Faking deduplication to prevent timing side-channel attacks on memory deduplication

Jens Lindemann
Department of Informatics
University of Hamburg, Germany
jens.lindemann@uni-hamburg.de

***Abstract*—Content-based memory deduplication offers potential for saving physical memory by merging identical virtual pages. However, it also opens up a timing side-channel that allows attackers to probe whether an identical copy of one of their pages exists elsewhere on the system. This allows, for example, to find out what exact version of an application is being executed in another VM, and can also facilitate other side-channel attacks, e. g. some Rowhammer-based attacks. This paper presents FakeDD, a modification to Linux KSM, which uses fake deduplication as a countermeasure against such attacks that equalises the write times to duplicate and unique pages. Nevertheless, it still allows to make use of the memory savings offered by deduplication. FakeDD is available as open-source. It is evaluated with regard to its effectiveness and performance overhead. The results indicate that it can effectively prevent side-channel attacks based on the write time differences between duplicate and unique pages and that the additional performance overhead is smaller than that already incurred due to the standard KSM implementation and may thus be an acceptable trade-off for the additional security.**

1. Introduction

Sharing computing resources between different services and users allows for an efficient use of these resources. For example, there is no need to run dedicated server hardware to host a small website. Instead, it can be placed on a server that also hosts websites or even other server software for other users. In this scenario, the server operator will normally want to make sure that each user's domain is isolated from other users' domains, i. e. they should not be able to interfere with or gain information about activities or data of other users. Isolated domains can be realised by different means, e. g. containers or virtual machines (VMs).

Despite these efforts at isolation, resource sharing does, however, increase the risk of successful attacks. A malicious user may be able to breach the isolation and interfere with the domain of another user on the same host. On the one hand, components such as the host operating system or hypervisor may be vulnerable. On the other hand, side-channel attacks could be used to deduce information about other, supposedly isolated, domains.

Introducing additional mechanisms aimed at optimising the resource utilisation can further increase the risk of side-channel attacks between isolated domains on a host. One such mechanism is memory deduplication, which aims to save physical memory by referencing identical virtual memory pages to a single physical memory page. There are various implementations of memory deduplication, one of them being Linux kernel same-page merging (KSM). It regularly scans for such pages and deduplicates them across the boundaries of VMs. This gives rise to a side-channel, as in KSM and similar content-based implementations write operations to a deduplicated page take longer than to a non-deduplicated page due to a copy-on-write operation being performed. This allows an attacker to infer whether a memory page is present within another VM on a host [1].

The main contribution of this paper is FakeDD, a modified implementation of Linux KSM that eliminates the write timing side-channel caused by memory deduplication by means of fake deduplication. This is achieved by copying a page that is being written to not only if it is actually deduplicated, but also if it is unique, but would be eligible for deduplication if there was a second, identical virtual page in memory. FakeDD is provided as an open-source patch to the Linux kernel's implementation of KSM on GitHub[1]. Its effectiveness in eliminating the side-channel and its retention of memory savings potential are evaluated, as is its influence on performance compared to a stock kernel with and without KSM as well as VUsion [2], which similarly aims to eliminate side-channels, albeit using the even more aggressive technique of copy-on-access.

The remainder of this paper is structured as follows: Section 2 discusses background and related work. Section 3 discusses the attacker model underpinning the defence mechanism. The standard KSM implementation is compared to the modified implementation in Section 4. In Section 5, FakeDD is evaluated with regards to its effectiveness and performance, before Section 6 concludes the paper and gives an outlook on future work.

1. https://github.com/jl3/FakeDD

2. Background and related work

In this section, background information is explained and related work discussed. Section 2.1 first introduces the concept of memory deduplication. Section 2.2 explains side-channel attacks based on memory deduplication, before Section 2.3 discusses countermeasures against such attacks.

2.1. Memory deduplication

Memory deduplication is implemented in many operating systems and hypervisors to save physical memory. An operating system organises a computer's memory as a set of memory pages $\mathcal{M}$, with the typical size of a normal (non-huge) page $p_i \in \mathcal{M}$ being 4 KiB. Consequently, a page resident in physical memory will consume 4 KiB without deduplication. Memory deduplication tries to identify sets D_i of multiple identical pages where $\forall p_j, p_k \in D_i : (p_j, p_k \in \mathcal{M}) \wedge (p_j = p_k)$. It then removes all but one page $p_m \in D_i$ from physical memory and updates the memory mappings of all other pages $\forall p_j \in \mathcal{M} : (p_j = p_m) \wedge (j \neq m)$ to point to p_m instead. Subsequently, when a page $p_i \in D_i$ is to be changed, a copy-on-write operation is triggered, i. e. the page is copied to a new physical page. This allows it to be modified without affecting the other copies of the page.

In most implementations, the identification of sets of identical pages is implemented by regularly scanning the memory for such pages. While this consumes computing resources, it ensures that identical pages are found even if they were loaded from different sources and/or only became identical during the run-time of an application. Alternatively, it is, however, also possible to identify identical pages when originally mapping the pages into virtual memory based on where they were loaded from, i. e. if the same page gets mapped from the same file twice, these pages will be identical and can thus be deduplicated, unless they are later written to. A similar concept underlies the shared memory mappings that operating systems create when shared libraries are used by multiple applications: Where pages are loaded from an identical file location and do not need to be adapted to individual applications, only one physical memory page is used.

In Linux, the Kernel Same-page Merging (KSM) mechanism is used for deduplication [3], which is described in more detail in Sect. 4.1. Its main purpose is to deduplicate the memory of VMs created through the Kernel-based Virtual Machine (KVM) hypervisor, which does not have a separate deduplication mechanism.

VMWare uses its own deduplication mechanism [4], which the author refers to as 'content-based' because it tries to identify candidates for deduplication based on their contents by regularly scanning the memory for identical pages – just as KSM does. However, current versions of VMWare only deduplicate identical pages within the memory of a VM by default [5], i. e. no pages are deduplicated across the boundaries of VMs. Unless this setting is changed, this implies that cross-VM side-channel attacks based on the write time differences induced by memory deduplication are not possible, but also that memory savings are reduced.

Xen includes supports for memory deduplication under the denomination of 'memory sharing'. As of release 4.18, this is still under experimental status [6], despite the code file implementing it originally stemming from 2009 [7]. The Xen hypervisor no longer comes with any functionality to identify pages that should be deduplicated, though: The originally included `memshr` library, which served to deduplicate pages at load-time when pages are loaded multiple times from the same disk image [8], was removed in 2020 [9]. Difference Engine [10] is a proposal to add a content-based scanning mechanism for the identification of identical pages. In addition to completely identical pages, it aims to also deduplicate partly-identical pages by only storing information about the differences of a page to a similar page that is stored in physical memory.

2.2. Side-channel attacks based on memory deduplication

A problem with memory deduplication is that when a deduplicated virtual page is modified, the changes can no longer be written directly to physical memory, as this would also affect all other virtual pages mapped to the physical page. Deduplicated pages are therefore marked copy-on-write. This implies that when one of the virtual pages is to be modified, the contents of the page are first copied to a new physical page. Only afterwards are the modifications written to the copied page.

The copying process takes time to complete, which prolongs the write operation for deduplicated pages. On the one hand, this can affect application performance, which can be an acceptable trade-off for the memory savings achieved.

On the other hand, however, this gives rise to side-channel attacks that can break the isolation between the domains of different users, e. g. separate virtual machines. As modifying a deduplicated page takes longer than modifying a non-deduplicated page, an attacker can probe whether one of their pages was deduplicated [1]. While this does not allow an attacker to read data from the memory of other users, it allows an attacker to check whether a page containing specific data is present elsewhere on the host. To do this, they need to fill a page with the data that they wish to probe for, as well as wait for the deduplication mechanism to find any potential duplicate pages and deduplicate the pages. Afterwards, they can measure the time it takes to modify the page: If this takes relatively long, this indicates that an identical page is present on the host. Otherwise, this indicates that their page was unique, i. e. there is no identical page elsewhere on the host. Note, however, that such attacks only work at the granularity at which deduplication is performed, i. e. on full pages for KSM and other common implementations of content-based deduplication. Thus, attackers cannot directly probe for the presence of small amounts of data, such as a password, without also knowing (or brute-forcing) the rest of the page.

This side-channel can be used for various purposes and can reveal confidential information from the domain of a user. Suzaki et al. [1] were the first to describe an attack that exploits the write time differences between deduplicated and non-deduplicated pages caused by KSM. Their attack uses it to detect applications running in a co-resident VM. Owens and Wang [11] demonstrate an attack that can detect which operating system is being executed within a co-resident VM on a VMWare ESXI host. Lindemann and Fischer [12], [13] demonstrate that it is possible to create signatures that can be used to efficiently detect the exact version of applications executed in co-resident VMs down to the patch level.

