Slicedup

Vañó-García, Fernando; Marco-Gisbert, Hector

Published in:
UBICOMM 2018

Published: 18/11/2018

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Vañó-García, F., & Marco-Gisbert, H. (2018). Slicedup: a tenant-aware memory deduplication for cloud computing. In C. D. C. Monteiro, K. Chatzikokolakis, & C. H. C. Tolentino (Eds.), *UBICOMM 2018: The Twelfth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies* (pp. 15-20). (UBICOMM International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies; Vol. 2018). International Academy, Research, and Industry Association.

Slicedup: A Tenant-Aware Memory Deduplication for Cloud Computing

Fernando Vañó-García, Hector Marco-Gisbert

School of Computing, Engineering and Physical Sciences
University of the West of Scotland
High St, Paisley PA1 2BE, UK
Email: {Fernando.Vano-Garcia,Hector.Marco}@uws.ac.uk

Abstract—**Memory deduplication allows cloud infrastructure providers to increase the profit of memory resources by taking advantage of the redundant nature of virtual machines footprint. Although it is an important feature to manage the memory resources of a cloud system efficiently, unfortunately, it enables different types of side-channel attacks which, in practice, means disabling memory deduplication. In this paper, we present Slicedup, a tenant-aware memory deduplication mechanism that prevents side-channel attacks. Our proposal enables cloud providers to get the deduplication saving benefits while preventing side-channel attacks among tenants. Since Slicedup is a design-solution, it can be implemented in any operating system, regardless of its version, architecture or any other system dependence. Finally, we show how Slicedup prevents side-channel attacks while providing similar memory savings when the number of tenants per physical host is low.**

Keywords–Cloud; Memory Deduplication; Information Security; Memory Management; Virtualisation.

I. INTRODUCTION

Cloud computing has extraordinary importance in modern society. It is a computing paradigm that allows users to access external resources in an "on-demand" basis, without having to bear the costs of infrastructure maintenance (physical servers, electricity bills, etc.). Physical resources are owned and maintained by a third party (cloud infrastructure provider), and users can obtain the desired computing resources securely and flexibly [1]. It is commonly believed that the concept of cloud computing first appeared in the 90's [2]. Since then, its popularity has grown broadly. Nowadays, cloud computing has been widely adopted in many sectors (e.g., telecommunications, content providers, etc.), mainly because it reduces the cost of performing tasks in a scalable and reliable way.

Despite the effort of many researchers to provide protection mechanisms [3] [4], attackers always end up finding new techniques to achieve their goals. Cloud infrastructure providers desire efficient resource management as well as to provide adequate levels of security for their customers. Unfortunately, this is not always possible, and performance mechanisms can compromise the system security. Memory deduplication is an instance of this problem. On the one hand, it offers the possibility of saving memory by eliminating duplicate contents. On the other hand, memory deduplication in a multi-tenant cloud environment can lead to serious security issues, which could allow a malicious attacker to abuse it and compromise other customer's highly sensitive information. For this reason, operating systems recently decided to deactivate memory deduplication by default [5] [6].

The main contributions of this paper are the following:

- We provide a state-of-the-art review of the known side-channel techniques related to memory deduplication in cloud systems.
- We propose and discuss a suitable solution to enable memory deduplication, avoiding side-channel attacks in multi-tenant cloud systems.

In the following sections, we present the challenges of memory deduplication concerning security in cloud systems. Section III surveys the state-of-the-art of known problems that memory deduplication introduces in virtualised systems. Then, Section IV provides our proposed solution to solve the memory deduplication side-channels in multi-tenant cloud systems. Finally, in Section V, we discuss the advantages and disadvantages of this approach in contrast with other countermeasures, and Section VI concludes.

II. BACKGROUND

In this section, the memory deduplication saving mechanism and cache memories are summarised.

