PCA: Page Correlation Aggregation for Memory Deduplication in Virtualized Environments

Min Zhu[1,2], Kun Zhang[1,2(✉)], and Bibo Tu[1,2]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{zhumin,zhangkun,tubibo}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. To intelligently share limited memory across VMs in IaaS cloud, content-based page sharing (CBPS), like KSM, is utilized to greatly reduce the memory footprint of VMs. CBPS merges same-content pages into a single copy. However, it introduces some serious cross-VM covert channel threats. Besides, it has heavy overhead due to vast otiose operations, such as page comparisons and checksum calculations, when detecting page sharing opportunities. In this paper, we propose a novel memory deduplication approach called page correlation aggregation (PCA), which can efficiently reduce otiose operations. Meanwhile defends covert channels. One key idea of PCA is to divide VMs' pages into several sets, since pages with similar attributes have the greatest possibility with the same content. In PCA, the pages of VMs are firstly divided into different groups according to VMs' attributes. In each group pages are further separated into different classifications based on their access permissions. Thus page comparisons are restricted to the same classification for sharing. The other is that PCA introduces a dedicated cache to mitigate the latency of COW (Copy- On-Write) used for conducting covert channels. We have conducted a prototype on KSM, one popular CBPS technique. Our experimental results show that PCA reduces otiose operations about 40%, and can effectively resist covert channels.

Keywords: Covert channel · Secure memory deduplication
Page classification · KSM

1 Introduction

In IaaS (Infrastructure as a Service) clouds, the available memory size has become one of major bottlenecks to run more virtual machines (VMs) on a single machine. In this scenario, however, there is plenty of redundant data, resulting in lower utilization and higher hardware costs. To alleviate it, memory deduplication is proposed to detect and eliminate redundant memory pages.

D. Naccache et al. (Eds.): ICICS 2018, LNCS 11149, pp. 566–583, 2018.
https://doi.org/10.1007/978-3-030-01950-1_33

A typical representative is content-based page sharing (CBPS) [2–4] that is transparently implemented in hypervisor layer, such as VMware vSphere, Xen and KVM etc. If multiple memory pages have the same content, CBPS only reserve a single copy for these so-called deduplicated pages in a copy-on-write (COW) way. When a deduplicated page is written, a new page is re-created with a copy. Many work [7,13] has shown virtualized environments have a large amount of memory can be condensed, especially for VMs running similar applications or OSes.

However, CBPS introduces a new security threat: covert channels, which can extract and transmit data from co-located VMs exploiting additional access delays caused by COW for deduplicated pages. In this way, standard security measures, such as access control and data audit, are bypassed. Covert channels indeed pose realistic and serious threats to information security in the cloud [23–32]. For example, a hundred bytes of credit card can be secretly stolen less than 30 s, even a thousand bytes of a file can be stealthily trafficked within 3 min [31]. However, covert channels cause negligible CPU and memory utilization, so it's hard to be detected in a cloud of heavy workloads. Moreover, it is robust against environmental noises than cache based covert channel [21,22]. Therefore, covert channels become ideal choices for secret data transmissions.

CBPS also has performance interference [11,12]. Since CBPS manages memory pages of VMs indiscriminately and globally, for each candidate page it needs to be compared with a large number of otiose pages repeatedly to detect its sharing opportunities, which induces plenty of otiose page comparisons and checksum calculations (so-called otiose operations), which take up the main run-time consumption of CBPS (over 60%). The induced CPU overhead will degrade the performance of VMs. As the number of comparisons increases, the CPU overhead increases correspondingly. And as the increasing capacity of mergeable memory, the comparisons expand proportionally, which makes the situation worse.

In this paper, we propose an efficient and secure memory deduplication approach called page correlation aggregation (PCA), which highly aggregates VM's pages based on correlative page properties. In PCA, VMs are divided into disparate groups based on their attributes (e.g. OS types), because inter-VM sharing is very low with different VM attributes. Further, according to page access permissions, pages of each group are divided into different classifications, since these pages have a higher possibility with the same content. PCA depends on VM introspection (VMI) to obtain VM attributes and access permission. Thus, a candidate page is only compared with pages in its classification and group, which reduces otiose operations. In addition, this feature prohibits covert channels if the receiver and sender are separated into different groups. Memory contents of one group are securely protected from another group. To prevent covert channels within the same group, an intuitive approach would be to add noise to obfuscate the specific covert-channel information. Based on this idea, PCA provides a dedicated cache to reserve deduplicated pages, so as to mitigate the write latency when occurring COW on deduplicated pages. By doing this, the receiver may consider the received bit as '1' due to low write latency, whereas the actual transmitted bit is '0'.

We have implemented a prototype in Kernel Samepage Merging (KSM), one widely used implementation of CBPS. PCA is guest-transparent due to the use of VMI. We evaluated it through several benchmarks. The results show that PCA is efficient and practical. Although our current implementation is performed on KSM, PCA can be applied to any CBPS instances due to its generality. Actually, PCA also works for deduplicating native processes.

Overall, we have made the following contributions:

- We propose a novel memory deduplication approach for covert channels defense and otiose operations reduction assisted by VMI.
- We employ the inter-VM grouping and intra-group classification mechanism to efficiently reduce otiose comparisons. Page sharing is performed just in the same classification of the same group.
- We provide a dedicated cache for writable deduplicated pages to mitigate the write latency by adding timing noise when occurring COW.
- We implement PCA based on KSM in a real experimental system. The experimental results show that PCA is able to reach its effectiveness.

The rest of this paper is structured as follows. Section 2 presents the necessary background. Section 3 details its design and implementation. Section 4 presents the analysis of PCA in terms of security and efficiency. Section 5 surveys related work of page sharing and covert channel. Section 6 concludes this paper.

2 Background and Threat Model

In this section, we describe some necessary background including the basic principles of KSM, and CBPS-based covert channel, which will help motivate our solution. And then the threat model and assumptions are discussed.