Instead of trying to detect executed applications, other attacks target the address space layout randomisation (ASLR) techniques of operating systems. Barresi et al. [14] demonstrate an attack that can reveal the randomised base address of applications executed in co-resident Windows and Linux VMs. Bosman et al. [15] demonstrate a JavaScript-based attack on Windows 8.1 and 10 end-user operating systems exploiting memory deduplication through the Microsoft Edge browser. Their attack can reveal the randomised address of the browser's heap and code pointers. Combined with a Rowhammer attack [16], the attack manages to read arbitrary data from the memory of the victim computer.

Memory deduplication can also be used to establish a covert channel through which two co-resident virtual machines can communicate, as shown by Xiao et al. [17]. Additionally, they show that memory deduplication can be used to monitor the integrity of a VM's kernel from the outside.

Furthermore, there are side-channel attacks that are not based on the write time differences between deduplicated and non-deduplicated pages, but that nevertheless rely on memory deduplication or that are at least facilitated by it. For example, Irazoqui et al. [18] describe an attack that can detect what version of a cryptographic library is being used in a co-resident VM through a Flush+Reload attack. It measures the reload time of functions characteristic to the library (version). The attacker flushes the cache and later tries to reload the function: Reloading will be faster if another VM used the function in the meantime. The attack requires the memory page with the targeted function to be deduplicated between the VMs of the attacker and the victim.

If an attacker wanted to perform a targeted attack on a specific VM, this would require them to achieve co-residence with that VM. Research indicates that this is realistic even in large public cloud environments [19], [20]. While this effect has diminished significantly [20], it used to be possible to achieve co-residence with a VM in Amazon EC2 by launching VMs close to the launch time of a target VM [21].

Concerns of data leaking between VMs has lead to the feature being disabled in some environments at the cost of not being able to make use of its memory savings potential. For example, VMWare disabled deduplication between different VMs in the default configuration [5]. Google does not use KSM with their KVM VMs [22]. Amazon also stated to Irazoqui et al. [23, pp. 14–15] that their public cloud products do not use deduplication.

2.3. Countermeasures against memory deduplication side-channel attacks

Different countermeasures against side-channel attacks exploiting the timing differences caused by memory deduplication have been proposed. As described in the previous subsection, concerns about data leaks have lead some vendors to disable memory deduplication. While this is indeed an effective and simple countermeasure, it also eliminates all potential benefits of deduplication in form of memory savings.

The proposal most closely related to that presented in this paper is VUsion [2], which also aims to eliminate side-channel attacks by modifying the handling of deduplicated pages. The most important difference is that VUsion employs copy-on-access instead of copy-on-write. This has the advantage of also countering attacks that rely on pages being deduplicated, but that can be triggered by read operations, such as some variants of the Rowhammer attack, e. g. the one described by Bosman et al. [15]. This does, however, come at the cost of also disturbing read operations: Pages not only have to be copied when written to, but also when read from. Also, note that this does not eliminate all Rowhammer attacks, as deduplication merely facilitates such attacks – the original Rowhammer attack [16] does not rely on pages being deduplicated. Furthermore, this can prevent what Oliverio et al. [2] refer to as a 'page sharing' attack, in which an attacker can detect page sharing changes through a cache-based side-channel attack. This attack assumes that the victim can be triggered to access a page, i. e. it requires some level of control over the VM owning the (suspected) second instance of a page.

A further difference is that VUsion also copies the contents of a page to a new physical page when it is deduplicated and/or marked as copy-on-access. On the one hand, this counters the Flip Feng Shui attack [24], which is based on Rowhammer. On the other hand, this also ensures that pages being marked copy-on-access will have a chance of being assigned a different page colour, irrespective of whether they are actually being deduplicated. This can prevent what Oliverio et al. [2] refer to as a 'page colouring' attack, through which an attacker can determine via a cache-based side-channel attack whether a page has been moved, which would indicate it having been deduplicated. Note, however, that such an attack is relatively complex, especially if an attacker wants to probe for the existence of many pages, as would be the case if they were trying to determine which version of an application is present [12]: An attacker would have to first find eviction sets for all page colours, identify the colour of all pages and then monitor for changes of the colour of all pages by separate Prime+Probe attacks.

Finally, VUsion employs a more complex algorithm to determine whether a page is stable enough to be considered for deduplication or being marked as copy-on-write ('working set evaluation'). While KSM and FakeDD only require pages to remain unchanged between two scans, VUsion aims to also determine whether the pages have been accessed at all. This serves to avoid excessive page faults on read operations, but is more restrictive in terms of pages that can be deduplicated.

Instead of eliminating the side-channel itself, proposals from the field of co-location resistant VM placement could also help to prevent side-channel attacks. This field explores how to assign newly created VMs to hosts in a way that reduces the likelihood of attacks between VMs on the same host by reducing the chance of a benign user's VM being placed on the same host as a VM of an attacker. Many different placement strategies aimed at increasing co-location resistance have been proposed. They base their placement decisions on various parameters, such as choosing hosts where at least one user already present has been encountered before [25], the proportion of users on a host that were previously encountered [26] or only placing VMs of a single user on a host [27].

Finally, another approach at thwarting side-channel attacks could be detecting them. One proposal to this end is HexPADS [28]. It aims to detect the CAIN attack by Barresi et al. [14] by means of monitoring the number of page faults and cache misses. Similarly, Paundu et al. [29] try to detect cache-based side-channel attacks by analysing KVM events gathered using the `ftrace` tool.

3. Attacker model

The attacker is assumed to be in control of a VM on the same host as the victim. The host is assumed to be using a content-based memory deduplication mechanism, i. e. one that regularly scans the memory of all guest VMs for identical memory pages and deduplicate these, e. g. Linux KSM. In particular, it is assumed to also deduplicate pages that belong to different VMs owned by different users. The attacker is assumed to be able to write arbitrary memory pages to the memory of their own VM, overwrite these pages later and perform an accurate measurement of the time it takes to complete a memory write within the VM. They do not, however, have control over anything else than their own VM, i. e. they cannot perform any actions within the host OS or the victim VM. The attacker's goal is to determine whether a memory page with specific contents exists in another VM on the host, i. e. whether all 4 096 bytes in the page match their 'guess'. Their memory resources are assumed to be restricted by their VM, i. e. they can only load a limited number of pages at a time. We do not assume the attacker to be limited in terms of time, however: Both the attack as well as the victim VM are assumed to remain on the host indefinitely. This effectively makes the memory restriction irrelevant even for attackers wanting to probe for the presence of many different pages. Similarly, the attacker's computational resources will also be bound by their VM, but this is irrelevant, as the attack neither consumes a lot of processing power nor is it relevant how long the attack takes.

4. Modifying Linux KSM

FakeDD was implemented as a modification to the content-based memory deduplication mechanism of the Linux kernel, KSM. It is available as an open-source patch to the Linux kernel on GitHub[2] alongside some of the tools used for evaluation. To explain how FakeDD works, this section first explains the standard KSM implementation in Section 4.1, before describing the modifications made for FakeDD in Section 4.2.

4.1. Implementation details of standard KSM

Linux kernel same-page merging (KSM) is the content-based memory deduplication mechanism of the Linux kernel. It operates on pages specifically marked as mergeable (`MADV_MERGEABLE`) by calling the `madvise` function [30]. The kernel automatically does this for the memory of VMs created through KVM. Theoretically, other applications could also make use of memory dedupication by marking their memory as mergeable, but desktop applications do not normally do this [31]. If an application running on the host operating system were to mark its pages as mergeable, this would imply that these pages would also be deduplicated with pages of VMs. Thus, attackers could also probe for the existence of these pages within a VM, unless countermeasures were deployed. All mergeable pages on a host are regularly scanned by KSM. If identical pages are found, they are deduplicated.

Figure 1 gives an overview of the standard KSM scanning procedure. After initialising the stable and unstable trees, KSM starts a scanning loop. This scans all pages in the physical memory that are marked as mergeable and are not already in one of the trees.

For each of these pages, it first checks whether the page is already in the stable tree, which is a red-black tree containing information about all pages currently marked as copy-on-write by KSM, i. e. pages which have previously been deduplicated. Note, however, that this does not imply that they will necessarily *still* be deduplicated – if just one unmodified copy of a deduplicated page remains in memory, it will stay marked as copy-on-write until modified. If a copy of the page is found in the stable tree, the scanned page is merged with this.