A. Memory Deduplication

Memory deduplication is a memory saving mechanism that consists in detecting identical pages in memory and unify them into one single copy, liberating the space occupied by the redundant copies. This technique allows a cloud infrastructure provider to reduce the consumption of physical memory. Given the exceptional importance of efficient memory resources utilisation on behalf of cloud computing providers, deduplication is an important feature. It can reduce the memory footprint across virtual machines, increasing the profit of existing memory resources and decreasing the total cost of managing and ownership.

When two pages are compared, and both contain identical contents, they are mapped into a single physical page frame in memory using `Copy-On-Write` (COW) semantics. The COW mechanism allows a memory manager to share an object among processes belonging to different virtual address spaces. It is an optimisation heavily used by operating systems for copy operations, for example when a new process is created. A *COWed* object is write-protected, and when any of the processes try to modify their own instance, a new copy is generated in such a way as to ensure the integrity of the contents.

In a virtualised environment, deduplication is commonly applied to the entire memory region corresponding to the virtual machine (often called guest physical memory). Hence, all those pages belonging to that memory region are candidates for being shared.

B. Cache Memory

Cache memory is a small and high-speed Static Random Access Memory (SRAM) located in the CPU, which stores recently accessed data from the main memory. The purpose is to speed up the access to program instructions and data, exploiting the principle of locality [7]. As a consequence, the processor can obtain the information directly from the cache rather than from main memory, which has more access latency. Modern processors contain different levels of cache in their memory hierarchy. Typically they consist of three cache levels, where the first and second are private for each execution core, and the last level is shared by all the cores. The lower levels in the memory hierarchy contain small and fast memories, while the higher levels consist of big and slow memories.

There are different ways of organising a cache. The simpler approach is a direct-mapped cache, where an address is always associated with a single entry in the cache. This is cheap and works fast, but it introduces problems of address collisions (conflict miss). Another approach is a fully-associative cache, where any address can be stored in any entry of the cache. It is more expensive because a comparator is used to check the existing tags in the cache for every access, along with the need for an eviction policy (e.g., Least Recently Used). The n-way set associative caches are a combination of both approaches. The cache is divided into cache sets, each consisting of several cache lines (or ways). Sets are indexed with addresses to map specific memory locations (directly mapped), and the lines of each set are fully associative.

III. Threats of Memory Deduplication

Memory deduplication is a significant mechanism to save considerable amounts of memory in a cloud environment. Nevertheless, it introduces weaknesses that can be exploited by a malicious attacker and compromise the security of the system. It allows guests to communicate through a covert channel, which can be used by attackers to perform cross-VM access driven attacks and leak sensitive information. In this section, we present the state-of-the-art of side-channel attacks where the attacker shares memory with the victim in a virtualised environment.

A. Shared Memory Side-Channels

In a virtualised environment, memory deduplication is applied to pages in the physical host. This includes the sharing of pages belonging to different virtual machines that are located in the same host. A covert channel can be made because of the timing difference of the write operation to a page that is being shared by deduplication. This operation can be distinguished from a standard write to a page which is not being shared because a private copy must be done, and it takes more time. Therefore, an attacker can craft a page in their own address space, and then check if it is deduplicated by the host. In the affirmative case, at least another virtual machine holds a page with those contents.

The first study that exploits memory deduplication to perform a side-channel was carried out by Suzaki et al. [8]. They were able to check/detect the existence of specific applications in a different virtual machine located in the same physical host. Later, the same authors improved their work, discussing more approaches and countermeasures, being able to detect a specific virtual machine previously marked in a multi-tenant cloud environment [9]. Concurrently, Owens and Wang [10] were able to fingerprint guest operating systems using the same technique.

Xiao et al. [11] [12] studied the security implications of memory deduplication from the perspectives of both attackers and defenders. They presented a method to detect virtualisation and another to detect rootkits that modify kernel read-only data, both using memory deduplication covert channels. Suzaki et al. [13] continued their research on this topic, presenting more memory disclosure attacks based on memory deduplication that are able to detect the security level of attacked operating systems, find vulnerable applications, and confirm the status of attacked applications.