KSM: KSM, one implementation of CBPS in KVM, exists as a kernel thread in the host OS (Operating System) and periodically scans advised anonymous pages to merge identical pages. When a VM starts, Qemu invokes *madvise* call to advise its memory as mergeable. KSM manages pages by two red-black trees: **stable tree** and **unstable tree**. The stable tree stores already shared pages with write-protected, also named KSM pages. The unstable tree records the candidate pages that don't change frequently.

The content of pages is the index of the two trees for node search and insert. As shown in Fig. 1, in each scan round, a candidate page is firstly compared with pages in the stable tree. If there is a match, the candidate page is merged with it. Otherwise, its checksum will be recalculated before searching in the unstable tree, because KSM does not merge frequently changed pages. If the calculated checksum differs from the previous recorded one, KSM updates the checksum and continues with the next candidate page. Otherwise, if there is a match in the unstable tree, the candidate page is merged and added into the stable tree, while the matched page is purged from the unstable tree. If no match is found, the candidate page is only inserted into the unstable tree. Wherever a match

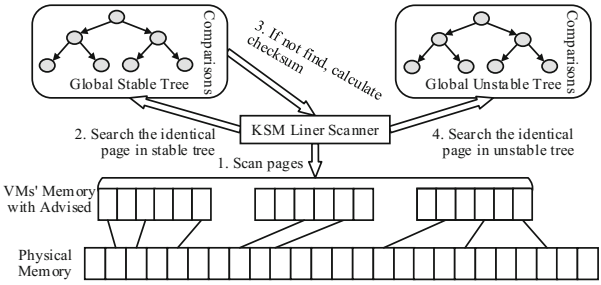

Fig. 1. The KSM overview for memory sharing.

is found, the page table entries of deduplicated pages are replaced by the KSM page with write-protected. After a full scan, the unstable tree is rebuilt since its pages content may be changed during the scan.

Covert Channel Attack: CBPS can be used to perform cross-VM covert channels to observe or transmit data because a COW page fault on deduplicated pages incurs measurable access latency than no COW. To transmit data, a program is required to run on the victim as the data sender. However, this requirement does not reduce their usefulness, since current OS is fragile and vulnerable, attackers are able to compromise the victim.

A successful covert channel attack needs four steps, as shown in Fig. 2. First, the sender and receiver load a certain amount of memory pages with identical content, for example reading a same file with memory alignment. Note, in this step, self-reflection should be avoided. Next, the sender encodes the information, e.g., writing certain pages. We make each page represent one bit of data. For instance, an unmodified page indicates bit 0 and a modified one denotes bit 1. If we want to transmit 10110010, the sender should modify the 1st, 3rd, 4th and 7th pages. Then, the sender and receiver need to wait for the eight pages being merged. Finally, the receiver writes all the eight pages and records their write access latency. At the receiver side, a long access time indicates 0, otherwise 1. The receiver can easily infer the transmitted data is 10110010.

Threat Model: We assume the attacker and victim are separate VMs co-resident on the same server. The attacker can compromise the target VM through multiple attack vectors. Thus she can stealthily place the 'sender' in the victim VM, which may be a spy program for filling pages with data that are expected to find in the victim's memory. We also assume an active adversary model, in which the sender and receiver are cahoots. We assume the hypervisor, VMI tools and hardware are trusted. The introspected data structures cannot be modified by attackers, which is common to most existing VMI-based solutions [5,6].

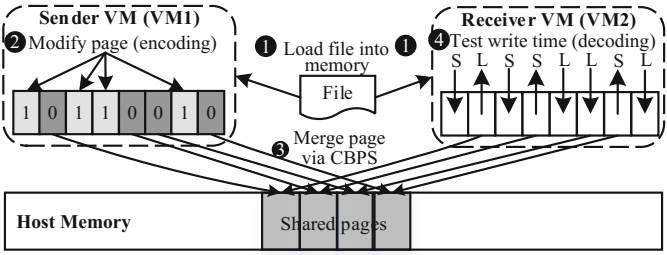

Fig. 2. To build a covert channel attack

3 Design and Implementation

In this section, we firstly introduce a highlight overview of PCA. Then we discuss how PCA classifies the pages of VMs, maintains the deduplicated pages to counter covert channels, and processes deduplication hints.

3.1 Overview

We have designed PCA with KSM as an example to clarify our approach. PCA includes two main mechanisms. One is "VM grouping and page classification", which places pages that have much higher probability with the same content together, while pages with different content are divided. This mechanism is used to reduce the KSM overhead. Incidentally, covert channels of inter-group VMs are prevented. The other is "KSM cache", which adds system noise for covert channels between intra-group VMs.

Figure 3 shows its architecture, which consists of four main components: (1) Page Permission Collector (PPC), (2) KSM Cache, (3) Grouping Manager (GM), and (4) Classification Manager (CM). PPC is used to capture the access permission of each candidate page from the guest OS by VMI-based interfaces, especially write access permission. Because pages with write access permission not only affect page sharing opportunities, but also may be used by covert channels. KSM cache maintains the private information of each deduplicated page: once a page is merged, it is inserted into the KSM cache. While the reconstructed KSM includes two parts: the GM and CM. The GM is in charge of dividing VMs into groups according to the VMs' attributes. The duty of CM is to aggregate pages into different classifications based on page access permissions collected by PPC. Once system startup, the components are enforced and complement each other, which will be discussed in the following sections.