If the page is not in the stable tree, KSM checks whether it has changed since it was last scanned by comparing its checksum to the value stored for the page. If this is not the case, it updates the stored checksum and ignores the page for this pass. Otherwise, it searches the unstable tree for an identical page. The unstable tree is another red-black tree and contains all pages that have been scanned so far during a memory pass. If a duplicate is found on the unstable tree, it is removed from it, the two pages are merged and inserted into the stable tree. If there is no identical page, the scanned page is added to the unstable tree, which concludes the scan of the page.

2. https://github.com/jl3/FakeDD

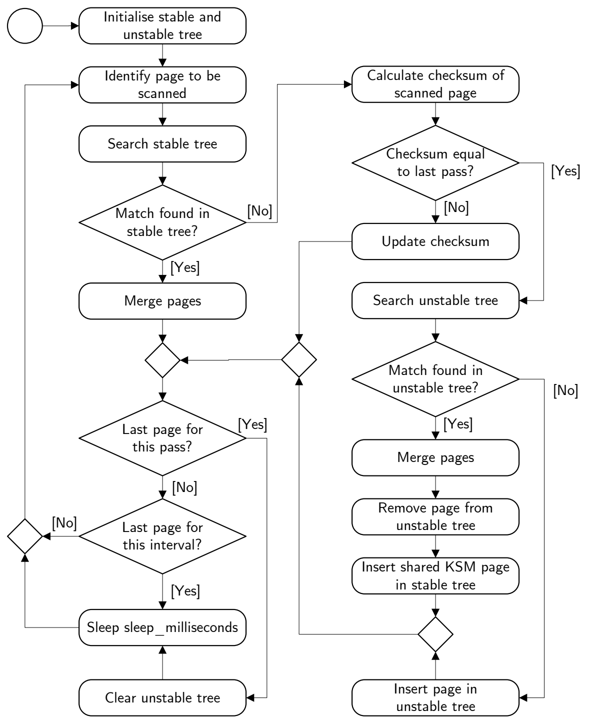

Initialise stable and unstable tree
Identify page to be scanned
Search stable tree
Match found in stable tree?
[No]
[Yes]
Merge pages
Last page for this pass?
[Yes]
[No]
Last page for this interval?
[No]
[Yes]
Sleep sleep_milliseconds
Clear unstable tree
Calculate checksum of scanned page
Checksum equal to last pass?
[Yes]
[No]
Update checksum
Search unstable tree
Match found in unstable tree?
[No]
[Yes]
Merge pages
Remove page from unstable tree
Insert shared KSM page in stable tree
Insert page in unstable tree

Figure 1: Overview of the standard KSM scanning procedure (based on flowchart describing the original KSM scanner by Arcangeli et al. [3, Fig. 4], modified to show sleep interval introduced in later kernel versions)

After scanning a page, KSM proceeds to scan the next page unless the KSM configuration requires it to pause before doing so. KSM allows to configure the number of pages to scan in a time interval by writing it to the pseudo-file `/sys/kernel/mm/ksm/pages_to_scan`. Likewise, the length of the interval can also be configured by writing to the pseudo-file `sleep_millisecs`. After KSM has scanned `pages_to_scan` pages back to back, it will pause `sleep_millisecs` milliseconds before scanning the next batch of pages. Once all mergeable pages have been scanned in a pass, KSM clears the unstable tree and begins another pass of the memory, i. e. it scans all mergeable pages again.

4.2. Implementation of FakeDD in KSM

The implementation of FakeDD is based on Linux KSM. An overview of the modified KSM implementation is shown in Figure 2. The FakeDD version of KSM no longer uses the unstable tree. As pages are no longer inserted into the unstable tree, the unstable tree also no longer needs to be searched for a page matching the currently scanned page.

Where a page would be merged with a page in the unstable tree or alternatively added to the unstable tree in the standard KSM implementation, it is now instead marked as copy-on-write on its own and directly added to the stable tree. This implies that a page is not immediately marked copy-on-write on being mapped or modified. Instead, this only happens when it would normally become eligible for deduplication, i. e. only after it has been passed twice by the scanner and has not been modified since its last scan. Furthermore, like in the standard KSM implementation, the page must be marked as mergeable. Once these prerequisites are met, the page is marked as copy-on-write irrespective of whether another identical page exists in the computer's memory.

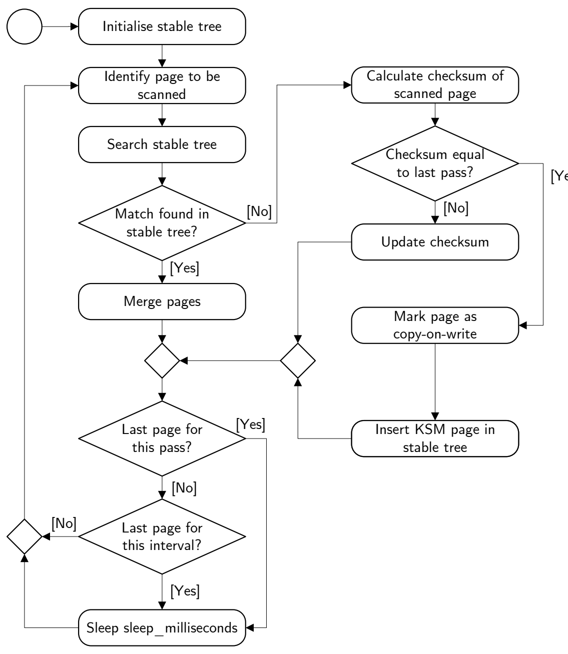

Initialise stable tree
Identify page to be scanned
Search stable tree
Match found in stable tree?
[No]
[Yes]
Merge pages
Last page for this pass?
[Yes]
[No]
Last page for this interval?
[No]
[Yes]
Sleep sleep_milliseconds
Calculate checksum of scanned page
Checksum equal to last pass?
[Yes]
[No]
Update checksum
Mark page as copy-on-write
Insert KSM page in stable tree

Figure 2: Overview of the FakeDD KSM implementation

Note that the modified KSM implementation leaves a residual side-channel open: An attacker can probe whether deduplication is active on a host. This is no longer possible by writing a unique page and a pair of identical pages to the memory of their VM and comparing the time it takes to write to these. However, an attacker can still see a difference in write times between a page that has been in memory for a long time and a page that has just been written to memory (and thus has not been scanned twice yet). Furthermore, an attacker can determine the speed of deduplication, i. e. how long it takes for a newly written page to be marked copy-on-write after having been scanned twice. This information is of little value to an attacker, though: There is no difference in write times between unique pages and pages of which an identical copy exists elsewhere on the host.

5. Evaluation

To ensure that FakeDD is useful, it is not only important that it can prevent the side-channel attack and that it is still able to deduplicate pages to save memory, but also that any impact it may have on performance is acceptable. To evaluate whether this is the case, Section 5.1 will first analyse the effectiveness of FakeDD in preventing side-channel attacks based on write time differences caused by KSM. Section 5.2 then evaluates the effect of FakeDD on memory savings. The impact on application performance is evaluated in Section 5.3. Finally, Section 5.4 investigates the resource usage of the deduplication mechanism itself.

All evaluations were run on Linux kernel 4.10-rc6 to allow for a comparison with VUsion, which is based on this kernel version and could not trivially be ported to a newer kernel version. Ubuntu 16.04 was used to minimise compatibility issues, as this distribution was originally running kernel 4.4 and was later updated to the 4.15 series, i. e. the required kernel version is within the range of versions used by this distribution release. All experiments were performed on kernels self-compiled based on the kernel sources obtained from the Linux kernel Git repository. The default compile options from Ubuntu kernel version 4.15.0-142 were used to create kernel image packages based on the source code of 4.10-rc6, patched with FakeDD or VUsion support, where applicable. Apart from applying the patches, no further changes were made to the kernel. Measurements using a newer 5.10 kernel on Debian 11 returned similar results with regard to the comparison between standard KSM and FakeDD. Therefore, results are only shown for kernel 4.10-rc6 for sake of brevity.

Except for changes noted in the description of the individual experiments, the default configuration of the deduplication mechanisms was used. Experiments were run on a computer equipped with an Intel Core i7-4790 quad-core CPU and 16 GiB of physical DDR3-1600 memory.

5.1. Effectiveness

This section aims at evaluating whether FakeDD is effective against memory deduplication side-channel attacks. This would be the case if an attacker could not observe any timing differences between writes to deduplicated and non-deduplicated pages.