Two years later, Barresi et al. [14] developed an attack using the same technique, leaking the address space layouts [15] of the victim virtual machines, while Gruss et al. [16] presented a memory-disclosure attack in sandboxed JavaScript, without the need for the victim to execute any program, merely visiting a remote website controlled by the attacker. Rong et al. [17] even proposed a practical protocol of Cloud Covert Channel based on Memory Deduplication (CCCMD).

B. Shared Memory + Cache Side-Channels

In addition to the techniques presented in the previous section, there is a rich literature of side-channel attacks that combine memory deduplication with cache covert channels. These techniques rely on memory sharing with the victim virtual machine. When an attacker accesses to one of these shared pages, she is accessing to the same physical page frame that the victim is using. As a consequence, that page is located in the same cache line for both attacker and victim, due to the physically-indexed Last-Level-Cache (LLC).

In 2014, Yarom et al. [18] [19] introduced the `Flush+Reload` technique as an extension of a previous study about cache side-channel attacks [20]. It consists in measuring the access time of a specific cache line in the LLC, instead of measuring the writing time to a COWed page. The attacker flushes the monitored cache line and waits for the victim to access the memory line. Given that the victim page is shared with the attacker, they share the cache set for that page, and the attacker can ensure that a specific memory line is evicted from the entire cache hierarchy. Then, if the victim accesses to the data while the attacker is waiting, the monitored cache line will be filled again. In the last phase, the attacker reloads the cache line and measures how much time it takes. If the victim has accessed the memory line, the time will be short. Otherwise, the cache line will be empty, and the operation will be longer. Given that this technique is using the LLC, the attacker and the victim can be running in a different execution core of the physical machine and the attack will still work. With this technique, the authors achieved a successful extraction of GnuPG private encryption keys from a victim, along with the exploitation of a vulnerability introduced to elliptic curve cryptographic protocols, recovering OpenSSL Elliptic Curve Digital Signature Algorithm (ECDSA) nonces.

This technique has been used by several studies since then. Irazoqui et al. [21] [22] retrieved an Advanced Encryption Standard (AES) cryptographic key and private keys from other cryptographic libraries in a cloud environment, using Flush+Reload. Later, they presented another paper where cryptographic libraries are detected across virtual machines, along with their IP addresses. Gülmezoğlu et al. [23] improved the technique and presented a *known-ciphertext only* cache side-channel attack against AES. Benger et al. [24] used Flush+Reload to attack OpenSSL ECDSA signatures.

The `Evict+Reload` technique, introduced by Gruss et al. [25], is a variation of Flush+Reload. It consists of two phases. First, in the profiling phase, the attacker crafts a model (a template [26] [27]) of signals and noise from the cache side-channel traces, which is a matrix with the cache-hit ratio of the address of a specific target event generated on a device that the attacker controls. Secondly, in the exploitation phase, the attacker uses the template matrix previously crafted to deduce events in the system cache, based on the differences of cache hits. After the attack is performed, a report is generated in the attacker machine to be manually analysed. Both phases use standard cache side-channel attack techniques (e.g., Flush+Reload) to obtain the cache hit ratio. Nevertheless, the authors noted that the technique can be adapted to evict a cache line without using the *flush cache line* instruction, thus invalidating a countermeasure proposed by other studies [18] [19] [28]. The method they used to evict a cache line indirectly is to access a physically congruent large array [25].

The `Flush+Flush` technique was presented by Gruss et al. [29]. With it, they achieved a stealthy method to perform cache side-channel attacks without access to the data. Consequently, Flush+Flush does not generate more cache misses than a benign program, avoiding some countermeasures that relied on hardware performance counters for detection [30]–[32]. Instead of measuring the access time of a memory location, they measured the time that the *flush cache line* instruction takes. If the data is cached, this instruction takes more time than if the data is not cached, enabling the side-channel. Furthermore, given that it can work at a higher frequency, Flush+Flush is more efficient than other cache side-channel techniques previously known, in terms of speed.