3.2 VM Grouping and Page Classification

Most of previous work is focus on how to detect more sharing opportunities, but they have overlooked the KSM own loss properties due to otiose operations in global trees. To solve this problem, the pages of VMs should be divided into

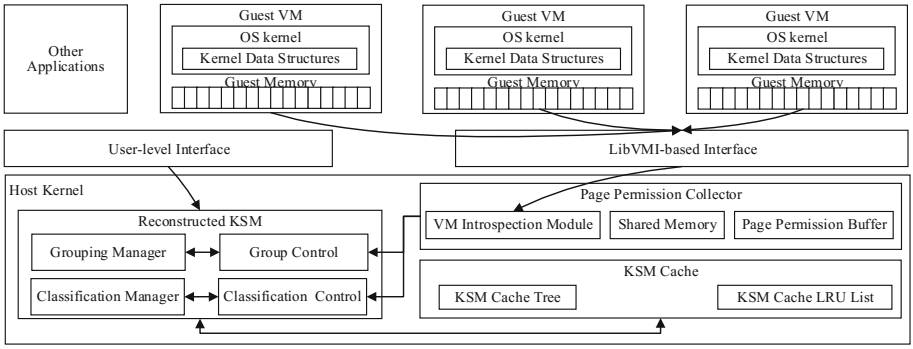

Fig. 3. The PCA architecture implemented with KSM

different sets to shrink looking up scope. While the division should meet the following two conditions: (1) pages with high probability to have same content should be divided into the same set, and vice versa. (2) the distribution of sharing potentials among sets should be balanced, because unbalanced division may still include a lot of otiose operations.

Grouping the VMs. The reason for grouping the VMs is the amount of redundant pages of inter-VM can be as low as 5%, but as high as 60% according to instances of the guest OS type and workloads [7]. For example, if several VMs run the same guest OS with same workloads their memory pages have a high possibility with equal content. Based on this fact, VMs can be divided into several groups. Accordingly, each global red-black tree is divided into multiple small trees, called G-trees. Thus, each group has a dedicated stable G-tree and unstable G-tree. A candidate page is only searched and compared within its G-trees. As the amount of tree nodes decreases, otiose operations are greatly reduced, while page sharing opportunities only have a slight variation. Besides, since the runtime of identification is saved, PCA can detect page sharing efficiently.

To support VM grouping, we mainly provide two ways. First, some options (user-level interfaces) are provided to customers. When renting VMs, they can use these interfaces to specify their VM's workloads or security level. To achieve this, we extend the Qemu parameters for VM creation. This symbiotic manner can explicitly assist the provider to group VMs more accurate, and is fit for confidential scenarios, in which each VM has a security level. Another way is grouping the VMs based on their potential sharing opportunities. In this way, VMs are proactively grouped during their startup by analyzing their disk image or obtaining their guest kernel structures via VMI-based detection module.

To support multiple groups, we modify the KSM algorithm to break the global trees into multiple G-trees, and extend some additional structures. During startup, each VM is initialized with a group tag (called GID) according to its attributes. In the current prototype, the GID is attached to the *mm_struct* structure of each VM process. For each group, we defined a *group_node*

Table 1. The page classifications in each group

Classification	Classification ID	Description
Unused pages	0x000	The pages is not used by anyone
Kernel read-only pages	0x001	Pages is only readable in kernel space
Kernel read-write pages	0x010	Pages is writable in kernel space
User read-only pages	0x011	Pages mapped only readable in user space
User read-write pages	0x100	Pages is mapped writable in user space

structure, which obtains this group's private information, such as GID, *ksm_scan* variable, tree root of G-trees and our introduced KSM cache etc. When function *ksm_madvise()* is invoked, the GID is retrieved for obtaining which group the VM belongs to. Thus, each VM's *mm_struct* structure is registered to the *ksm_scan* of its group. Thus, each group has its own registered *mm_struct* list. By doing so, every page to be scanned should reference its GID first to choose its corresponding G-trees. Since each page has its private *rmap_item* structure, we can speed up the search in G-trees by bounding page's GID into the address field of its *rmap_item* structure.

To save the cost of CPU and memory, we use a single thread as KSM (ksmd) to manage all groups. Since KSM is a linear scanner, to treat every group fairly every group has a weight value based on its sharing opportunities. At first, the weight is initialized by the ratio of the total memory of each group and total memory of mergeable. During runtime, the weight will be recounted by the shared pages of each group at the end of each round. The following formula is used to calculate the number of pages that should be scanned for each group, where N is the number of pages to be scanned per scan round, which is specified by the administrator.

$$\texttt{PerGroupScannedPages} = \texttt{N} \times \frac{\texttt{PerGroupSharedPages}}{\texttt{TotalSharedPages}} \tag{1}$$

Classifying the Advised Pages. We further discover that the sharing possibility between two pages of the same access permissions is more than those of not. For example, pages mapped to binary file will not be merged with pages of stack. Pages with most sharing are heap, shared library and page cache, which validates page access permission based classification is practicable. Thus, we further divide G-trees of each group into multiple classification trees (GC-trees) to improve KSM more effective.

To classify memory footprints of the VM into several classifications, we utilize VMI to obtain the page access permissions. During the page scan, PPC dynamically captures the access permissions for each candidate page, and pages with similar permissions are gathered into the same classification. In the current implementation, we defined five static page classifications as listed in Table 1. Therefore, each G-tree will be divided into five local GC-trees. That means each classification has a stable GC-tree and an unstable GC-tree. Since pages of

different classifications are cross-distribution, we need to obtain the access permissions in real time. For performance optimization, we provide a buffer in PPC for getting the access permission more rapidly. Each time access permissions of ten pages are obtained from the guest OS. In doing so, otiose operations are reduced, which will decrease the runtime to detect the same proportion shareable pages compared to traditional KSM. Thus, the scanner possesses more time to detect short-lived page sharing opportunities. In some case, new page sharing opportunities may be detected, which will be shown in Sect. 4.

Discussion. Except efficiency improvement, security is another advantage for grouping and classification. If the sender and receiver of a covert channel may be separated into different groups, thus the covert channels will be completely prevented without reducing the benefits of memory deduplication.