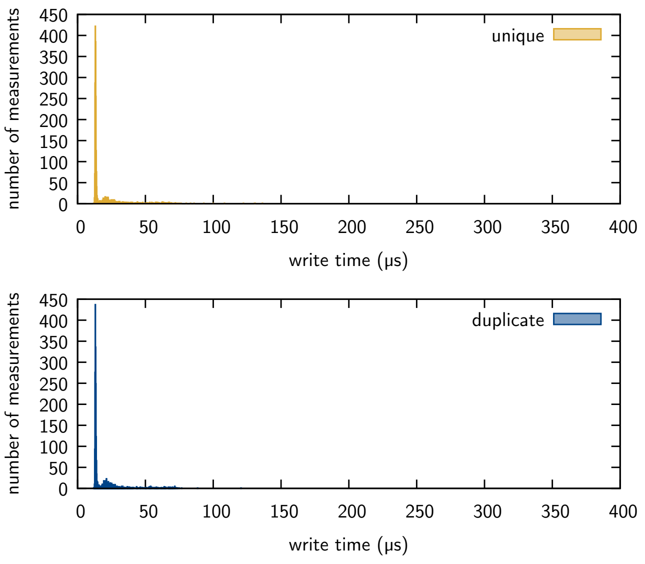
To evaluate whether this is the case, Linux was booted with the different kernels prepared for evaluation to compare their behaviour with regard to the write times to deduplicated and non-deduplicated pages. KSM was enabled and set to scan 500 pages every 10 milliseconds. Two VMs were started on the host to serve as a victim and an attacker VM. The attacker VM was allocated 1.5 GiB of memory and running Debian 11, while the victim VM was allocated 1 GiB of memory and running Debian 10. In the victim VM, a randomly generated set of 50 pages was loaded. In the attacker VM, the same set of pages was loaded in addition to another identically sized randomly generated set of pages. Using relatively large sets of 50 pages ensures that write times to deduplicated and non-deduplicated sets of pages should be clearly distinguishable in scenarios where a side-channel is present. After waiting for 120 seconds, the sets of pages in the attacker VM were overwritten, while measuring the time this took. The wait time was chosen so that KSM could scan the full memory of the VMs more than twice, i. e. it should always be able to identify potential duplicates before the pages are overwritten. For this purpose, a single write operation was performed per set of pages. No background activity was running on either the host or within the VMs apart from the basic operating system setup to ensure that any noise in the measurements caused by background activity is kept to a minimum. This process was performed 2 500 times each for sets of pages matching between the VMs and sets of pages unique to the attacker VM.

Figure 3 shows histograms of the observed write times to the sets of 50 deduplicated and non-deduplicated pages within the attacker VM for the different kernels. Figure 3a shows the histograms for a standard kernel with deactivated KSM. As expected due to deduplication being deactivated, it can be seen that the timing distributions are very similar to each other, with the distributions peaking around 16 ms. This indicates that an attacker would be unable to tell whether pages are also present in another VM.

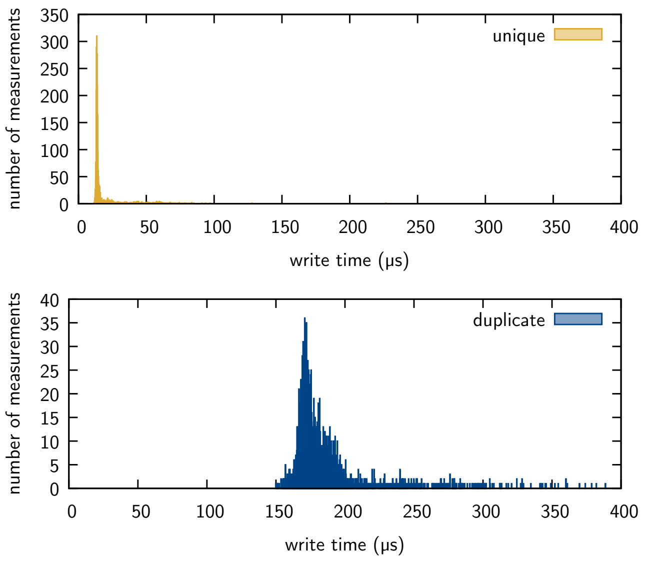
Figure 3b shows the histograms for a standard kernel with activated KSM. Here, the distribution for the write times to sets of pages that were loaded in both VMs is quite different to that for sets of pages unique to the attacker VM. While the distribution for unique pages is almost identical to the experiment without deduplication, write operations tend to take longer for pages present in both VMs, with the distribution peaking at about 170 µs. While both distributions contain outliers for which write operations took longer than is typical, there is no overlap between the distributions. This indicates that an attacker would be able to tell whether pages are also present in another VM.

The histograms for FakeDD are shown in Figure 3c. On the one hand, it can be seen that the distributions are now once more very similar to each other, indicating that an attacker would not be able to tell whether a page is also present in another VM despite deduplication taking place. On the other hand, the write operations take longer than for the standard kernel without KSM: Write times to unique pages are now as long as write times to deduplicated pages for the standard kernel with KSM. Both distributions are almost identical to that for duplicate pages using standard KSM, having their peak at about 170 µs. Therefore, while it cannot hide that a deduplication mechanism is active due to its longer write times, FakeDD is effective against timing side-channel attacks aimed at determining whether a page with specific contents is present on a host.

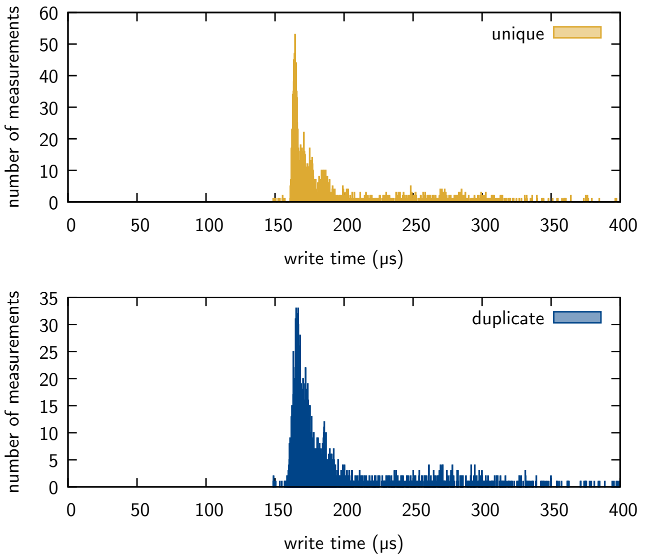
Finally, the histograms for the VUsion kernel are shown in Figure 3d. Again, both distributions are almost identical to each other, indicating that an attacker would be unable to tell the difference between the two types of pages, i. e. that VUsion is equally effective at preventing an attacker from finding out whether an identical copy of a page is present in another VM. However, the distributions are different from that for duplicate pages using standard KSM and those for FakeDD. The peak is higher at around 200 µs, indicating that the handling of a copy-on-write page fault takes longer. Furthermore, the distributions not only contain outliers with



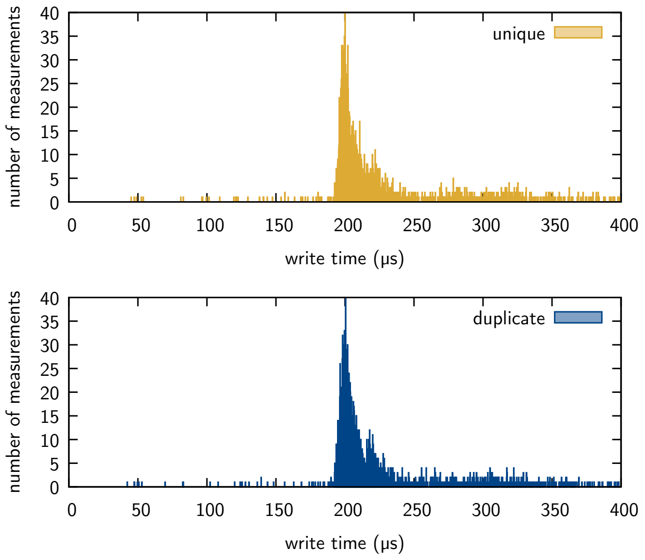
(a) Standard kernel, no deduplication



(b) Standard KSM



(c) FakeDD



(d) VUsion

Figure 3: Histograms of the write time to sets of 50 duplicate or unique pages for the different deduplication mechanisms

longer write times, which are present in all distributions, but also ones with write times significantly shorter than the peak. These can likely be attributed to the working set estimation mechanism preventing some pages from being marked copy-on-access.

5.2. Deduplication performance

A change in the deduplication mechanism may change its ability to identify and deduplicate identical memory pages. On the one hand, this may have an impact on the total amount of memory that can be saved, e. g. due to pages becoming ineligible for deduplication. On the other hand, the time it takes to realise the potential savings after identical pages have been placed in memory could change, e. g. due to the deduplication mechanism taking longer to find duplicate pages.