There is a study that introduces a different technique for victim-memory shared side-channel attacks, which avoids the measurement of time. Disselkoen et al. [33] proposed a method to abuse last level cache in a virtualised system using the Intel Transactional Synchronization Extensions (TSX) instruction set [34] [35]. Intel TSX allows the programmer to specify *transactions* of code, in a way that either all the code is successfully executed (transaction completed) or, if anything fails, all the changes made in memory during the transaction are cancelled (transaction aborted). The authors are able to determine if the cache state has been modified by the victim or not, by means of the hardware callbacks provided by Intel TSX. These callbacks are executed if the victim accesses to the target data. As a consequence, there is no need for timing measurement, given that the attacker gets a side-channel through the state of the transactions (completed or aborted).

C. *Shared Memory + Rowhammer*

Previous techniques exploit the fact that the attacker is sharing memory with the victim in a read-only fashion, to leak sensitive information of all kinds and bypass security mechanisms like Address Space Layout Randomisation [15]. On the other hand, some studies combine this condition with the `Rowhammer` technique [36] to not only read arbitrary data in the victim system but also to write. Rowhammer is a technique that exploits a hardware vulnerability present in many modern Dynamic Random-Access Memory (DRAM) modules. DRAM memory cells can leak their charges to nearby memory rows if they are repeatedly activated in a short period of time, modifying the contents of a row which was not intended to be accessed. As a consequence, an attacker is able to flip bits of arbitrary physical memory locations by repeatedly activating one or both adjacent rows.

Bosman et al. [37] built a “weird machine” that is able to perform a byte-by-byte disclosure of sensitive data of neighbour virtual machines. In this paper, they also disclose high-entropy randomised pointers, providing three different approaches: memory alignment probing, partial reuse, and birthday heap spray. After leaking and gathering all the needed information of the victim using the previous methods, the authors combine memory deduplication with Rowhammer to attack the browser, performing a write operation in a physical page belonging to the victim, without requiring any software vulnerability to perform the attack.

Similarly, Razavi et al. [38] introduced a novel exploitation technique called Flip Feng Shui (FFS). With it, an attacker is able to *exchange* the physical location of a page that is shared with the victim by using memory deduplication. Then, she can trigger a Rowhammer attack and modify their contents, inducing bit flips in a fully controlled way. As a consequence, this technique allows an attacker to write into a page of the victim virtual machine.

IV. Proposed Solution

In this section, we present Slicedup, a memory deduplication design for multi-tenant clouds, where each tenant is the administrator of a group of trust that consists of multiple virtual machines. The sharing is limited to a given group, and thus the sharing of pages belonging to different tenants is not allowed. This way, memory deduplication can be enabled,

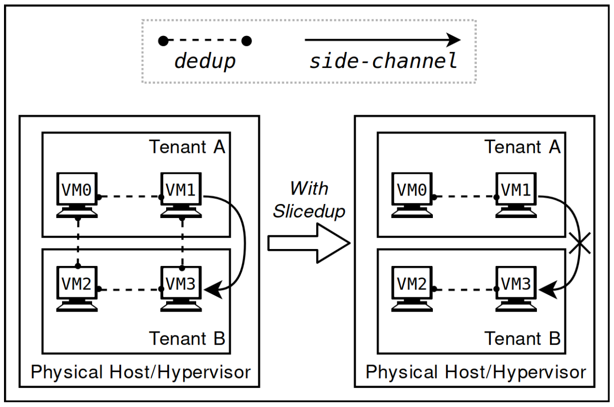

Figure 1. Tenant-aware memory deduplication scheme.

maintaining an equivalent level of security for each tenant, given that pages are never shared across security boundaries. In addition, the deduplication rates for the virtual machines of a specific tenant are not affected.

