Memory Band-Aid: A Principled Rowhammer Defense-in-Depth

Carina Fiedler[1], Jonas Juffinger[1], Sudheendra Raghav Neela[1], Martin Heckel[2]
Hannes Weissteiner[1], Abdullah Giray Yağlıkçı[3], Florian Adamsky[2], Daniel Gruss[1]
[1] Graz University of Technology, [2] Hof University of Applied Sciences, [3] ETH Zürich
{firstname.lastname}@tugraz.at, {martin.heckel.2, florian.adamsky}@hof-university.de, giray.yaglikci@safari.ethz.ch

***Abstract*—Rowhammer bit flips in DRAM enable software attackers to fully compromise a great variety of systems. Hardware mitigations can be precise and efficient, but they suffer from long deployment cycles and very limited or no update capabilities. Consequently, refined attack methods have repeatedly bypassed deployed hardware protections, leaving commodity systems vulnerable to Rowhammer attacks.**

In this paper, we present Memory Band-Aid, a principled defense-in-depth against Rowhammer. Memory Band-Aid is no replacement for long-term, efficient hardware mitigations, but instead a defense-in-depth that is activated when hardware mitigations are insufficient for a specific system generation. For this purpose, Memory Band-Aid introduces per-thread and per-bank rate limits for DRAM accesses in the memory controller, ensuring that the minimum number of row activations for Rowhammer bit flips cannot be reached. We implement a proof-of-concept of Memory Band-Aid on Ubuntu Linux and test it on 2 Intel and 2 AMD systems, building on global bandwidth limits due to the lack of per-bank limits in current hardware. Using this PoC, we find that a full implementation including minor hardware changes would have a low overhead of 0 % to 9.4 % on a collection of realistic Phoronix macro-benchmarks. In a micro-benchmark to cause DRAM pressure, we observe a slowdown by a factor of 1 to 5.1. Both overheads only apply to untrusted, throttled workloads, e.g., all userspace programs or only selected sandboxes, such as those in browsers. Especially as Memory Band-Aid can be enabled on demand, we conclude that Memory Band-Aid is an important defense-in-depth that should be deployed in practice as a second defense layer.

I. Introduction

Rowhammer is a software-based fault attack exploiting read disturbance effects in DRAM, undermining system security. Rowhammer was first described as a potential security issue by Kim et al. [1] in 2014. Seaborn and Dullien demonstrated that an attacker can exploit this effect in privilege escalation and sandbox escape attacks [2]. An adversary can cause the Rowhammer effect by rapidly reading the content of memory rows (hammering), which can cause bit flips, *i.e.*, bit errors, in nearby memory rows. Over the years, the research community has developed various hammering patterns to induce bit flips, e.g., single-sided hammering [1], [2], and double-sided hammering [2]. At the same time, both the research community and industry have investigated ways to mitigate these attacks.

The focus of industry research on Rowhammer mitigations was mainly on mitigations in hardware. A central reason is that hardware-level mitigations can be self-contained within the DRAM module or involve only the DRAM module and the memory controller. Hence, a DRAM vendor can, without coordinating with software vendors or releasing software themselves, deploy a hardware-level mitigation in a new DRAM chip generation, transparent to the remainder of the system. Furthermore, hardware mitigation can be very efficient: Rowhammer access pattern detection can be implemented directly in hardware and when necessary, the hardware, in particular the DRAM module and the memory controller, can very quickly take action (e.g., refresh victim rows within a few nanoseconds) [1], [3]. However, hardware-level mitigation mechanisms suffer from two shortcomings. First, they do *not* apply to already deployed systems and require a long time to be implemented in commodity systems as they require changes in the hardware. Second, hardware-level mitigations can fundamentally become stale as (1) newer DRAM chips get more vulnerable to Rowhammer [4], (2) new aspects of the Rowhammer phenomenon are discovered [5], and (3) new attacks are developed [6]–[10].

Researchers repeatedly refined their attacks to bypass the hardware mitigations deployed to secure DRAM. An early mitigation was to increase the refresh rate [1]. However, this approach significantly reduces the performance, increases power consumption, and does not prevent Rowhammer attacks [1]. ECC RAM is also not suitable for defending against Rowhammer attacks since it can only correct a single bit per word. Cojocar et al. [11] demonstrated a reliable Rowhammer attack against ECC DIMMs. For DDR4, the industry developed Target Row Refresh (TRR), which tracks DRAM accesses and refreshes potential victim rows. Several studies [6], [7], [10], [12], [13] found ways to bypass TRR using more advanced hammering patterns. Even on systems with Per-Row Activation Counting (PRAC), a hardware mitigation defined in the DDR5 specification, Jattke et al. [14] observed bit flips. Thus, some commodity systems remain vulnerable to Rowhammer attacks, requiring mitigations beyond the existing hardware solutions.

In this paper, we present Memory Band-Aid, a principled software-level defense-in-depth against Rowhammer, requiring

Network and Distributed System Security (NDSS) Symposium 2026
23 - 27 February 2026 , San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.230156
www.ndss-symposium.org

only minimal hardware support to limit memory bandwidth. Memory Band-Aid exploits that an attacker cannot cause Rowhammer bit flips if they cannot reach the minimum required number of row activations. Consequently, by limiting the attacker's maximum number of row activations, Memory Band-Aid mitigates Rowhammer. Memory Band-Aid is a defense-in-depth that can be applied, like a band-aid, when hardware mitigations are discovered to be insufficient on a specific system generation. It is no replacement for long-term and efficient hardware mitigations against Rowhammer, but acts as a second defense layer. Still, in contrast to existing stop-gap solutions like error-correcting codes, doubled refresh rates, and disabling specific instructions, Memory Band-Aid is a principled defense.