Since current KSM does not consider page access permission, the attacker's writable pages can be merged with pages of victim's application code to detect target vulnerable application [23,25]. However, this situation cannot happen in PCA, since PCA has classified pages into different classification based on their access permissions. Besides, page classification provides a prerequisite for defending intra-group covert channels.

3.3 KSM Cache

The extreme countermeasure for covert channels is to forbid the pages of the sender and receiver to be merged, like our VM grouping. But if the sender and receiver are arranged in a same group, what should we do? The best way is to distinguish the pages employed by covert channels from others. However, it is challenging and may be impossible. We adopt another direction that pages used by covert channels are allowed to merge, but with some additional efforts to reduce the difference of write access time between deduplicated and KSM pages. For this purpose, we introduce KSM cache, which disturbs the receiver to decode the transmitted data by mitigating the write latency of COW. Thus, PCA can mitigate or even prevent covert channels between VMs in any cases.

In our design, each group has a dedicated KSM cache that is used for storing the deduplicated pages. Note, we only cache pages with writable access permission used to build covert channels. When two identical pages are detected, we do not free the duplicated page immediately. Instead we move it from the stable GC-tree to KSM cache, as shown in Fig. 4. Thus we can find a copy of the deduplicated pages as quickly as possible when a COW occurs, without requiring a new copy of the KSM page.

To implement KSM cache, we have chosen two kinds of data structures to store the cached pages with low overhead: a red-black tree (kcache tree) and a LRU (least recently used) linked list (kcache list). Duplicated pages are managed by the kcache tree whose node includes PFN (page frame number), hva (host virtual address), child nodes, *mm_struct* of deduplicated pages, and a list containing the VM identity etc. During tree search and insertion, there is no checksum and byte-for-byte content comparisons. So it is lightweight compared

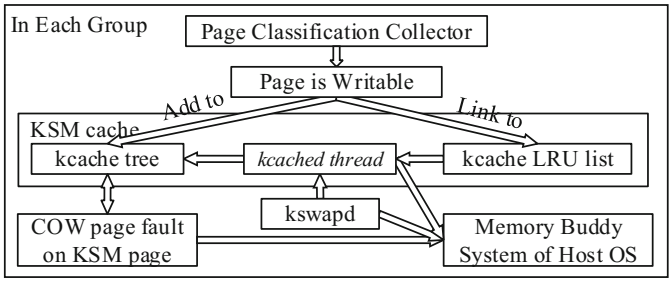

Fig. 4. The KSM cache architecture

to the KSM trees. The time complexity of the insert, delete and search operations is O(log n) in both average and worst-case. Besides, duplicated pages are maintained in a global kcache list for releasing them in a unified manner.

We have extended KSM to incorporate our KSM cache through a suite of hooks embedded in KSM. During KSM initialization, the KSM cache interfaces are mounted to the hooks. Like KSM, KSM cache is carried out as a kernel module that executes as a kernel thread, named kcached. Thus, since each group has its own data structures, kcached can manage the KSM cache group by group.

The aim of memory deduplication is to reduce the consumption of physical memory, so we must timely release the cached pages from the KSM cache. There is a tradeoff between the residence time of the duplicated page in KSM cache and the available memory freed by KSM. The longer the deduplicated pages reside in the KSM cache, the more security is ensured, and the less available memory is gained. To keep a better balance between available memory capacity and security, KSM cache is allowed to free its captured pages in three conditions. The first one is when a COW page fault occurs on a KSM page. In this case, we first find whether there is a match with the fault host virtual address and its VM identity in the KSM cache. If so, we directly map the matched page to this fault host virtual address. Thus, we avoid recreating a new copy. If not, the general COW process is followed. The second one is that we use the kcache list to periodically release a certain amount of outdated pages. Due to the natural characteristics of the LRU list, we will always process the oldest pages first. To better defend the covert channels, we add a random probability to the kcache list, in which outdated pages are freed randomly. We can therefore prevent the covert channels with justifiable overhead. Meanwhile keep the benefit of memory deduplication. The last one is when the system memory is insufficiency. We extend the memory deallocation to reclaim the pages of our KSM cache.

3.4 VMI-Based Memory Deduplication Scanner

Combing the above techniques, we implement a VMI-based memory deduplication scanner, using semantic information of VM's memory footprints. Figure 5 shows how the pages are organized in PCA. VMs' pages are first organized into

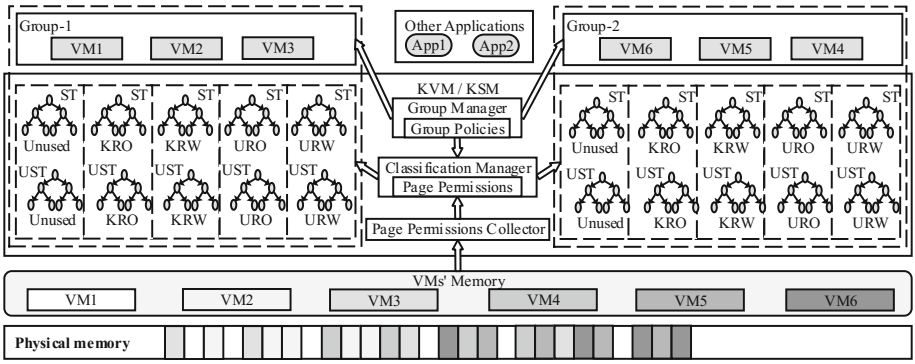

Fig. 5. Dividing VMs' pages into different sets

different groups by G-trees. In each group, pages are classified by GC-trees. During grouping and classification, PCA needs to get the guest OS internal state. For simplicity, we take advantage of LibVMI [20] to achieve these. To reduce interaction, the VMI tool is divided into two parts: a LibVMI-based user application and a kernel module. They interact with each other through a shared memory mapped to a character device file (see Fig. 3).