Our proposed solution is to add tenancy awareness to the deduplication algorithm, identifying every virtual machine based on a Tenant ID. Then, when a page is being processed before being shared with another one, the Tenant ID is combined with the page contents in a way that other pages with the same contents will only be shared with pages belonging to virtual machines of the same tenant. This solution provides a fair trade-off, where the isolation of virtual machines of a tenant is kept along with the sharing effectiveness of memory deduplication.

Figure 1 shows an example, where VM1 of the tenant A is attacking to VM3 of the tenant B. Memory deduplication is applied to all the virtual machines hosted in the same physical machine. Therefore, the attack that VM1 was performing to VM3 is prevented with Slicedup. As a trade-off, this action has a price because the sharing is being limited, and there will exist duplicates of pages from different tenants that would have been merged otherwise.

We have calculated an estimate of the sharing rate based on previous experiments [39] [40], fitting the points into the $log(x)$ function, using the *least-squares fit* technique. Figure 2 shows the resulting function, where the x-axis is the number of virtual machines running at the same time. We can get the amount of memory saved for a given number of virtual machines.

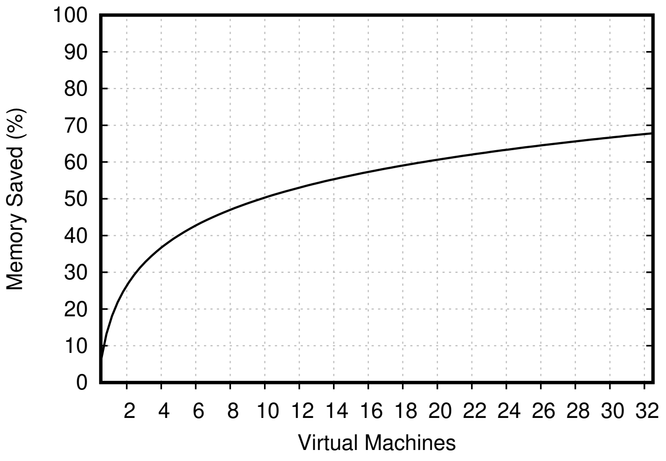

Figure 2. Estimated rate of memory saved by deduplication.

Table I shows the estimation of memory saved in a physical host, comparing two cases: with standard memory deduplication and with Slicedup. In the table, we are using the approximation showed by Red Hat [39] assuming that the virtual machines are Windows XP with 1 GiB of RAM. Then, the number of virtual machines in each case is the maximum we can run when memory deduplication is disabled.

Slicedup offers a compromise solution between performance and security, allowing memory sharing without compromising the system security. With it, customers of cloud infrastructure providers can run more virtual machines than if memory deduplication is disabled. For example, when the host has 32 GiB of RAM and 8 tenants, each tenant is allowed to run two more virtual machines.

TABLE I. MEMORY SAVED WITH STANDARD AND SLICEDUP.

Physical	Num. of	Memory Deduplication	
Memory	Tenants	Standard	Slicedup
8 GiB	2	3.76 GiB (47.00%)	2.94 GiB (36.75%)
	4	3.76 GiB (47.00%)	2.11 GiB (26.37%)
	8	3.76 GiB (47.00%)	1.29 GiB (16.12%)
16 GiB	2	9.17 GiB (57.31%)	7.52 GiB (47.00%)
	4	9.17 GiB (57.31%)	5.90 GiB (36.87%)
	8	9.17 GiB (57.31%)	4.20 GiB (26.25%)
32 GiB	2	21.6 GiB (67.50%)	18.30 GiB (57.18%)
	4	21.6 GiB (67.50%)	15.00 GiB (46.87%)
	8	21.6 GiB (67.50%)	11.70 GiB (36.56%)

V. SOLUTION DISCUSSION

Memory deduplication was designed as a performance technique to increase the memory savings of a system. It has been proved [41]–[44] that it offers an effective and efficient improvement of the physical memory resources management when it is enabled. However, the side-channel produced by memory deduplication is a security problem for the clients of cloud infrastructure providers. Although several studies proposed different possible countermeasures to avoid this issue, there is no consolidated solution which offers similar performance than the original implementations and provides a defence to all the possible attacks without adding complexity.