Our design of Memory Band-Aid introduces per-thread and per-bank rate limits for DRAM accesses and implements these in the processor's memory controller. Our analysis of current Intel and AMD systems shows that memory bandwidth limits are already available on AMD processors since Zen 2, under the name "L3 Bandwidth External" (L3BE) limit [15], and on Intel Xeon processors since Xeon 2nd Gen, under the name "Memory Bandwidth Allocation" (MBA) [16], [17]. However, we observe several practical limitations on both Intel and AMD: First, Intel only supports limits per physical core on the processor side, and only started with Intel Xeon 3rd Gen processors to support actual rate limiting in contrast to artificial delays [16], [17]. Second, on both Intel and AMD, the rate limits are applied to the entire DRAM and are not implemented on a per-bank granularity. Hence, on these commodity systems, we would have to impose a limit that is 64–256 times lower than Memory Band-Aid requires on a per-bank level. Still, we implement a proof-of-concept of Memory Band-Aid on Ubuntu Linux (22.04 and 24.04), leveraging the global memory bandwidth limit like a per-bank limit. We evaluate Memory Band-Aid on 2 Intel and 2 AMD systems, reporting the measured overheads for a simulated full implementation of our design as well as an effective implementation on current hardware.

In our evaluation, we use representative realistic workloads and worst-case workloads. For the worst-case for Memory Band-Aid, we run a small micro-benchmark that consists of only a tight loop of AVX512 loads. In this worst-case micro-benchmark, Memory Band-Aid slows down the loop by a factor of 1 to 5.1 with per-bank limits. For realistic macro-benchmarks, we used the Phoronix Productivity test suite, constituting 17 benchmarks (9 GEGL operations, 4 GIMP benchmarks, Inkscape, LibreOffice, GNU Octave, librsvg) representative of Desktop user's workloads as well as Chrome headless to cover an application with a sandbox and multiple processes involving inter-process communication. In the macro-benchmarks, we observe overheads of $0\,\%$ to $9.4\,\%$ for a Memory Band-Aid with per-bank limits. These overheads are only applied to untrusted workloads, e.g., userspace programs or sandboxes, letting the remaining system run at full performance. We also show that without a per-bank limit, *i.e.*, an implementation that is secure on current hardware, overheads are prohibitively high, *i.e.*, up to $349\times$ in micro-benchmarks and up to $4.3\times$ in macro-benchmarks. From this, we conclude that vendors need to extend their hardware support for memory limits to a per-bank level and that Memory Band-Aid should be deployed in practice as an effective defense-in-depth against Rowhammer.

In summary, we make the following main contributions:

- We present a principled Rowhammer defense-in-depth in software using hardware-based *per-bank* bandwidth limits that are *cross-bank* already available on AMD and Intel.
- Memory Band-Aid is adaptive and can be configured to mitigate current and future Rowhammer attacks, with strict security guarantees keeping row activations below the required Rowhammer bit-flip threshold.
- We implement a proof-of-concept of Memory Band-Aid on Ubuntu Linux, on 2 Intel and 2 AMD systems.
- We evaluate a full Memory Band-Aid implementation, yielding practical overheads in micro-benchmarks ($\leq 5.1\times$) and macro-benchmarks ($0\,\%$ to $9.4\,\%$).

Outline. Section II provides background on DRAM, Rowhammer, and bandwidth control. Section IV presents the design of Memory Band-Aid. Section III evaluates the security of Memory Band-Aid. Section V details current hardware support. Section VI evaluates the performance and security on a proof-of-concept implementation. Section VII discusses impact and limitations. Section VIII concludes.

II. Background

In this section, we provide background on DRAM, Rowhammer attacks, Rowhammer mitigations in hardware and software, and discuss memory-bandwidth control mechanisms.

A. DRAM

A DRAM module contains one or multiple DRAM ranks that time-share a memory channel connected to the processor's memory controller. A DRAM rank consists of multiple DRAM chips, each split into multiple DRAM banks. DRAM cells are organized in a two-dimensional array of rows and columns in a DRAM bank and are internally accessed at a DRAM row granularity. A DRAM cell stores 1 bit of data as an electrical charge of a capacitor and is accessed via an access transistor.

To access a DRAM cell, the memory controller performs an operation called *activation*, which opens the row with the requested data and fetches the row's content into a buffer structure called row buffer. The memory controller can access the data in the row buffer in a DRAM word granularity (e.g., 64 bytes). An activated row is closed with an operation called *precharge*. DRAM cells are inherently volatile, *i.e.*, they loose charge over time and need to be refreshed periodically.

B. Rowhammer

Accesses to DRAM cells cause increased charge leakage in other physically nearby DRAM cells, which are not accessed. Rowhammer is a prime example of DRAM read disturbance, where repeatedly activating a DRAM row (*i.e.*, aggressor row) can induce bit flips in other physically nearby

DRAM rows (*i.e.*, victim rows), by draining their charge. Prior works experimentally demonstrate that the number of row activations needed to induce the first bit flip in a victim row, *i.e.*, Rowhammer threshold, has reduced by more than an order of magnitude over the last decade from around 139K in 2014 [1] down to around 7K activations in 2025 [18]. Many prior works demonstrate various reliable procedures to perform a Rowhammer attack on DDR3 [1], [9], [19]–[28], DDR4 [6]–[9], [12]–[14], [22], [27]–[35], DDR5 [14], LPDDR2 [31], [36], LPDDR3 [36], [37], LPDDR4 [6], [36], and LPDDR4X [6], [7], [12] DRAM chips. Most of these works use flush instructions to evict cache lines from memory, effectively forming a very tight Flush+Reload [38] loop.