To arrange VMs into proper group, we get their OS type, version and workloads during their startup. To classify the pages of each group, the scanner requires to know the access permission of pages in guest OS. To obtain these, we firstly translate the candidate page's host virtual address into guest physical address (GPA) through the structure *kvm_memory_slot*. Then through VMI tool we can get the page structure array of guest OS, like *vmemmap* or *mem_map*. Thus, we can get the page structure in this array indexed by GPA. After that, we can learn whether this page is used via the _mapcount field. If the value is -1, this page isn't used. Otherwise, if the mapping field of the page structure is not empty, we can get the *anon_vma* or *address_space* structure. In this structure, we can get the access permissions. Otherwise, this page is used by kernel. According to its virtual field, we can know its access permissions. Note, except kernel code and module code, other pages are considered to be writable.

Figure 6 shows PCA's flowchart. In each periodic scan, the KSM thread gets a candidate page, it firstly gets the GID from the GM. Then obtains the CID (classification ID) from CM. Thus, it can locate its local trees. Like KSM the candidate page will be searched in its stable GC-tree and then in its unstable GC-tree. The difference is that when a match is found, the deduplicated page is inserted into the kcache tree and kcache list. After finishing each scan round, all unstable GC-trees need to be rebuilt in the next scan round. KSM cache also needs to be freed periodically based on its kcache list.

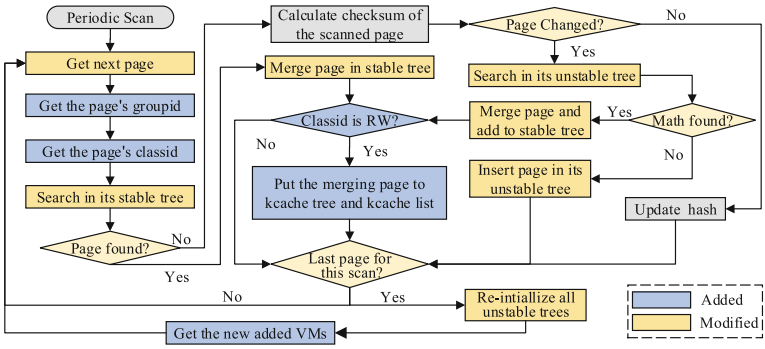

Fig. 6. The work-flow of PCA.

4 Evaluation

In this section, we firstly run the following benchmarks in VMs to show the effectiveness of PCA. Then we present its defense against covert channels and its overhead of preliminary evaluation. Our experiments are performed on a server with 4 2.6 GHz Intel Xeon E7 processors and 16 GB memory. Each processor has 8 physical cores. The server runs CentOS-6.5 virtualized by KVM with Linux-3.18.1. While VMs run CentOS-6.5 with Linux-2.6.38.

- Kernel Build: we compile the Linux kernel-3.18.1 in VMs. We begin this after the stable sharing opportunities are detected.
- Web Service: we run the Apache httpd server in VMs. We test the ab [38] benchmark with a local webpage.
- Bonnie++: we use bonnie++ [39] tool to do the hard disk benchmarking. The test file size is 2 times of the VM's memory.

4.1 Deduplication Effectiveness

We were particularly interested in seeing how does PCA work on merging pages compared to original KSM. To verify this, we have tested different workloads in PCA and KSM. In experiments, 4 VMs are booted, two in a group, the other two in different groups. To be able to measure the count of page comparisons and checksum calculations accurately, we modified some KSM functions to output the count per second. For simplicity, we don't show the inter-group test result, since the VMs in different groups will not be merged.

Figure 7 shows the page sharing opportunities. For Kernel Build and Apache, we can see that PCA is able to detect almost all page sharing opportunities, whic is more than 97% of KSM. The results prove that fine-granularity is a good hint for page classification, and page access permission is a better guide for page classification. However, there is still room for improvement, because pages with same content but with different access permission will be separated

into different classifications. For bonnie++, PCA detects more page sharing opportunities than KSM. It might because PCA can find many additional short-lived sharing opportunities that KSM is not able to detect. This further proves that fine-granularity classification can achieve higher accuracy, since the close contacts between page content and access permissions. We can also see that PCA can merge pages more quickly than KSM. Because PCA costs less comparison time in its classification trees, so that equal pages are identified earlier, thereby new sharing opportunities may be detected.

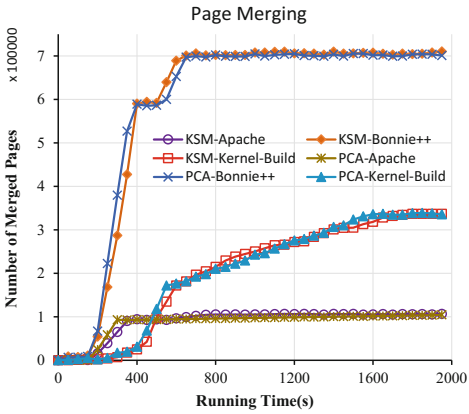

Fig. 7. Page sharing opportunities between 2 VMs with different workloads.

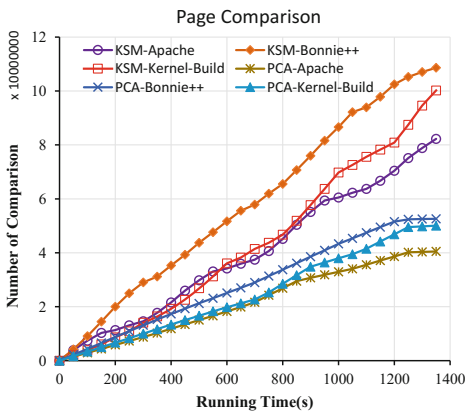

Fig. 8. The number of pages that need to be compared with different workloads.