To evaluate what impact FakeDD has on the deduplication performance, the memory savings in a test environment were monitored over time for both the stock and patched KSM implementations. To this end, four VMs were set up. Two VMs were running Debian 11. One VM was running Debian 12, i. e. software that was relatively similar, but newer. The last VM was running FreeBSD 14.0, i. e. a completely different operating system. Each VM was allocated two CPU cores and 2 GiB of memory. For all VMs, a default installation of the respective operating system was performed. The software used in the VMs ensures that there was a large number of pages eligible for deduplication, of which some were actually duplicates, while others were unique to a VM. After booting and logging into each VM, a snapshot was created, so that the identical system state could be restored for experiments with the different deduplication types.

To launch an experiment, the snapshots were restored, with the VMs initially remaining suspended. KSM was then enabled and set to scan 100 pages per 20 milliseconds, i. e. the default configuration of KSM was used. At the same time, the VMs were unsuspended. From this point onwards, the following KSM statistics, which are accessible through the path `/sys/kernel/mm/ksm/`, were monitored every 0.25 seconds for a period of 30 minutes:

- the number of physical pages used by KSM, i. e. the number of copy-on-write pages (`pages_shared`) and
- the number of extra virtual pages referring to these physical pages (`pages_sharing`), which allows determining how much memory was saved and
- the number of physical copy-on-write (standard KSM, FakeDD) or copy-on-access (VUsion) pages that have only one virtual page referring to them (`pages_fakededup`).

The number of physical copy-on-write/-access pages with one reference is not normally reported by the kernel. Therefore, the kernels used in this experiments are specially patched to monitor and report this value. However, as this introduces additional instructions into the KSM implementation, these kernels are *only* used in the experiments for this section, while the kernels used in all other experiments do not include the additional statistics code.

Figure 4 shows the development of `pages_shared` for standard KSM, FakeDD and VUsion over time. The results indicate that FakeDD and VUsion both add significantly more pages to the stable tree. In the case of FakeDD, these pages are additional copy-on-write pages with only one virtual page pointing to them, due to FakeDD fake-deduplicating unique pages. This is similar for VUsion, but pages are copy-on-access. The graph for VUsion lags significantly behind that for FakeDD, though, which is likely due to VUsion considering fewer pages as candidates for addition to the stable tree owing to its working set estimation. VUsion only catched up towards the end of the experiment. It is likely that the number of `pages_shared` would remain lower in an environment with more VM activity, though: VUsion removes such pages on merely being read, as copy-on-access is used instead of copy-on-write both for deduplicated pages and unique pages that have been made copy-on-access. The results are in line with expectations, as the countermeasures also perform copy-on-write (or even copy-on-access) for pages which are unique within the host's memory.

Note that while the increase in the number of `pages_shared` for FakeDD and VUsion indicates that

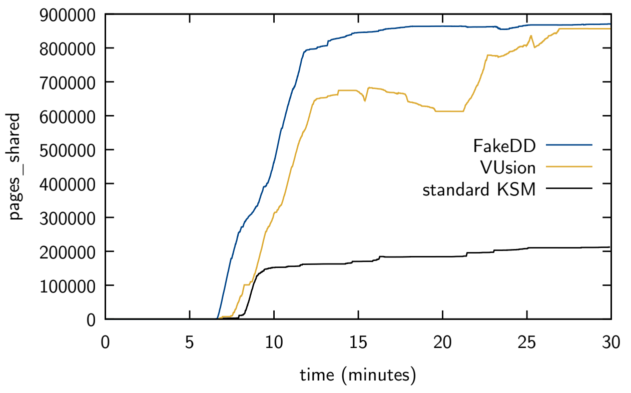


Figure 4: `pages_shared` over time

the stable tree is larger, these pages no longer have to be managed on the unstable tree. With standard KSM, the number of pages on the stable tree and the unstable tree before it is cleared at the end of a memory pass would be similar to the number of pages observed on the stable tree for FakeDD in the experiment. This is due to both implementations considering the same pages for deduplication. Only VUsion manages to actually reduce the number of pages that have to be managed in trees, but this comes at the cost of considering fewer pages for deduplication.

Figure 5 shows how much memory standard KSM, FakeDD and VUsion saved by deduplicating pages over time. The results indicate that FakeDD is almost equally effective at achieving memory savings as standard KSM. The amount of memory saved is almost identical throughout the experiment. However, the results indicate that FakeDD realises the memory savings minimally later than standard KSM. VUsion, on the other hand, achieves significantly lower memory savings. Not only do its memory savings lag behind in time, they also never reach the level of standard KSM and FakeDD. This indicates that VUsion's working set estimation as well as breaking deduplication even for pages being read comes at the cost of reducing memory savings. After 20 minutes, VUsion saved 932.47 MiB less than the standard KSM impementation (-30.99 %). At the end of the experiment, the lost memory savings of VUsion reduced to 371.43 MiB (-11.87 %).

Figure 6 shows the number of copy-on-write (standard KSM, FakeDD) or copy-on-access (VUsion) pages on the stable tree with only one virtual page attached. The results indicate that FakeDD creates many copy-on-write pages with just one reference at the same time as the rise in `pages_shared` and memory savings. This is in line with expectations, as unique pages would be scanned and become eligible at a similar time as the duplicate pages. Also as expected, the number of copy-on-write pages with only one reference about matches the difference in `pages_shared` between standard KSM and FakeDD.

Results are similar for copy-on-access pages when running VUsion. It also creates a large number of such pages with only one virtual page attached. As with

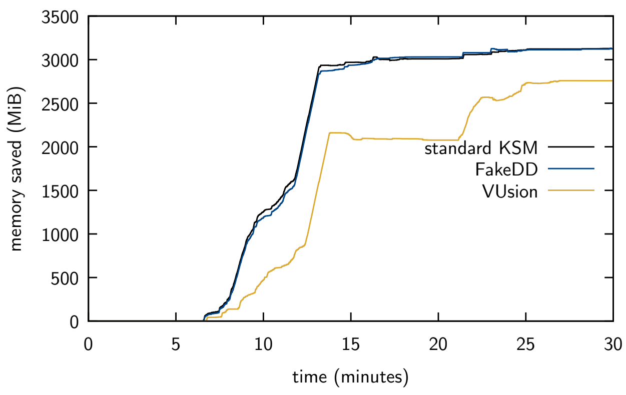


Figure 5: Memory saved through deduplication over time

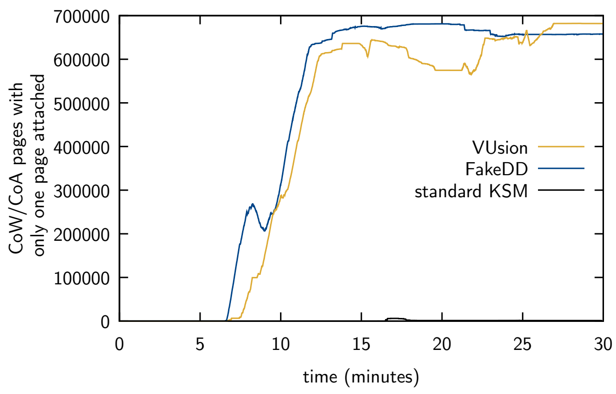


Figure 6: Number of copy-on-write pages on the stable tree with only one virtual page attached

`pages_shared`, the graph lags behind FakeDD slightly in time. The number of such pages is slightly lower for VUsion than for FakeDD throughout most of the experiment, but – unlike for memory savings – slightly exceeds it towards the end of the experiment.

Note that the number of copy-on-write pages with only one reference is larger than zero even for standard KSM. This is because while pages are only marked as copy-on-write when there is more than one identical virtual page, the copy-on-write state is not removed from a physical page even if there is only one reference left to it. Such situations can arise when all but one copy of a page have been modified after originally having been deduplicated.

5.3. Application performance

While it is important that the countermeasure protects against side-channel attacks and deduplication still works as intended, it is also important that is does not cause a large overhead compared to standard KSM. One aspect of this is that changing the deduplication mechanism may also have an impact on the performance of applications executed within VMs, which this section aims to evaluate. For FakeDD, some write operations will be slower than on a system using the stock KSM implementation. This is because pages that are unique and stable are marked copy-on-write by the deduplication mechanism. It thus takes longer to write to these pages. Furthermore, the copying operations use memory bandwidth that will not be available for other operations.

For the experiments, four VMs were used. Two VMs were allocated two CPU cores and 1 GiB of memory each and were running Debian 10. In addition to the base system, these VMs both kept an identical set of 100 files with a size of 1 to 100 pages in memory. This means that the memory of these VMs contained many identical pages that could be deduplicated. A third VM was allocated two CPU cores and 4 GiB of memory and was also running Debian 10. This VM kept a separate set of five 512 MiB files in memory. Thus, the memory of this VM contained a large number of unique pages that could not be deduplicated. Finally, a fourth VM running Debian 11 was allocated eight CPU cores and 4 GiB of memory. This VM was used to perform the application performance benchmarks. Assigning it a number of cores identical to the number of virtual cores on the host ensures that the benchmarks could make use of the full CPU resources of the system. Therefore, any potential CPU overhead of the changed deduplication mechanism would also impact performance within the VM and could not be hidden away on another core.