For example, Payer [32] proposed HexPADS as an anomaly detection system. It is a signature-based Attack Detection System that relies on performance counters. HexPADS analyses running processes, measuring their performance and checking a set of signatures. It is able to detect long running side-channel attacks with low overhead. Nevertheless, it is prone to false positives and false negatives. Besides, as a signature-based system, it doesn't detect new attacks (unknown signature). Furthermore, given that it is based on performance counters, other advanced and stealthy cache attacks can bypass the detection, such as Flush+Flush.

Oliverio et al. [45] proposed VUsion as a new design of memory deduplication that cuts information leaks and Rowhammer attacks based on memory deduplication. It hides the ability of the attackers to distinguish between shared pages and non-shared pages, thus reducing the attack surface. To achieve this, the authors follow a fundamental principle that they call Same Behavior (SB), which means that the attacker will obtain the same results whether the page being tested is merged or not. VUsion allows page sharing among different tenants with an acceptable memory sharing rate. On the other hand, the Same Behavior principle reduces the pages that can be candidates to be merged (only idle pages). Unfortunately, VUsion is intricate to implement. Author's *proof of concept* implementation relies on using reserved bits of the Page Table Entry (PTE) as an alternative to avoid the use of the `present` bit, which would need intrusive changes to the Linux kernel.

Other solutions were proposed, for example, to encrypt the memory of a given process to avoid deduplication [9] [18], software diversification to detect anomalies [18], to share only pages containing zeros [37], or to completely disable memory deduplication [28]. However, those ideas didn't get to consolidate a suitable solution that would provide secure memory sharing.

Slicedup achieves its purpose with a straightforward and simple but effective approach. It merges not only idle pages but also the active ones (as standard deduplication). There are also a few drawbacks to this approach. The memory sharing efficiency is tied to the number of tenants present in a physical host. Given that sharing pages among different tenants is not allowed, the set of pages that can be potentially shared decreases when more tenants are located in the same physical host. Besides, although Slicedup is providing protection among different tenants, it can not protect virtual machines residing in the same tenant. In that scenario, information disclosure and physical memory massaging [38] are still feasible. However, Slicedup offers strong protection on systems where all virtual machines in a particular tenant trust each other. Consequently, attacks from distrusting tenants are ineffective.

VI. Conclusions and Future Work

In this paper, we have presented Slicedup, a memory deduplication design for multi-tenant cloud systems. Slicedup offers a trade-off between sharing pages of all the virtual machines, which weakens the system, and disabling entirely memory deduplication, which means loss of memory savings.

With Slicedup, the virtual machines belonging to the same tenant are sharing their memory because they are part of a group of trust, but not among tenants. As we showed, this is preventing side-channel attacks among machines belonging to different tenants and, at the same time, it provides good memory savings. Our analysis showed that Slicedup prevents side-channels attacks while offering similar memory savings when the number of tenants per physical host is low, and around a 50% of memory savings when the number of tenants is higher.

In future work, a proper evaluation of this approach using benchmarks needs to be done, comparing it with existing solutions and measuring performance and memory saving rates.

References

[1] M. Armbrust et al., "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, 2010, pp. 50–58.

[2] Compaq Computer Corporation. Internet solution division strategy for cloud computing. [Retrieved: Sep, 2018]. [Online]. Available: http://www.technologyreview.com/sites/default/files/legacy/compaq_cst_1996_0.pdf [retrieved: November, 1996]

[3] H. Marco-Gisbert and I. Ripoll, "Preventing brute force attacks against stack canary protection on networking servers," in Network Computing and Applications (NCA), 2013 12th IEEE International Symposium on. IEEE, 2013, pp. 243–250.

[4] K. Braden et al., "Leakage-resilient layout randomization for mobile devices." in NDSS, 2016.