Following the first demonstration of Rowhammer as a widespread vulnerability [1], Seaborn and Dullien [2] demonstrated two Rowhammer exploits: A local privilege escalation based on page table entries (PTE) and a sandbox escape. Later on, many works demonstrated Rowhammer attack variations including using JavaScript code within the web browser [13], [22], GPU kernels [39], and network packets [24], [31]. Along the way, new experimental findings led to various access patterns and attack strategies, including single- and double-sided hammering of one or two neighbors of a victim row [1], [2], one-location hammering only a single row in a bank [9], half-double hammering far aggressors with few accesses to near aggressors [6], many-sided hammering many aggressor rows simultaneously [12], and access patterns to extend row-open time frames [5], [8]. Modern Rowhammer attacks employ fuzzing strategies that explore the most effective many-sided access patterns to induce bit flips in victim rows [7], [13], [34]. Today, Rowhammer attacks are practical on many systems using Intel [1], Arm [36], and AMD [14] processors, and even RISC-V-based platforms [35]. Some trusted execution systems (e.g., SGX) can defend systems against Rowhammer attacks at the cost of significantly reduced availability due to exposure to denial-of-service attacks [9], [29].

C. Rowhammer Hardware Mitigations

Three early Rowhammer mitigations are increased refresh rate [40], [41], error correcting codes (ECC), and probabilistic adjacent row activation (PARA) [1]. An increased refresh rate incurs prohibitively large performance and energy overheads and ECC's fundamental assumption of uniformly distributed bit errors is *not* correct for Rowhammer, *i.e.*, it can still be attacked [11]. PARA [1] successfully mitigates Rowhammer by refreshing potential victim rows after an aggressor row activation with a very low probability, albeit with a potentially high overhead for lower Rowhammer thresholds [4].

Early solutions' limitations and overheads led the DRAM manufacturers to introduce TRR [42]–[44], which uses a small set of counters to track DRAM accesses and refresh potential victim rows. TRR implementations are proprietary and *not* necessarily secure: Several recent works defeat TRRs in real modern DRAM by exploiting that they do not track all aggressor rows [7], [12]–[14], and do not account for Rowhammer effects in rows in > 1 distance [6]. These findings suggest that deployed hardware mitigations have not been sufficient to mitigate Rowhammer attacks in practice. With DDR5, an *optional* new in-DRAM hardware mitigation called PRAC [45] is introduced. PRAC enables the DRAM chip to (1) internally track activation counts precisely per DRAM row, and (2) demand refresh time windows from the memory controller. Canpolat et al. [46] show that PRAC works even for low Rowhammer thresholds but incurs significant slow downs and even enables denial-of-service attacks. There is also *no* guarantee that a DRAM chip implements PRAC, given that major DRAM manufacturers have not widely implemented similar mechanisms in the past, e.g., many DDR4 DRAM chips report a maximum activation count of infinite [12].

A large body of prior works (e.g., [1], [3], [47]–[83]) propose hardware-level mitigations to prevent read disturbance bit flips. These proposals provide a map to different design points in the trade-off space between performance, area overhead, and security. However, a common limitation of these proposals is that they are not widely deployed in practice, and many of them require significant changes to the memory controller or DRAM chips. For example, some proposals require additional hardware to track row activations or to implement new refresh policies, which may not be feasible for existing systems. There are two outstanding limitations of these hardware-level mitigations: First, the security of these mechanisms relies on the fundamental assumption that "the system designer knows the minimum activation count at which a Rowhammer bit flip might occur." However, this assumption is *not* correct because exploring all aspects of Rowhammer is still an active research field, and recent works demonstrate that new insights into Rowhammer can be exploited to achieve bit flips at significantly lower activation counts [5], [8], [18], [76]. Second, DRAM protocols do *not* mandate implementing *any* of these Rowhammer mitigation mechanisms to either DRAM or processor manufacturers [12], [46]. Therefore, despite this large body of research, modern and future systems might still be vulnerable to Rowhammer.

D. Software-level and Software-controlled Mitigations

Different software-based approaches are used to mitigate Rowhammer. Closest to the working principle of hardware mitigations is ANVIL [84]. Instead of detecting Rowhammer attacks in hardware, ANVIL uses existing performance counters and refreshes victim rows in software. However, similar to early hardware-level defenses, this approach suffers from not detecting and not mitigating all attack variations.

Several defenses prevent bit flips in victim memory by altering memory allocation mechanisms. CATT [85] extends the physical memory allocator to isolate kernel and user space. An attacker can only map pages that are physically separated from the pages the kernel allocates. GuardION [86] prevents DMA-based Rowhammer attacks on Arm devices by isolating DMA buffers with guard rows. An attacker can not map memory from rows next to DMA buffers. RIP-RH [87] isolates the memory of different processes using guard rows between each process' memory. ZebRAM [88] implements a similar

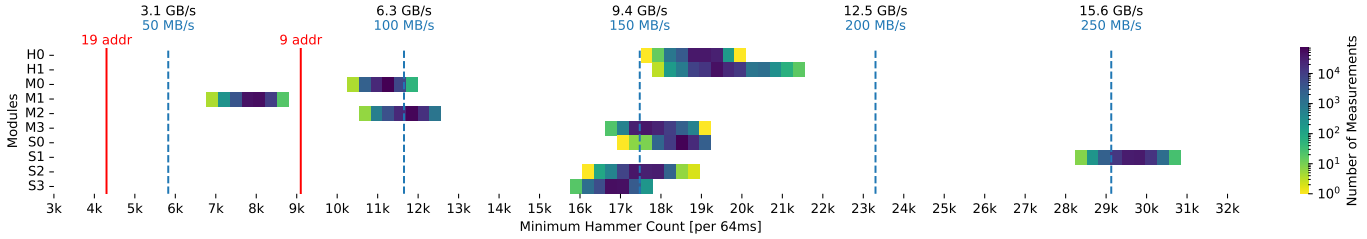


Fig. 1. Distribution of minimum hammer count per aggressor of 10 tested DDR4 modules [18]. Vertical blue dashed lines indicate required per-bank (blue) and resulting overall (black) memory bandwidth to perform 9-sided Rowhammer on a system with 64 banks. Red solid lines indicate the hammer count limits that can be enforced with Memory Band-Aid for 9- and 19-sided attacks with currently available hardware support.

approach by isolating every DRAM row containing data with guard rows. These guard rows can be used as swap space. Zhang et al. [89] showed that kernel interfaces can also act as confused deputies in Rowhammer attacks. Mitigating their attack would require a defense that can limit user and kernel memory accesses equally.