For each experiment, the three background VMs were first started and the data files loaded into memory. Then, the benchmark VM was started. Before starting the benchmarks, KSM was activated if the experiment required it. It was set to scan 500 pages every 10 milliseconds, i. e. a relatively aggressive configuration was chosen that should have a higher performance impact than the default configuration. The KSM statistics values indicating the number of physical pages used by deduplication (`pages_shared`) and the number of extra virtual pages referring to them (`pages_sharing`) were monitored. Once these values stabilised, the benchmarks were launched on the benchmark VM. For this, Phoronix Test Suite (PTS) 10.8.4 was used to run a range of individual benchmarks covering different workloads. PTS automatically adjusts the number of runs of each benchmark to achieve a low standard deviation [32]. The StandardDeviationThreshold parameter was set to 2.5. To increase the reliability of the results and avoid PTS cutting down on the number of executions of long-running benchmarks, the `strict-run` command was used to run the benchmarks.

Table 1 shows the results of the individual benchmarks, comparing the performance of the different deduplication set-ups tested. The results indicate that the impact of the different deduplication mechanisms depends strongly on the workload.

For some workloads, performance is almost unaffected by activating deduplication with or without one of the countermeasures. One example for this is the ‘syscall basic’ test of perf-bench, where activating standard KSM reduced performance by 0.97 %. FakeDD and VUsion achieved slightly better results than standard KSM, but no significant differ-

Table 1: Results of Phoronix kernel benchmarks compared between a host without KSM as well as with standard KSM, FakeDD and VUsion. Performance differences that are not statistically significant are marked with [].

Test (Unit, △/▽ better?)	No KSM	Std. KSM	± no KSM	FakeDD	± std. KSM	VUsion	± FakeDD
7-Zip, compression (MIPS, △)	29899	26369	-11.81%	27689	+5.01%	27015	-2.43%
7-Zip, decompression (MIPS, △)	22263	21636	-2.82%	22299	+3.06%	22159	-0.63%
Apache, 1000 concurrent requests (requests/s, △)	29170.63	24829.55	-14.88%	25724.87	+3.61%	25383.8	-1.33%
Dbench, 12 clients (MB/s, △)	33.0597	33.3719	[+0.94%]	33.5226	[+0.45%]	33.6481	[+0.37%]
LAME MP3 Encoding (s, ▽)	9.797	9.831	+0.35%	9.835	+0.04%	9.863	+0.28%
memcached, set-to-get ratio: 1:1 (operations/s, △)	814728.37	786836.49	-3.42%	690308.26	-12.27%	703472.92	+1.91%
OpenSSL, RSA4096 (sign/s, △)	655.1	630	-3.83%	653	+3.65%	671.2	+2.79%
OpenSSL, RSA4096 (verify/s, △)	42064.1	41252	-1.93%	43238.6	+4.82%	43870.8	+1.46%
perf-bench, epoll wait (ops/sec, △)	132073	127383	-3.55%	129692	+1.81%	129628	-0.05%
perf-bench, futex hash (ops/sec, △)	1500254	1475094	-1.68%	1497932	+1.55%	1483198	-0.98%
perf-bench, futex lock-pi (ops/sec, △)	1711	1674	-2.16%	1666	-0.48%	1666	[±0%]
perf-bench, memcpy 1MB (GB/sec, △)	21.419358	21.658203	[+1.12%]	21.754798	+0.45%	21.443243	-1.43%
perf-bench, memset 1MB (GB/sec, △)	34.885568	34.634202	-0.72%	34.844534	+0.61%	34.953145	+0.31%
perf-bench, sched pipe (ops/sec, △)	188780	194652	+3.11%	188388	-3.22%	184173	-2.24%
perf-bench, syscall basic (ops/sec, △)	3023633	2994345	-0.97%	2999231	+0.16%	3002090	[+0.1%]
pmbench, 8 threads, read+write (us (latency), ▽)	0.0846	0.0971	+14.78%	0.095	-2.16%	0.0981	+3.26%
pmbench, 8 threads, read (us (latency), ▽)	0.0475	0.0469	[-1.26%]	0.0464	-1.07%	0.0527	+13.58%
pmbench, 8 threads, write (us (latency), ▽)	0.0455	0.0549	+20.66%	0.0516	-6.01%	0.0561	+8.72%
pgbench, scaling 1, 100 clients, read (tps, △)	99258	84827	-14.54%	82687	-2.52%	83610	[+1.12%]
pgbench, scaling 1, 100 clients, read (ms (latency), ▽)	1.008	1.179	+16.96%	1.21	+2.63%	1.196	[-1.16%]
SQLite, 8 threads (s, ▽)	3002.437	3404.865	+13.4%	2998.585	-11.93%	3269.939	+9.05%
x264, Bosphorus 1080p (fps, △)	38.29	36.39	-4.96%	37.67	+3.52%	36.73	-2.5%

ence could be found between the two countermeasures.

Other workloads see a larger hit in performance. Two examples of this are the tests of pmbench involving write operations, where activating standard KSM causes a performance degradation of 14.78 % and 20.66 %, respectively. For these, FakeDD performs slightly better than standard KSM, while VUsion performs worse. Somewhat similarly, Apache sees a 14.88 % degradation in performance with standard KSM, while VUsion performs slightly better and FakeDD again slightly better than VUsion. The results indicate that the largest part of the performance overhead for these benchmarks is caused by deduplication being activated at all, while the countermeasures have a smaller impact on performance.

For memcached, the situation is somewhat different. Here, the results indicate that standard KSM causes a relatively small performance overhead of 3.42 %. The two countermeasures, on the other hand, incur an additional performance overhead of more than 10 %.

For some workloads, PTS reports a positive performance impact of the countermeasures. For example, 7-Zip compression sees a reduction in performance by 11.81 % when standard KSM is activated. FakeDD, however, achieved a 5.01 % higher performance than standard KSM. VUsion was 2.43 % slower than FakeDD, but still faster than standard KSM.

The results indicate that VUsion causes a larger performance overhead than FakeDD in most scenarios. In some cases, this was quite drastic, e. g. for the read test of pmbench, which was 13.58 % slower than with FakeDD. In other cases, however, it offers a performance benefit over FakeDD, e. g. for the RSA4096 tests of the OpenSSL benchmark. A possible explanation for this is that VUsion breaks deduplication once pages are accessed. If a workload often reads pages that VUsion marks as copy-on-access, this could be expected to negatively affect performance compared to standard KSM and FakeDD, which would not copy a page on a read operation. On the other hand, this may lead to pages that are frequently accessed by the benchmark simply remaining unaffected by copy-on-access with VUsion, while standard KSM and FakeDD may mark these pages as copy-on-write due to them only being read, but not written to.

Table 2 shows the average number of page faults per second for each individual Phoronix benchmark compared between the different deduplication mechanisms. To obtain these numbers, each benchmark was run for a period of 30 minutes using the `stress-run` command. Simultaneously, the page fault rate was measured using the sar command on the host OS. Otherwise, the experiment setup was identical to the previously described application performance benchmarks.

Any deduplication mechanism creates potential for additional page faults: For standard KSM, this occurs only for pages which were deduplicated due to multiple copies existing on the host. For FakeDD, page faults can additionally occur for unique pages marked copy-on-write. VUsion not only allows for page faults on write operations on both deduplicated and unique pages, but also on mere read operations. However, this is countered by its working set estimation enforcing stricter inactivity requirements to be fulfilled before a page is considered for deduplication or being marked copy-on-access. Note that any potential performance impact of the additional page faults will also be reflected in the performance measurements presented in Table 1.

As expected, the results indicate that activating deduplication increases the page fault rate for all benchmarks. Activating standard KSM increases it by at 62.15 % (perf-bench, memset) to 1 339.92 % (Apache). For 6 of 19 benchmarks, FakeDD reduces the page fault rate compared to standard KSM. For the other benchmarks, it further increases the page fault rate, although in most cases, the increase is relatively small in comparison to that caused by activating standard KSM. Compared to FakeDD, VUsion reduces the

Table 2: Average number of page faults per second observed while running Phoronix benchmarks for 30 minutes each.