[5] Red Hat. Disabling ksm service to avoid memory deduplication as an advanced exploitation vector. [Online]. Available: https://access.redhat.com/solutions/2356551 [retrieved: Sep, 2018]

[6] C. Huffman. Memory combining in windows 8 and windows server 2012. [Online]. Available: https://blogs.technet.microsoft.com/clinth/2012/11/29/memory-combining-in-windows-8-and-windows-server-2012/ [retrieved: Sep, 2018]

[7] P. J. Denning, "The locality principle," in Communication Networks And Computer Systems: A Tribute to Professor Erol Gelenbe. World Scientific, 2006, pp. 43–67.

[8] 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, ser. EUROSEC '11. New York, NY, USA: ACM, 2011, pp. 1:1–1:6. [Online]. Available: http://doi.acm.org/10.1145/1972551.1972552

[9] ——, "Software side channel attack on memory deduplication," in ACM Symposium on Operating Systems Principles (SOSP 2011), Poster session, 2011.

[10] R. Owens and W. Wang, "Non-interactive os fingerprinting through memory de-duplication technique in virtual machines," in Proceedings of the 30th IEEE IPCCC, ser. PCCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/PCCC.2011.6108094

[11] J. Xiao, Z. Xu, H. Huang, and H. Wang, "A covert channel construction in a virtualized environment," in Proceedings of the 2012 ACM Conference on Computer and Communications Security, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 1040–1042. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382318

[12] ——, "Security implications of memory deduplication in a virtualized environment," in Proceedings of the 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), ser. DSN '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1109/DSN.2013.6575349

[13] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Implementation of a memory disclosure attack on memory deduplication of virtual machines," IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences, vol. E96-A, no. 1, 2013, pp. 215–224, qC 20170104.

[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). Washington, D.C.: USENIX Association, 2015. [Online]. Available: https://www.usenix.org/conference/woot15/workshop-program/presentation/barresi

[15] Pax address space layout randomization (aslr). [Retrieved: Sep, 2018]. [Online]. Available: https://pax.grsecurity.net/docs/aslr.txt

[16] D. Gruss, D. Bidner, and S. Mangard, "Practical memory deduplication attacks inźsandboxed javascript," in Proceedings, Part I, of the 20th European Symposium on Computer Security – ESORICS 2015 - Volume 9326. New York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 108–122. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-24174-6_6

[17] H. Rong, H. Wang, J. Liu, X. Zhang, and M. Xian, "Windtalker: An efficient and robust protocol of cloud covert channel based on memory deduplication," in 2015 IEEE Fifth International Conference on Big Data and Cloud Computing, Aug 2015, pp. 68–75.

[18] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in Proceedings of the 23rd USENIX Conference on Security Symposium, ser. SEC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 719–732. [Online]. Available: http://dl.acm.org/citation.cfm?id=2671225.2671271

[19] Y. Yarom and N. Benger, "Recovering openssl ecdsa nonces using the flush+reload cache side-channel attack," IACR Cryptology ePrint Archive, vol. 2014, 2014, p. 140.

[20] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games – bringing access-based cache attacks on aes to practice," in Proceedings of the 2011 IEEE Symposium on Security and Privacy, ser. SP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 490–505. [Online]. Available: http://dx.doi.org/10.1109/SP.2011.22

[21] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, Wait a Minute! A fast, Cross-VM Attack on AES. Cham: Springer International Publishing, 2014, pp. 299–319. [Online]. Available: https://doi.org/10.1007/978-3-319-11379-1_15

[22] ——, "Lucky 13 strikes back," in Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ser. ASIA CCS '15. New York, NY, USA: ACM, 2015, pp. 85–96. [Online]. Available: http://doi.acm.org/10.1145/2714576.2714625

[23] B. Gülmezoğlu, M. S. İnci, G. Irazoqui, T. Eisenbarth, and B. Sunar, "A faster and more realistic flush+reload attack on aes," in Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design - Volume 9064, ser. COSADE 2015. New

York, NY, USA: Springer-Verlag New York, Inc., 2015, pp. 111–126. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-21476-4_8

[24] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom, "“ooh aah... just a little bit” : A small amount of side channel can go a long way," in Cryptographic Hardware and Embedded Systems – CHES 2014, L. Batina and M. Robshaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 75–92.