Copy-on-Flip [90] is a software-based solution that improves the resistance of ECC DRAM against Rowhammer attacks such as the one demonstrated by Cojocar et al. [11] by relocating memory pages when ECC faults are observed.

CSI:Rowhammer [91] is another software level defense although requiring minimal hardware changes. CSI:Rowhammer also assumes ECC DRAM but replaces the ECC code with a strong cryptographic MAC. While yielding strong guarantees for mismatch detection, a cryptographic MAC cannot correct, hence requiring error correction in software.

In summary, existing mitigations have drawbacks, such as high overheads, the need for specific hardware features, or the assumption of specific attack scenarios that can be bypassed. A software-level defense-in-depth with strong security guarantees would be a crucial building block for sustainably mitigating Rowhammer attacks in practice.

E. Memory Bandwidth Limits

Both Intel and AMD implement configurable memory bandwidth limits as a quality-of-service mechanism. Memory Bandwidth Allocation (MBA) is a feature in Intel's Resource Director Technology (RDT) suite which enables control over memory bandwidth [16], [17]. AMD offers a comparable feature within their Platform Quality of Service suite to specify memory bandwidth limits, referred to as L3 External Bandwidth Enforcement (L3BE) [15]. The Intel MBA and AMD L3BE features can help manage applications with excessive bandwidth use in environments where they are co-located with other workloads, such as in data centers [92]. While Intel plans for the next generation of MBA to operate on a per-SMT-thread rather than per-core basis [93], currently only AMD L3BE supports different limits for SMT sibling threads.

Class of Service. System software designates a numeric Class of Service (COS) to each logical core via a model-specific register (MSR). Each COS can be configured with a distinct memory bandwidth limit, thereby limiting the memory bandwidth for the cores which were assigned that COS. During a context switch, system software can adjust program priorities by modifying the COS as needed.

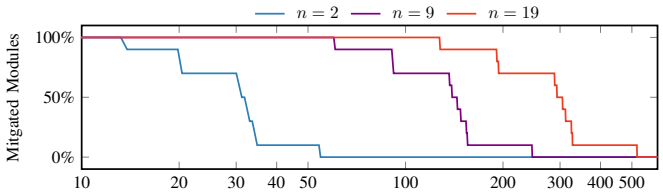


Fig. 2. Per-bank Bandwidth Limit [MiB/s], Percentage of tested DDR4 modules [18] on which n-sided Rowhammer attacks can be mitigated with per-bank memory bandwidth limits. Many-sided attacks can still be mitigated with higher per-bank bandwidth limits.

Specifying Memory Bandwidth Limits. AMD allows 128 MiB/s increments for memory bandwidth limits, ranging from 128 MiB/s to a theoretical maximum of 256 GiB/s [15]. Specifying the limit to 256 GiB/s is equivalent to unlimited bandwidth. In contrast, Intel permits specifying memory bandwidth limit as a percentage in increments of 10 %, where 0 % indicates no throttling (*i.e.*, unlimited) and 90 % represents the maximum throttling level [16]. Intermediate values are rounded, and attempts to write larger values fail.

Currently, neither Intel MBA nor AMD L3BE support applying the bandwidth limit per DRAM bank. That is, if the accesses to one bank must be limited to a specific value, this value has to be configured as a global limit. The limit is then applied regardless of whether all accesses went to the same DRAM bank or were evenly spread across DRAM banks.

III. Minimum Rowhammer Counts

Rowhammer attacks require high-frequency memory accesses. In this section, we determine how tight memory bandwidth restrictions need to be to prevent Rowhammer bit flips in practice.

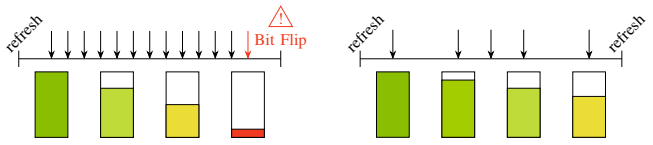
To quantify how often an adversary has to access memory to flip bits, previous work [4], [6], [18], [58], [94]–[96] measured the *minimum hammer count* (also referred to as *read disturbance threshold*). Previous works on Rowhammer attacks have reported a wide range of minimum hammer counts. In fact, more recent DDR versions appear more vulnerable to Rowhammer attacks with minimum hammer counts as low as 7K [18] per aggressor. Multiple systematic analyses of

minimum hammer counts for DDR4 chips [4], [18] have been performed on FPGAs without hardware mitigations. Research done on full systems, *i.e.*, CPU-based attacks, typically require higher hammer counts [96]. The reason for this may be reordering and merging to batch bank accesses in the memory controller. Consequently, memory accesses from the CPU do not directly translate to DRAM bank row activations. Many Rowhammer attack works [8], [12], [13], [34] on CPU-based systems do not experimentally determine the required minimum hammer counts. Instead, they choose hammer counts well above the FPGA-determined thresholds to produce bit flips reliably. Hence, we rely on FPGA-based hammer counts as a secure lower bound.