Test	No KSM	Std. KSM	± no KSM	FakeDD	± std. KSM	VUsion	± FakeDD
7-Zip, compression+decompression	73.66	258.92	+251.51%	495.72	+91.46%	296.28	-40.23%
Apache, 1000 concurrent requests	49.32	710.17	+1339.92%	584.05	-17.76%	303.68	-48.00%
Dbench, 12 clients	98.57	1336.13	+1255.51%	1202.44	-10.01%	225.93	-81.21%
LAME MP3 Encoding	111.91	327.46	+192.61%	313.91	-4.14%	204.30	-34.92%
memcached, set-to-get ratio: 1:1	45.61	215.14	+371.69%	289.12	+34.39%	756.67	+161.71%
OpenSSL, RSA4096	72.78	176.57	+142.61%	195.54	+10.74%	132.49	-32.24%
perf-bench, epoll wait	95.46	194.18	+103.42%	283.61	+45.06%	165.65	-41.59%
perf-bench, futex hash	98.10	186.31	+89.92%	212.28	+13.94%	146.55	-30.96%
perf-bench, futex lock-pi	105.45	234.30	+122.19%	253.35	+8.13%	2495.89	+885.15%
perf-bench, memcpy 1MB	178.01	291.56	+63.79%	263.79	-9.52%	206.95	-21.55%
perf-bench, memset 1MB	192.85	312.70	+62.15%	308.08	-1.48%	242.18	-21.39%
perf-bench, sched pipe	84.94	211.40	+148.88%	220.13	+4.13%	161.70	-26.54%
perf-bench, syscall basic	105.25	219.93	+108.96%	276.31	+25.64%	169.51	-38.65%
pmbench, 8 threads, read+write	67.84	202.91	+199.10%	326.28	+60.80%	195.71	-40.02%
pmbench, 8 threads, read	82.40	164.92	+100.15%	216.19	+31.09%	130.86	-39.47%
pmbench, 8 threads, write	67.18	215.59	+220.91%	245.83	+14.03%	191.19	-22.23%
pgbench, scaling 1, 100 clients, read	48.18	486.13	+908.99%	973.59	+100.27%	527.75	-45.79%
SQLite, 8 threads	177.21	350.21	+97.62%	336.49	-3.92%	265.88	-20.98%
x264, Bosphorus 1080p	59.71	785.64	+1215.76%	997.29	+26.94%	832.49	-16.52%

page fault rate in 17 of 19 benchmarks. In some cases, its page fault rate is even lower than that of standard KSM, likely due to its working set estimation (which also reduces memory savings, though). In two benchmarks, however, it leads to a sizeable further increase in page fault rate. None of the benchmarks using deduplication exhibited a lower page fault rate than the corresponding benchmark without deduplication.

Overall, the results indicate that there is a performance penalty for enabling deduplication in most tests. For most of these, the penalty is relatively small and enabling deduplication only leads to a slow-down in the single-digit percentage range. However, the performance impact is larger for some tests, e. g. the pmbench tests involving write operations. Where there is a performance penalty, most of it is already incurred by enabling the standard KSM implementation. The countermeasures incur a higher performance overhead for many tests. However, it is typically smaller than that incurred by activating KSM in the first place. In some cases, a countermeasure can even offer better performance than standard KSM. All in all, where the performance impact of standard KSM is deemed acceptable, the slightly higher performance overhead caused by one of the secure modified implementation may also be acceptable in exchange for the added security.

5.4. Resource usage of deduplication mechanism

In addition to influencing the performance of applications, e. g. through additional pages faults, a content-based deduplication also causes a more directly observable overhead. It needs to regularly scan the memory of a computer for duplicate pages. Due to the required comparisons of page contents, this consumes CPU resources. In Linux, the memory scanning including the page comparison is performed by the `ksmd` process. The resource consumption is thus allocated to this process and can be measured directly. This section will therefore look into what impact FakeDD has on the CPU time consumed by the `ksmd` process.

An identical VM set-up as for the application performance experiments (cf. Section 5.3) was used, i. e. two VMs with 1 GiB of RAM were holding an identical set of 100 files with a size between 1 and 100 pages. A third VM was holding a set of five files with a size of 512 MiB each. The last VM was executing the Phoronix Test Suite 10.8.4 also used for the evaluation of application performance to ensure that memory pages are constantly being modified, so that they must be scanned by KSM. The same range of diverse benchmarks was used to ensure a variation of workloads.

As for the page fault rate experiment, the host system was freshly booted with the appropriate kernel for each implementation (standard KSM, FakeDD, VUsion). Snapshots of the VMs were restored in a paused state. KSM was then activated and set to scan 500 pages every 10 milliseconds, i. e. a relatively aggressive configuration was again used to more clearly emphasise any potential performance impact. After activating KSM, the VMs were unpaused. The benchmarks were then started: Each benchmark was run for 30 minutes using the `stress-run` command. Three minutes after a benchmark had ended, the next one was started, i. e. the total run time was 624 minutes (19 benchmarks * 30 minutes + 18 intervals * 3 minutes). Benchmarks were run in the same order as shown in Table 2. From the time of starting the first benchmark, the CPU time consumed by the `ksmd` process was polled and stored every 60 seconds until all benchmarks had been executed.

Figure 7 shows the CPU time consumed by the `ksmd` process over time for standard KSM, FakeDD and VUsion. Note that this does *not* represent the full overhead of KSM, but only that incurred by the actual scanning of memory pages and related management activities, i. e. the overhead in application performance, as analysed in the previous section, is separate from this.

For all three KSM implementations, the CPU time consumed rises almost linearly over time, with only little variation in the rate of CPU consumption between workloads. The standard KSM implementation consumes 6 487 CPU seconds over the course of the experiment. Interestingly, both implementations aiming at countering side-channel

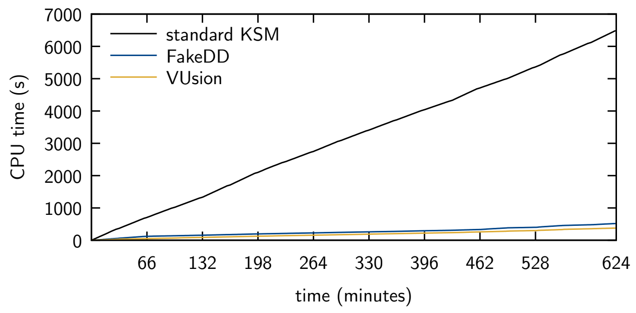


Figure 7: CPU time consumed over time by the `ksmd` process of the modified implementation compared to the stock and VUsion implementations

attacks *reduce* the CPU consumption. FakeDD consumes 518 CPU seconds throughout the experiment, representing a 92 % reduction. The VUsion implementation needs even fewer CPU resources and consumes just 377 CPU seconds throughout the experiment, representing a 94.2 % reduction compared to standard KSM.

The results indicate that both countermeasures *decrease* the CPU consumption of the KSM scanner. At first, this may seem counter-intuitive. One might expect CPU consumption to increase, as significantly more pages are converted to copy-on-write KSM pages and inserted into the stable tree. However, this effect could be countered by the FakeDD scanner actually skipping pages that have been changed since the last scan in the first pass instead of immediately comparing them to pages on the stable tree. Furthermore, FakeDD does not constantly re-scan unchanged pages during each scan cycle. The results indicate an even stronger effect for VUsion with its working set estimation algorithm, which tries to skip even pages that have merely been accessed in any form.

6. Conclusion

This paper presents FakeDD, which uses fake deduplication as a countermeasure against timing side-channel attacks based on the write time differences caused by content-based memory deduplication. It marks pages as copy-on-write even if they are unique on the host and no second identical page is present.

The results indicate that FakeDD eliminates timing differences between overwriting unique and duplicate pages. Therefore, it prevents an attacker from determining whether another copy of a page exists on the host. Furthermore, the results indicate that FakeDD's effectiveness in saving memory is almost identical to that of standard KSM and superior to VUsion. Finally, the results indicate a decrease in the CPU time consumption of the scanner process, while there is some overhead in terms of application performance. While FakeDD increases this overhead for some usage scenarios, the increase is typically smaller than the overhead caused by deploying KSM in the first place. Therefore, where the performance overhead of standard KSM is deemed acceptable, the additional overhead may be an acceptable trade-off for the added security.

A limitation of FakeDD is that it only defends against side-channel attacks based on write operations. Rowhammer-based attacks, which rely on read operations, are therefore not covered, unlike in VUsion. Furthermore, FakeDD, unlike VUsion, does not move fake-deduplicated pages to a new location. Therefore, page colouring and page sharing attacks, as described by Oliverio et al. [2], which rely on cache-based side-channel attacks, and, in the latter case, on some level of control over the victim, are not covered by its attacker model.