Figures 8 and 9 respectively show the amount of page comparisons and checksum calculations. Figures show that KSM owns the largest number of page comparisons and checksum calculations due to its large global trees. While PCA has almost forty percent optimizations of KSM. Because in PCA multiple small classification trees contain less page nodes but have a much higher probability to have same content, which significantly reduces the otiose operations. Combining with Fig. 7, we can conclude that PCA has a available tradeoff between detecting page sharing opportunities and reducing otiose operations. The average reduction is about 40%. This results prove that based on page access permissions of guest OS PCA can accurately classify the pages.

4.2 Effectiveness of Covert Channel Defense

To evaluate the effectiveness of PCA against covert channels, we perform two types of experiment: sender and receiver in the same group and in different groups. We boot two virtual machines to deploy the sender and receiver process respectively. Each VM is configured with 1 VCPU and 512 MB memory. They load a 404 KB file (i.e. 101 4 KB pages) into memory. To ensure each page is

unique, the file is generated randomly by /dev/random. To guarantee that all pages are merged, we set the sleeping time to 5 s. In each type, we test five times. In each test, the sender transfer different data to the receiver, and in receiver we record the write access time of the 101 pages to decode the delivered data.

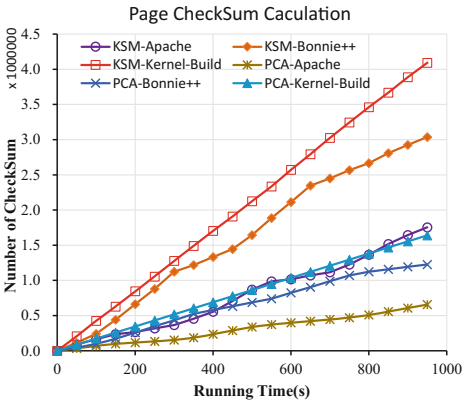

Fig. 9. The number of page checksums with different workloads.

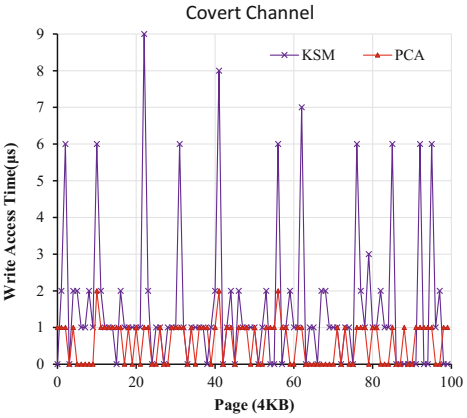

Fig. 10. Based on VM grouping and page classification to against covert channel.

Figure 10 shows the experimental results (VMs in the same group), which transmits a 101-bit data. To do this, the sender modifies the 3rd, 11th, 22th, 32th, 42th, 57th, 63th, 77th, 86th, 93th, 96th pages to encode a data. From Fig. 10, we can see that although different tests demonstrate different write access spikes, the write access time of sender-modified pages is always much less than the write access time of sender-unmodified pages in traditional KSM. While in PCA, there is no difference between sender-modified pages and sender-unmodified pages. This is because in PCA, once a page is merged, it will be added into the kcache tree immediately, and it will be hit in the kcache tree when the receiver decodes the data. Hence the write access time to the deduplicated pages has a negligible effect. Thus, the receiver cannot correctly recover the transmitted data.

4.3 Performance Overhead

The general trade-off of memory scanner is CPU utilization and memory consumption versus the security and efficiency. We have measured the CPU consumption of the scanner with some benchmarks by top measurements taken by per second. Figure 11 shows the average CPU utilization. We can see that PCA reduces the CPU overhead compared with original KSM. The highest impact is kernel build workload about 6%. Since kernel build workload has high memory usage during runtime, it exists vast otiose operations. However, PCA can effectively reduce these otiose operations. Thus the CPU overhead is reduced. While apache has low memory usage, so its overhead is reduced relatively less.

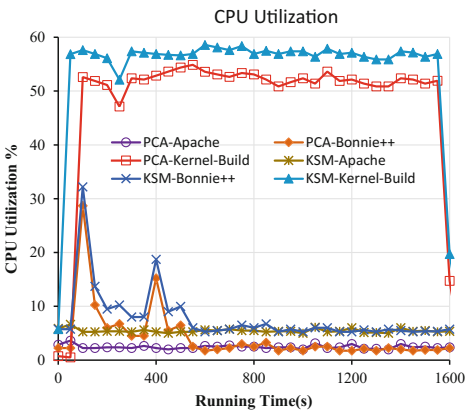

Fig. 11. The CPU utilization of KSM and PCA with different workloads.

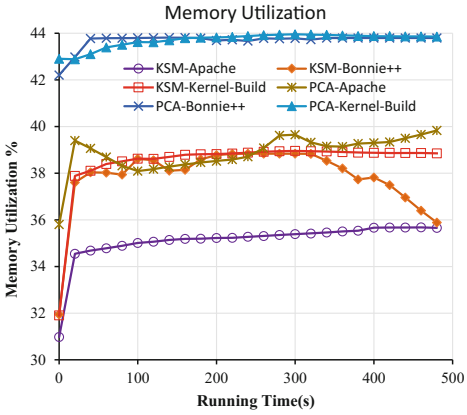

Fig. 12. The memory consumption of KSM and PCA with different workloads

Although the VMI used in our work may lead to a certain of performance loss. But the overhead is very little. Further, PCA provides a buffer to store access permissions, which reduces the number of VMI invocation. However, inserting duplicated pages into kcache tree is very cheap.

The only additional memory space is used by storing the page permission, the kcache tree, the kcache list, some structures and a few locks to sequential access shared data structures. From Fig. 12, we can see that PCA only consumes little memory during VMs execution. This overhead comes from KSM cache, because some key fields are recorded into the kcache tree nodes.