DDR3 chips require higher minimum hammer counts to induce Rowhammer bit flip than DDR4 and DDR5 chips without mitigations [4]. Hence, Rowhammer bit flips on DDR3 memory are easier to mitigate by restricting memory bandwidth. As DDR3 chips are becoming outdated, and only a few DDR5 DIMMs have been shown to be vulnerable to Rowhammer, we focus our analysis on recent DDR4 chips. However, secure memory bandwidth restrictions apply equally to DDR3 and DDR5, merely with a different configuration for the minimum hammer count. As we assume a system with TRR mitigations in place, preventing double-sided attacks, we focus our analysis on many-sided attacks. We consider one-location Rowhammer [9] and Rowpress [8] out of the scope of the analysis, as these have not been reproduced on newer DDR4 modules.

Native Attacks. Olgun et al. [18] demonstrated that the hammer count varies over time and is therefore difficult to determine precisely. Figure 1 shows the distributions of the minimum hammer count required per aggressor to induce Rowhammer bit flips on the 10 tested DDR4 modules. The hammer count from Olgun et al. [18] ranges from 7K to 31K accesses for different DRAM modules. Hence, limiting the maximum hammer count per aggressor to 7K prevents Rowhammer bit flips on all 10 modules, while a maximum hammer count of 15K still mitigates 7 out of 10 modules. Without hardware mitigation (e.g., TRR), double-sided Rowhammer attacks only require a cumulative hammer count of 14K–62K per refresh interval on the tested modules. How low these numbers are becomes clear when realizing that 7K–14K accesses correspond to reading about 0.5 MiB to 1 MiB of memory from DRAM. However, many-sided attacks circumventing TRR mitigations need a cumulative hammer count of 63K–279K when hammering 9 addresses and a cumulative hammer count of 133K–589K for 19 addresses. In this paper, we focus on the mitigation of many-sided Rowhammer attacks on commodity systems with hardware mitigations in place.

JavaScript-based Attacks. Gruss et al. [22] demonstrated that Rowhammer attacks can be mounted from a website via JavaScript. Instead of the `clflush` instruction, which is unavailable in JavaScript, they use cache eviction. To speed up cache eviction to the level where Rowhammer bit flips can occur, they measured hundreds of eviction strategies and found strategies that have, on average, as little as 2 extra cache misses, while all other eviction accesses are served from the cache. Ridder et al. [13] extended this idea to many-sided hammering from JavaScript. More recently, Ridder et al. [10] showed that this can be further optimized with special memory patterns that lead to postponing of refresh commands in the memory controller. Consequently, they observe more bit flips on more devices from JavaScript.



(a). Many accesses on the same bank inducing a bit flip during the refresh interval.

(b). Limiting number of accesses per refresh interval prevents excessive charge leakage.

Fig. 3. Limiting per-bank memory access frequency to prevent Rowhammer bit flips.

The memory bandwidth required for these attacks is not higher than in native attacks. Hence, the attacks only differ in how long it takes to evict or flush the target memory addresses and reach DRAM. If refresh intervals can be postponed on specific systems for a pre-configured number of additional accesses [10], a suitable memory bandwidth limit can similarly be adjusted to this maximum refresh interval including any potential postponement. Thus, also these advanced and Javascript-based Rowhammer attacks can be mitigated with the same memory bandwidth limits based on the same minimum hammer counts.

IV. Memory Band-Aid

Hardware mitigations are not flexible and typically cannot be adjusted once they are deployed in consumer devices. Hence, when new Rowhammer attacks are discovered, bypassing hardware mitigations, many commodity systems are again unprotected from Rowhammer attacks. The main idea behind Memory Band-Aid is to have a principled software-level defense-in-depth against Rowhammer by limiting the memory bandwidth of untrusted code per DRAM bank. This keeps the number of row activations per refresh interval lower than a Rowhammer attack requires to induce any bit flips as shown in Figure 3. Restricting memory bandwidth per-thread comes at a performance cost for the untrusted threads but leaves other workloads unaffected. While current processors already have limited support for memory bandwidth limits, we can implement a Memory Band-Aid proof-of-concept on these processors that provides strong security guarantees, albeit with a higher performance impact than necessary (see Section VI).

An effective Rowhammer defense based on memory bandwidth limiting needs to fulfill the following criteria: First, the mitigation must limit the memory bandwidth to values that are low enough to prevent a critical number of memory accesses per refresh interval. Second, the mitigation needs to restrict the number of memory accesses in each individual refresh interval rather than just on average over a longer

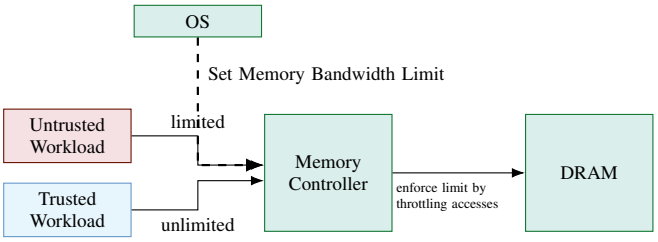


Fig. 4. Overview of Memory Band-Aid. The operating system schedules threads of different processes and instructs the memory controller on when to limit the memory bandwidth.

