rate drops a little again to 11.13bps.

Overall, from Figure 4, we can see that the performance of the covert channel is not severely influenced by system workload. When the system workload increases, the bit rate only drops slightly. Moreover, in the experiments of the sixth case, all the virtual machines are busy running benchmarks and we leverage 100MB memory to construct the covert channel. Since the stable sleeping time is no more than 320s, we can still achieve a bit rate above 80bps. This indicates that even in a close-to-real-world scenario where multiple users run different services on different VMs above a single physical machine, our covert channel is still able to transfer information in a decent rate. According to the analysis of [4], a real world L2 cache channel can only achieve a bit rate around 11bps with some errors. Thus, it is clear that our memory deduplication covert channels outperform L2 cache covert channels.

4. CONCLUSION

Memory deduplication is originally designed for improving performance, however, it can also be exploited for security purposes. In this poster, we have demonstrated that, by exploiting one artifact of memory deduplication, attackers can build a reliable covert channel in a virtualized environment.

5. REFERENCES

[1] Cuadro cpu benchmark. `http://sourceforge.net/projects/cuadrocpubenchm`.

[2] K. Suzaki, K. Iijima, T. Yagi, and C. Artho. Software side channel attack on memory deduplication. *SOSP POSTER*, 2011.

[3] K. Suzaki, K. Lijima, T. Yagi, and C. Artho. Memory deduplication as a threat to the guest OS. *European Workshop on System Security*, 2011.

[4] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting. An exploration of L2 cache covert channels in virtualized environments. In *Cloud computing security workshop*, pages 29–40. ACM, 2011.