5 Related Work

Page Sharing. Limited main memory size has become one of the major bottlenecks in virtualization environment, and as an efficient approach to reduce server memory requirement, memory deduplication thus has attracted a large body of work on it. Disco [1] was the first system to implement page sharing on code pages with assistance from guest OS. CBPS requires no assistance from the guest OS, and was firstly implemented in VMware ESX server [2]. Then, CBPS was introduced in Xen [3] and KVM [4] to increase the memory density of VMs. But they can only merge anonymous pages, as the host regards the guests' memory as anonymous memory due to the semantic gap.

To acquire more sharing opportunities, Difference Engine [13] and Memory Buddies [14] proposed sub-page sharing, which not only explores the potential of same pages but also similar pages. Satori [15] employed sharing-aware virtual disks to find short-lived sharing opportunities. KSM++ [16] found pages in host cache are strong sharing candidates and preferential scan them can exploit short-lived sharing opportunities. Based on this, XLH [17] generates page hints in the host's virtual file system for merging them earlier. Singleton [18] combined the

host and guest double-caching into an exclusive cache. However, they didn't consider reducing needless overhead of KSM. While we argue that page sharing needs to consider classification for reducing otiose operations.

Empirical studies [7,8] show CBPS can achieve memory savings up to 50% on I/O intensive workloads. But CBPS has higher runtime overhead by otiose comparison. To solve this, an adaptive policy [9] was proposed to obtain more sharing opportunities but with little CPU overhead. Sindelar et al. [10] proposed two hierarchical sharing models through sharing-aware algorithms without heavy CPU overhead. CMD [11] proposed a classification-based approach with a dedicated hardware. While IBM's AMD [12] generates a signature for each physical page to avert page comparison. PageForge [19] firstly proposed a hardware-based design for same-page merging that effectively reduces the CPU overhead. Except the performance improvement, PCA also concerns security.

Covert Channels. Recent research efforts [23–26] have mentioned the potential threat of covert channels based on memory deduplication. However, in their context, the covert channel is used primarily for leaking information. Xiao et al. [27] firstly constructed a rough covert channel to transmit information between two VMs. In an ideal situation, the bit rate of such covert channel can be around 1kbps. However, its bit errors make it impractical in reality due to uncertain merging time. For this, Rong et al. [28] proposed a robust communication protocol for high-speed transmission and reliability. Xen's event channel can be used to conduct covert channels [29,30], which has bend demonstrated in Amazon EC2 [31]. Moreover, Gruss et. al [32] proposed the JavaScript based covert channel to collect private information in sandboxes. Our work aims to implement a defense scheme to these attacks.

Cloud providers can tackle covert channels through either preventative or detective approach, since they are much more resourceful. Amazon EC2 provides a dedicated instances service [33], in which different tenants' VMs do not share physical hardware. While the significant service charge reduces its attractiveness. Also, Wu et al. [31] advised the cloud provider to define a policy, which only allows two tenants to be shared in each server. But the tenant's neighbor is predetermined. These approaches may mitigate covert channels, but the memory utilization is low. In contrast, PCA has a low cost, but allows all tenants to share system memory.

Kim et al. [34] proposed a group-based memory deduplication scheme that aims to provide performance isolation on each single server. Deng et al. [35] also proposed a similar memory sharing mechanism, in which the global KSM thread is divided into per-group threads. Also, Ning et al. [36] proposed a covert channel defense mechanism based on VM grouping. SEMMA [37] provides a security architecture for performance isolation and security assurance. All of the above work only provides inter-group protection. Further, they did not consider otiose operations. Our work not only prevent covert channels in both inter-group and intra-group, but also reduce otiose operations to improve performance.

6 Conclusion

Memory deduplication is an important feature in modern hypervisors. However, it has otiose operations, and induces covert channels. In this paper, we put forward a highly efficient and secure memory deduplication approach called page correlation aggregation (PCA). PCA achieves two important objectives. First, it significantly reduces otiose operations. Meanwhile it speeds up the identification of page sharing and boosts the sharing opportunities. Second, it effectively mitigates or even prevents covert channels. We have implemented our design and evaluated it on KVM with different workloads. The experimental results show that PCA is effective, efficient and practical. In the future, we will plan to investigate the combination of PCA and balloon technique for efficiently managing the memory in the cloud. Also, we are interested in research logging mechanism with machine learning to detect the covert channels.

Acknowledgments. We would like to thank the anonymous reviewers for their constructive suggestions. This work was supported by the National Key Research and Development Program of China under grant No. 2016YFB0801002.

References

1. Bugnion, E., Devine, S., Govil, K., et al.: Disco: running commodity operating systems on scalable multiprocessors. ACM Trans. Comput. Syst. **15**(4), 412–447 (1997)
2. Waldspurger, C.A.: Memory resource management in VMware ESX server. ACM SIGOPS Oper. Syst. Rev. **36**(1), 181–194 (2002)
3. Kloster, J.F., Kristensen, J., Mejlholm, A., et al.: On the feasibility of memory sharing: content-based page sharing in the Xen virtual machine monitor (2006)
4. Arcangeli, A., Eidus, I., Wright, C.: Increasing memory density by using KSM. In: Proceedings of the Linux Symposium, pp. 19–28 (2009)
5. Srivastava, A., Giffin, J.: Tamper-resistant, application-aware blocking of malicious network connections. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 39–58. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87403-4_3
6. Payne, B.D., Carbone, M., Sharif, M., Lee, W.: Lares: an architecture for secure active monitoring using virtualization. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 233–247 (2008)
7. Chang, C.R., Wu, J.J., Liu, P.: An empirical study on memory sharing of virtual machines for server consolidation. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications, IEEE, pp. 244–249 (2011)
8. Barker, S.K., Wood, T., et al.: An empirical study of memory sharing in virtual machines. In: Proceedings of the Annual Technical Conference, pp. 273–284 (2012)
9. Rachamalla, S., Mishra, D., Kulkarni, P.: All page sharing is equal, but some sharing is more equal than others (2013). http://www.cse.iitb.ac.in/internal/techreports/reports/TR-CSE-2013-49.pdf
10. Sindelar, M., Sitaraman, R.K., Shenoy, P.: Sharing-aware algorithms for virtual machine colocation. In: Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures, pp. 367–378 (2011)