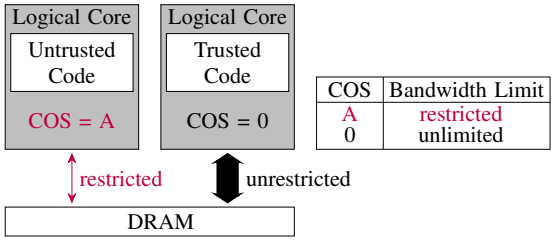


Fig. 5. Memory Band-Aid proof-of-concept implementation. Logical cores are associated with Classes of Service. Classes of Service are assigned different memory bandwidths.

period of time. Third, the memory bandwidth limit must not be bypassed or modified by any attacker-controlled mechanisms. To avoid any performance impact for benign processes, a defense must provide per-thread memory bandwidth limits that can be dynamically managed by the operating system. As a consequence, this also allows to adjust memory bandwidth limits via an update to the software configuration when new insights on Rowhammer susceptibility arise.

Figure 4 provides an overview of the working principle of Memory Band-Aid. For Memory Band-Aid, we focus on a scenario in which commodity systems with full Rowhammer hardware-mitigations (e.g., TRR) are still vulnerable to many-sided Rowhammer attacks. Memory Band-Aid can be implemented for different threat models, classifying all userspace applications as untrusted or only restricting specific sandboxes, e.g. browser. Memory Band-Aid introduces a new operating system interface (e.g., in `/sys` on Linux), allowing a root-privileged user to configure the maximum number of per-bank row activations. The operating system deduces the corresponding memory limit and configures the hardware to not exceed this per-bank memory bandwidth limit. The operating system applies these limits upon context switch, *i.e.*, it can enable and disable the limit for specific workloads, such that only untrusted workloads are affected by the memory limit. The memory controller enforces the limit, throttling memory accesses of untrusted workloads appropriately while providing full memory bandwidth to trusted workloads. In fact, limiting the memory bandwidth of a memory-intensive untrusted workload can even speed up the performance of simultaneously running trusted workloads, as more bandwidth is available to these trusted workloads.

Adaptive Attacks. Since our mitigation does not make any assumptions about the access patterns, it is also resilient against future Rowhammer variations, and it can be configured dynamically to mitigate Rowhammer on any system. We derive formal security guarantees in Appendix C. Since Memory Band-Aid enforces a memory bandwidth limit in the memory controller, its security is not affected by software-initiated bursts of memory accesses: at the memory controller it is plainly a stream of memory accesses that is throttled. Multithreaded Rowhammer attacks on a single bank have not been successfully demonstrated yet. Attacks targeting separate banks are independent and are independently mitigated by our per-bank limits. The bandwidth limits configured for Memory Band-Aid are based on the maximum uncore frequency (see Section VI-F). Therefore, an attacker cannot circumvent the limit via uncore frequency scaling.

Proof-of-Concept Implementation. Even though both Intel and AMD offer hardware support to limit memory bandwidth from software, it was designed as a quality of service and not a security feature. Therefore, they lack per-bank controls. Hence, we can only implement partial Memory Band-Aid proof-of-concepts based on the existing hardware features that limits the overall memory bandwidth. The proof-of-concept implementation can be configured in two ways, as we show in Section VI. First, a *security-wise*, functional proof-of-concept where the performance overhead is significantly higher than that of a correct per-bank implementation of the limit. Second, an *unsecure* proof-of-concept where the performance overhead is very to close to the performance overhead of an implementation on a system with per-bank limits. On current Intel and AMD processors, restrictions can be imposed at the granularity of a Class of Service (COS), which can be associated with one or several logical processors. Figure 5 shows a mapping from a logical core to a memory bandwidth via a Class of Service.

In general, we envision Memory Band-Aid to restrict user processes or specific sandboxes (untrusted) but not the kernel (trusted). However, to also mitigate confused-deputy attacks [89], Memory Band-Aid can be also be enabled for kernel workloads, e.g. syscalls of an untrusted process. All performance overheads we report in Section VI are based on this conservative approach: We temporarily restrict a specific logical core to different limits via a Class of Service and taskset the test program on this core. We thereby temporarily restrict all workloads running on this logical core, including syscalls performed by the test program.

We also implement a prototype for a Memory Band-Aid implementation[1], as a kernel patch for Ubuntu 24.04, kernel version 6.11.0, focussing on the userspace-kernelspace security boundary. With minimal code changes of 171 lines, our Memory Band-Aid proof-of-concept configures the Classes of Service such that they specifically restrict the userspace workloads. One COS for kernel workloads is configured to allow unlimited bandwidth, while a second COS for userspace workloads is configured to limit memory bandwidth to a secure

[1] https://github.com/isec-tugraz/MemoryBandAid

limit. On each jump to kernelspace, the logical core is assigned to the unrestricted COS. On each return to userspace, the logical core is assigned to the restricted COS. The memory controller ensures that all memory requests of the program are throttled accordingly. The appropriate secure values for restricted userspace workloads can be configured globally by a privileged user and is automatically applied whenever a userspace program is scheduled. To minimize misconfiguration risk, we propose conservative default values to be set by default, based on published attacks for different Rowhammer-susceptible technology (e.g., DDR3, DDR4, DDR5), for reasonable defaults across most systems that can be updated via software when new attacks emerge.

Instead of implementing Memory Band-Aid at the userspace-kernelspace boundary, it could also be employed for sandboxed environments. In this case, rather than applying limits to all userspace processes, the scheduler checks if the to-be scheduled program is trusted on each context switch. If this is the case, the logical core is assigned to the unrestricted COS. Otherwise, the logical core is assigned to the restricted COS and the memory controller ensures appropriate rate limiting.

V. Limitations on Current Hardware

In this section, we discuss the impact of the limited hardware granularity, in particular per-bank throttling, and the generalizability of Memory Band-Aid. Vendors introduced bandwidth limits as a quality-of-service feature, not a security feature. Therefore, current hardware support lacks tighter bandwidth-, per-bank and per-logical-core limits. With "L3BE", AMD supports bandwidth limits from Zen 2 onwards in their product range, from Ryzen consumer CPUs to Epyc server CPUs. In contrast to AMD, Intel only provides their bandwidth limiting feature ("MBA") on Xeon processors.