Due to its higher memory savings potential and lower overhead compared to VUsion, FakeDD complements the state of the art – which of the two countermeasures is more suitable for a system will depend on the attacker model. If additional protection against only some of the attacks additionally covered by VUsion is desired, additional points on the trade-off could be approached, e. g. by adding a mechanism to FakeDD that moves a page on marking it copy-on-write, thereby protecting against page colouring and page sharing attacks. Note, however, that while many attacks based on Rowhammer are facilitated by deduplication due to making it easier to access a physyical memory page without having any direct access to the application controlling the target virtual page, some forms of Rowhammer attacks (e. g. [16], [33], [34]) pose a danger even without memory deduplication being active. Therefore, this vulnerability should best be addressed separately. If systems were made unaffected by Rowhammer attacks, this would remove the need to consider such attacks in the context of memory deduplication on such systems.

Potential future work includes implementing fake deduplication for other content-based memory deduplication mechanisms. This would allow to evaluate its effectiveness and performance impact in other environments. Furthermore, even if the information exposed seems useless to an attacker at present, it might be worthwhile to consider whether countermeasures could be designed in a way to remove the residual side-channel described in Section 4.2, which could give away information about the configuration of the deduplication mechanism.

References

[1] 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, EUROSEC'11*, E. Kirda and S. Hand, Eds. ACM, 2011.

[2] M. Oliverio, K. Razavi, H. Bos, and C. Giuffrida, "Secure page fusion with VUsion," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 531–545.

[3] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using KSM," in *Proceedings of the Linux Symposium*, 2009, pp. 19–28.

[4] C. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation, OSDI*, D. E. Culler and P. Druschel, Eds. USENIX Association, 2002.

[5] Broadcom, “Additional Transparent Page Sharing management capabilities and new default settings,” Broadcom Knowledge Base Article 323624, 1 2021, (accessed 2024-05-28). [Online]. Available: https://knowledge.broadcom.com/external/article?legacyId=2097593

[6] Xen Project, “Support statement for this release,” 2023, (accessed 2024-05-28). [Online]. Available: https://xenbits.xen.org/docs/4.18-testing/SUPPORT.html

[7] ——, “[xen.git] / xen / arch / x86 / mm / mem_sharing.c,” 2011, (accessed 2024-05-28). [Online]. Available: https://xenbits.xen.org/gitweb/?p=xen.git;a=blob;f=xen/arch/x86/mm/mem_sharing.c;h=da28266ef0769570c598c097e0d917af0b634820;hb=HEAD

[8] K. Fraser, “Reads from read only parent disk images are intercepted, and are used to detect potentially sharable memory pages,” Git commit in the Xen project repository, 12 2009, (accessed 2024-05-28). [Online]. Available: https://xenbits.xenproject.org/gitweb/?p=xen.git;a=commit;h=4515e6d82b23366933055b0ad60d8ed89f5a2b92

[9] T. K. Lengyel, “Remove undocumented and unmaintained tools/memshr library,” Git commit 8d1d28bfcfd04d15c07c2f5c63aed3c7d220b024 in the Xen project repository, 1 2020, accessed 2024-05-28. [Online]. Available: https://xenbits.xenproject.org/gitweb/?p=xen.git;a=commit;h=8d1d28bfcfd04d15c07c2f5c63aed3c7d220b024

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

[11] R. Owens and W. Wang, “Non-interactive OS fingerprinting through memory de-duplication technique in virtual machines,” in *30th IEEE International Performance Computing and Communications Conference, IPCCC 2011*, S. Zhong, D. Dou, and Y. Wang, Eds. IEEE Computer Society, 2011.

[12] J. Lindemann and M. Fischer, “A memory deduplication side-channel attack to detect applications in co-resident virtual machines,” in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018*, H. M. Haddad, R. L. Wainwright, and R. Chbeir, Eds. ACM, 2018, pp. 183–192.

[13] ——, “Efficient identification of applications in co-resident vms via a memory side-channel,” in *ICT Systems Security and Privacy Protection - 33rd IFIP TC 11 International Conference, SEC 2018, held at the 24th IFIP World Computer Congress, WCC 2018*, ser. IFIP Advances in Information and Communication Technology, L. J. Janczewski and M. Kutylowski, Eds., vol. 529. Springer, 2018, pp. 245–259.

[14] A. Barresi, K. Razavi, M. Payer, and T. R. Gross, “CAIN: silently breaking ASLR in the cloud,” in *9th USENIX Workshop on Offensive Technologies, WOOT ’15*, A. Francillon and T. Ptacek, Eds. USENIX Association, 2015.

[15] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, “Dedup est machina: Memory deduplication as an advanced exploitation vector,” in *IEEE Symposium on Security and Privacy 2016*. IEEE Computer Society, 2016, pp. 987–1004.

[16] Y. Kim, R. Daly, J. S. Kim, C. Fallin, J. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors,” in *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA*. IEEE Computer Society, 2014, pp. 361–372.

[17] J. Xiao, Z. Xu, H. Huang, and H. Wang, “Security implications of memory deduplication in a virtualized environment,” in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE Computer Society, 2013.

[18] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, “Know thy neighbor: Crypto library detection in cloud,” *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 1, pp. 25–40, 2015.

[19] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. M. Swift, “A placement vulnerability study in multi-tenant public clouds,” in *USENIX Security Symposium*. USENIX Association, 2015, pp. 913–928.

[20] Z. Xu, H. Wang, and Z. Wu, “A measurement study on co-residence threat inside the cloud,” in *USENIX Security Symposium*. USENIX Association, 2015, pp. 929–944.

[21] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my cloud : Exploring information leakage inthird-party compute clouds,” in *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS*, E. Al-Shaer, S. Jha, and A. D. Keromytis, Eds. ACM, 2009, pp. 199–212.

[22] A. Honig and N. Porter, “Google Cloud Platform: 7 ways we harden our KVM hypervisor at Google Cloud: security in plaintext,” January 2017, (accessed 2024-05-27). [Online]. Available: https://cloud.google.com/blog/products/gcp/7-ways-we-harden-our-kvm-hypervisor-at-google-cloud-security-in-plaintext?m=1

[23] G. Irazoqui, T. Eisenbarth, and B. Sunar, “Jackpot stealing information from large caches via huge pages,” *IACR Cryptology ePrint Archive*, 2014. [Online]. Available: http://eprint.iacr.org/2014/970

[24] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, “Flip feng shui: Hammering a needle in the software stack,” in *25th USENIX Security Symposium*, T. Holz and S. Savage, Eds. USENIX Association, 2016.

[25] A. Agarwal and T. N. B. Duong, “Secure virtual machine placement in cloud data centers,” *Future Gener. Comput. Syst.*, vol. 100, pp. 210–222, 2019.

[26] J. Lindemann, “How to hide your VM from the big bad wolf? co-location resistance vs. resource utilisation in VM placement strategies,” in *Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES*. ACM, 2023, paper 41.

[27] Amazon AWS, “Amazon EC2 Dedicated Instances,” 2024, (accessed 2024-05-28). [Online]. Available: https://aws.amazon.com/ec2/pricing/dedicated-instances/

[28] M. Payer, “Hexpads: A platform to detect ‘stealth’ attacks,” in *Engineering Secure Software and Systems - 8th International Symposium, ESSoS*, ser. Lecture Notes in Computer Science, J. Caballero, E. Bodden, and E. Athanasopoulos, Eds., vol. 9639. Springer, 2016, pp. 138–154.

[29] A. W. Paundu, D. Fall, D. Miyamoto, and Y. Kadobayashi, “Leveraging kvm events to detect cache-based side channel attacks in a virtualization environment,” *Security and Communication Networks*, vol. 2018, no. 1, p. 4216240, 2018.

[30] I. Eidus and H. Dickins, “Kernel samepage merging feature,” 11 2009, note: The file has been updated (and split from another file) after 2009, but the original date remains therein. (accessed 2024-05-28). [Online]. Available: https://elixir.bootlin.com/linux/v6.9.2/source/Documentation/admin-guide/mm/ksm.rst

[31] Debian Code Search, “MADV_MERGEABLE,” 5 2024. [Online]. Available: https://codesearch.debian.net/search?q=MADV_MERGEABLE

[32] M. Larabel, “Phoronix Test Suite documentation (documentation/phoronix-test-suite.md),” 5 2019, (accessed 2024-05-28). [Online]. Available: https://github.com/phoronix-test-suite/phoronix-test-suite/blob/7a93cf85f4b6f5a0049b25a02f4f483c74e4154c/documentation/phoronix-test-suite.md

[33] D. Gruss, C. Maurice, and S. Mangard, “Rowhammer.js: A remote software-induced fault attack in javascript,” in *13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA*, ser. Lecture Notes in Computer Science, J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds., vol. 9721. Springer, 2016, pp. 300–321.

[34] M. Seaborn and T. Dullien, “Exploiting the dram rowhammer bug to gain kernel privileges,” *Black Hat*, vol. 15, 2015.