11. Chen, L., Wei, Z., Cui, Z., et al.: CMD: classification-based memory deduplication through page access characteristics. In: Proceedings of the ACM International Conference on Virtual Execution Environments, vol. 49, no. 7, pp. 65–76 (2014)
12. Ceron, R., Folco, R., Leitao, B., et al.: Power systems memory deduplication. In: IBM Redbooks (2012). http://www.redbooks.ibm.com/abstracts/redp4827.html
13. Varghese, G., Voelker, G.M., Vahdat, A., et al.: Difference engine: harnessing memory redundancy in virtual machines. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, pp. 309–322 (2008)
14. Wood, T., Tarasuk-Levin, G., Shenoy, P., et al.: Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In: Proceedings of the International Conference on Virtual Execution Environments (2009)
15. Mios, G., Murray, D.G., Hand, S., Fetterman, M.A.: Satori: enlightened page sharing. In: Proceedings of the Annual Technical Conference, USENIX, pp. 1–14 (2009)
16. Miller, K., Franz, F., Groeninger, T., et al.: KSM++: using I/O-based hints to make memory-deduplication scanners more efficient. In: Proceedings of the ASPLOS Workshop on Runtime Environments, Systems, Layering and Virtualized Environments (2012)
17. Miller, K., Franz, F., Rittinghaus, M., Hillenbrand, M., Bellosa, F.: XLH: more effective memory deduplication scanners through cross-layer hints. In: Proceedings of the Annual Technical Conference, USENIX, pp. 279–290 (2013)
18. Sharma, P., Kulkarni, P.: Singleton: system-wide page deduplication in virtual environments. In: Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing, ACM, pp. 15–26 (2012)
19. Skarlatos, D., Kim, N.S., Torrellas, J.: Pageforge: a near-memory content-aware page-merging architecture. In: Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture, ACM, pp. 302–314 (2017)
20. LibVMI tool. http://libvmi.com/
21. Irazoqui, G., Inci, M.S., Eisenbarth, T., Sunar, B.: Wait a minute! A fast, cross-VM attack on AES. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 299–319. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11379-1_15
22. Irazoqui, G., Eisenbarth, T., Sunar, B.S.: $ A: a shared cache attack that works across cores and defies VM sandboxing-and its application to AES. In: Proceedings of the Security and Privacy, pp. 591–604. IEEE (2015)
23. Ristenpart, T., Tromer, E., Shacham, H., et al.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the ACM Conference on Computer and Communications Security, pp. 199–212. ACM (2009)
24. Suzaki, K., Iijima, K., Yagi, T., et al.: Software side channel attack on memory deduplication. In: Symposium on Operating Systems Principles, Poster session (2011)
25. Suzaki, K., Iijima, K., et al.: Memory deduplication as a threat to the guest OS. In: Proceedings of the Fourth European Workshop on System Security. ACM (2011)
26. Suzaki, K., Iijima, K., Yagi, T., et al.: Implementation of a memory disclosure attack on memory deduplication of virtual machines. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **96**(1), 215–224 (2013)
27. Xiao, J., Xu, Z., Huang, H., et al.: Security implications of memory deduplication in a virtualized environment. In: Proceedings of the Dependable Systems and Networks, pp. 1–12. IEEE (2013)

28. Rong, H., Wang, H., Liu, J., et al.: WindTalker: an efficient and robust protocol of cloud covert channel based on memory deduplication. In: Proceedings of the Big Data and Cloud Computing, pp. 68–75. IEEE (2015)
29. Shen, Q., Wan, M., Zhang, Z., Zhang, Z., Qing, S., Wu, Z.: A covert channel using event channel state on Xen hypervisor. In: Qing, S., Zhou, J., Liu, D. (eds.) ICICS 2013. LNCS, vol. 8233, pp. 125–134. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02726-5_10
30. Wu, J.Z., Ding, L., Wang, Y., et al.: Identification and evaluation of sharing memory covert timing channel in Xen virtual machines. In: Proceedings of the Cloud Computing, pp. 283–291. IEEE (2011)
31. Wu, Z., Xu, Z., Wang, H: Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In: Proceedings of USENIX Security Symposium, pp. 159–173 (2012)
32. Gruss, D., Bidner, D., Mangard, S.: Practical memory deduplication attacks in sandboxed Javascript. In: Pernul, G., Ryan, P.Y.A., Weippl, E. (eds.) ESORICS 2015. LNCS, vol. 9326, pp. 108–122. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24174-6_6
33. Amazon Web Services. Amazon EC2 dedicated instances. http://aws.amazon.com/dedicated-instances/
34. Kim, S., Kim, H., Lee, J.: Group-based memory deduplication for virtualized clouds. In: Alexander, M., et al. (eds.) Euro-Par 2011. LNCS, vol. 7156, pp. 387–397. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29740-3_44
35. Deng, Y., Hu, C., Wo, T., et al.: A memory deduplication approach based on group in virtualized environments. In: Proceedings of the Service Oriented System Engineering, pp. 367–372. IEEE (2013)
36. Ning, F., Zhu, M., You, R., et al.: Group-based memory deduplication against covert channel attacks in virtualized environments. In: Proceedings of the Trustcom/BigDataSE/I SPA, pp. 194–200. IEEE (2016)
37. Chen, X., Chen, W., Long, P., et al.: SEMMA: secure efficient memory management approach in virtual environment. In: Proceedings of the Advanced Cloud and Big Data, pp. 131–138. IEEE (2013)
38. Apache http server benchmarking tool. http://httpd.apache.org/docs/2.2/programs/ab.html
39. Bonnie++ file system benchmark. https://www.coker.com.au/bonnie++/