Lack of Per-Bank Limits. Figure 2 shows how far the per-bank memory bandwidth has to be limited to mitigate Rowhammer. A seemingly low per-bank limit of 50 MiB/s corresponds to an overall maximum bandwidth of 3.1 GiB/s to 12.5 GiB/s with typical DDR4 and DDR5 setups (with 64 to 256 banks) for the untrusted workload. However, since current hardware does not support per-bank limits, we can only limit the bandwidth across all banks.

Granularity of Memory Bandwidth Limits. Intel supports memory bandwidth limits relative to the system's memory bandwidth. Hence, the available absolute bandwidth restrictions vary between different Intel processors. While Intel states that the range of available settings may differ, all tested Xeon processors support multiples of 10 % up to 90 %. AMD supports more fine-grained 128 MiB/s increments. While we can implement a proof of concept with these memory bandwidth limits on Intel and AMD, the 10 % limit on Intel is not restrictive enough for a secure implementation of Memory Band-Aid, which requires a granularity of the per-bank limit of, e.g., 50 MiB/s to mitigate 9-sided attacks (see Figure 1). We expect this limitation to disappear with future hardware supporting finer granularities and per-bank limits.

Precision of Memory Bandwidth Limits. The configured memory bandwidth is enforced as an upper limit. For the lowest L3BE setting of 128 MiB/s on AMD, it should be possible to access 134.4 thousand cache lines within a refresh interval. Interestingly, for a continuous Flush+Reload loop on a single address, we observe that we can only access roughly 72.7 thousand cache lines within a refresh interval on the Ryzen 7700X. This is surprisingly close to 50 % of the configured maximum bandwidth and indicates that AMD might track and limit memory accesses at the interface between L3 cache and memory controller, where also prior work noticed that Intel surprisingly counted flush operations in some performance counters [97]. This limitation of the current proof-of-concept implementation is not inherent to Memory Band-Aid, potentially leading to an overly pessimistic overhead measurement. However, this also enables us to limit Flush+Reload loops iterations to a **lower** number of iterations with higher memory bandwidth settings than detailed in Figure 1.

While AMD supports finer granularity settings, we observe on the Ryzen 7700X that the intermediate steps in memory bandwidth are not always honored: Linearly increasing the bandwidth limit reveals a fine-grained step function in some cases, where two configured values result in the same (lower) actual bandwidth limit. The security of Memory Band-Aid is unaffected by this. However, it may again influence the performance of our proof-of-concept implementation negatively. Furthermore, it showcases that while the idea of Memory Band-Aid is generalizable to any hardware, the effective security and performance depend on the specific hardware implementation and its adherence to configured limits.

Per-Logical-Core Limits. Current Intel Xeon processors support setting memory control values per logical core, *i.e.*, sibling threads can be limited to different memory bandwidths. However, an unrestricted SMT thread can only utilize the higher bandwidth if the sibling core is not running simultaneously. When two sibling cores with different memory bandwidth limitations run concurrently, the minimum bandwidth is applied to both threads. This is interesting, as it indicates that the limit is implemented in the core, which counts the number of memory accesses to DRAM it caused and not in the memory controller where these memory accesses are performed.

AMD does not restrict two SMT threads to the same bandwidth. However, an unrestricted workload is still affected by a concurrent restricted workload. We scheduled an unrestricted and a fully restricted Flush+Reload memory pressure test on the same physical core on the Ryzen 7700X and observed a slowdown of factor 6 for the unrestricted workload. These SMT side effects only affect performance, but not the security of Memory Band-Aid. Memory bandwidth limits of one physical core do not affect other physical cores (asides from the positive effect of potentially freeing up memory bandwidth).

In our proof-of-concept evaluation, we do not schedule unthrottled trusted workloads on the same physical core as untrusted workloads to maintain high performance for trusted

workloads. However, our proposed full design for Memory Band-Aid does not have this restriction.

VI. Evaluation

In this section, we evaluate Memory Band-Aid, primarily based on our proof-of-concept implementation. We perform measurements to verify the adherence to configured limits and run both micro- and macro-benchmarks to understand the performance impact of Memory Band-Aid and our proof-of-concept implementation on current hardware. We perform all measurements on two AMD machines, Ryzen 7700X (A1) and Epyc 8024P (A2), and two Intel machines, Epyc 8024P (I1) and Xeon 4410T (I2), as listed in Table I. We perform all experiments on a single logical core, which we restrict for the duration of the experiments. Hence, we overapproximate the overhead, as all scheduled applications, including kernel applications (e.g., syscalls), are limited.

We focus our evaluation on the following three aspects: **First**, we show how memory bandwidth limits affect memory-intensive micro-benchmarks, allowing us to infer secure configuration parameters and worst-case numbers for the overhead: The first micro-benchmark is a workload that constantly flushes and reloads a set of memory locations, *i.e.*, a tight Flush+Reload loop (see Section VI-A), the other is a memory sweep (see Section VI-C). **Second**, we show that with full bank parallelism and per-bank limits, the performance impact is negligible. In contrast, a global limit for all accesses, as on current hardware, yields significantly lower performance (see Section VI-B). **Third**, with macro-benchmarks we show how untrusted workloads are affected by per-bank memory bandwidth limits in a full Memory Band-Aid implementation and by global memory bandwidth limits as in our proof-of-concept implementation on current hardware (see Section VI-D). For this purpose, we selected a benchmark collection from the Phoronix benchmark suite that represents typical desktop use. Furthermore, we assess the impact of varying memory bandwidth limits on the Chrome headless browser (see Appendix B).