[25] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in Proceedings of the 24th USENIX Conference on Security Symposium, ser. SEC'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 897–912. [Online]. Available: http://dl.acm.org/citation.cfm?id=2831143.2831200

[26] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in Cryptographic Hardware and Embedded Systems - CHES 2002, B. S. Kaliski, ç. K. Koç, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 13–28.

[27] B. B. Brumley and R. M. Hakala, "Cache-timing template attacks," in Advances in Cryptology – ASIACRYPT 2009, M. Matsui, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 667–684.

[28] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant side-channel attacks in paas clouds," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 990–1003. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660356

[29] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in Detection of Intrusions and Malware, and Vulnerability Assessment, J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds. Cham: Springer International Publishing, 2016, pp. 279–299.

[30] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," Cryptology ePrint Archive, Report 2015/1034, 2015, https://eprint.iacr.org/2015/1034.

[31] N. Herath and A. Fogh, "These are not your grand daddys cpu performance counters–cpu hardware performance counters for security," Black Hat Briefings, 2015.

[32] M. Payer, "Hexpads: A platform to detect “stealth” attacks," in Engineering Secure Software and Systems, J. Caballero, E. Bodden, and E. Athanasopoulos, Eds. Cham: Springer International Publishing, 2016, pp. 138–154.

[33] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+abort: A timer-free high-precision l3 cache attack using intel TSX," in 26th USENIX Security Symposium (USENIX Security 17). Vancouver, BC: USENIX Association, 2017, pp. 51–67. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/disselkoen

[34] P. Hammarlund et al., "Haswell: The fourth-generation intel core processor," IEEE Micro, vol. 34, no. 2, 2014, pp. 6–20.

[35] P. Guide, "Intel® 64 and ia-32 architectures software developer's manual," Volume 3B: System programming Guide, Part, vol. 2, 2011.

[36] Kim et al., "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," SIGARCH Comput. Archit. News, vol. 42, no. 3, Jun. 2014, pp. 361–372. [Online]. Available: http://doi.acm.org/10.1145/2678373.2665726

[37] 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, SP 2016, San Jose, CA, USA, May 22-26, 2016, 2016, pp. 987–1004. [Online]. Available: https://doi.org/10.1109/SP.2016.63

[38] K. Razavi et al., "Flip feng shui: Hammering a needle in the software stack," in 25th USENIX Security Symposium (USENIX Security 16). Austin, TX: USENIX Association, 2016, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[39] "Linux 2.6.32 release announcement," Dec 2009, [Retrieved: Sep, 2018]. [Online]. Available: https://kernelnewbies.org/Linux_2_6_32

[40] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Effects of memory randomization, sanitization and page cache on memory deduplication," in Proc. European Workshop on System Security (EuroSec 2012) :, 2012, qC 20170104.

[41] D. Gupta et al., "Difference engine: Harnessing memory redundancy in virtual machines," in Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 309–322. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855763

[42] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using ksm," in In OLS, 2009.

[43] N. Rauschmayr and A. Streit, "Reducing the memory footprint of parallel applications with ksm," in Facing the Multicore-Challenge III. Springer, 2013, pp. 48–59.

[44] Y. Deng, X. Huang, L. Song, Y. Zhou, and F. Wang, "Memory deduplication: An effective approach to improve the memory system," Journal of Information Science and Engineering, vol. 33, no. 5, 2017, pp. 1103–1120.

[45] M. Oliverio, K. Razavi, H. Bos, and C. Giuffrida, "Secure page fusion with vusion," in Proceedings of the 26th Symposium on Operating Systems Principles, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 531–545. [Online]. Available: http://doi.acm.org/10.1145/3132747.3132781