A. Flush+Reload Memory Bandwidth Limits

In this section, we evaluate the necessary slowdown Memory Band-Aid needs to enforce to mitigate Rowhammer attacks. For this part of the evaluation, we focus on a tight Flush+Reload loop, similar to those used in Rowhammer attacks. However, as we do not attempt to induce Rowhammer bit flips, we select any addresses across all banks. Typical DDR4 and DDR5 memory has 16 384 to 131 072 rows per bank. Thus, implicitly, the chance of hitting the same bank and the same row is practically zero. However, row conflicts occur during a Flush+Reload loop over n random addresses with an expected value of

$$E(n) = \binom{n}{2} \times \frac{1}{b},$$

where b is the number of banks. For an expected value of 1, we need nine addresses for a 32-bank system and 12 for a 64-bank system.[2] However, we emphasize that the number of row conflicts is the same with and without the limit, *i.e.*, these are just naturally occurring with the random address selection.

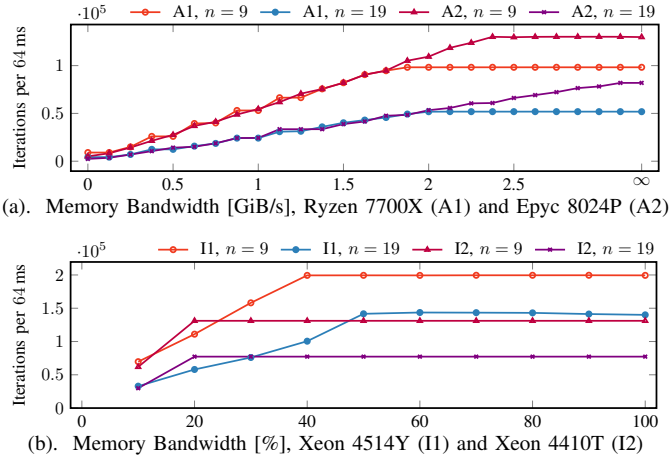


Fig. 6. Maximum iterations of Flush+Reload with $n = 9$ and $n = 19$ addresses for varying memory bandwidth limits on the 4 tested systems.

To limit the bandwidth, we assign a logical core to a Class of Service, iterating through the supported range of memory bandwidth limits. For each limit, we run the Flush+Reload loop with n addresses, just like an n-sided Rowhammer loop. We measure the performance and mitigation potential, *i.e.*, if the number of iterations the Flush+Reload loop achieves is equal to or greater than the hammer count required for a Rowhammer attack.[3] We run the loop for 10^6 iterations measuring the execution time via `rdtsc`. We compute the average iterations per refresh interval and repeat this measurement 10 times, averaging the results. We focus on Flush+Reload loops with 9 and 19 addresses, corresponding to the upper and lower bounds for recent Rowhammer attacks [7], [13], [14].

Figure 6 shows the results of our 4 test systems. For low memory bandwidths, we observe an approximately linear increase in the memory accesses with increasing memory bandwidth. Once the memory bandwidth reaches the maximum capacity required by the attack loop, the number of iterations remains constant for further increases in bandwidth.

On the consumer-level A1, an unlimited 9-address Flush+Reload loop achieves 98.2K (n=10, σ=26.8) iterations per 64 ms. The A2 server demonstrates a significantly higher number of 129.8K (n=10, σ=815) unconstrained 9-address iterations. With memory bandwidth limits, both devices demonstrate similar behavior: The tightest documented limit of 128 MiB/s achieves 9.1K (n=10, σ=0.9) iterations on the A1 and 8.3K (n=10, σ=3.3) iterations on the A2 in the same time. This restriction to 9.2 % (A1) and 6.4 % (A2) of the unlim-

[2]Note that this is the same reasoning used for single-sided hammering [2], [9] which also just exercised completely random memory locations.

[3]With more row conflicts in more targeted Rowhammer attacks, the hammer count decreases as row conflicts consume more time. Hence, in terms of security, we can treat the achieved iterations as an upper bound of the achieved hammer count on that system.

TABLE I
TEST SYSTEMS AND THEIR DRAM CONFIGURATION.

Abbreviation	Processor	OS	DRAM Vendor	DRAM Speed	Configuration	Channels	Ranks	Total Banks	ECC	MBA/L3BE
A1	AMD Ryzen 7700X	Ubuntu 22.04	Kingston	DDR5 4800 MT/s 62.7 GiB/s	2 × 16GB	2	1	64	✗	✓
A2	AMD Epyc 8024P	Ubuntu 22.04			3 × 16GB	3	1	96	✓	✓
I1	Intel Xeon 4514Y	Ubuntu 24.04			8 × 16GB	8	1	256	✓	✓
I2	Intel Xeon 4410T	Ubuntu 22.04			4 × 16GB	4	1	128	✓	✓

TABLE II
MAXIMUM FLUSH+RELOAD ITERATIONS, WITHOUT LIMIT AND WITH LOWEST MEMORY BANDWIDTH LIMIT. N = 10.

Abbreviation	Processor	9 addresses, no limit	9 addresses, limited	19 addresses, no limit	19 addresses, limited
A1	AMD Ryzen 7700X	98.2K (σ=26.8)	9.1K (σ= 0.9)	51.8K (σ=50.0)	4.3K (σ= 0.3)
A2	AMD Epyc 8024P	129.8K (σ=815)	8.3K (σ= 3.3)	81.9K (σ= 9.7)	3.6K (σ= 1.1)
I1	Intel Xeon 4514Y	199.4K (σ=287)	69.7K (σ=11.6)	140.0K (σ=40.7)	32.9K (σ=16.3)
I2	Intel Xeon 4410T	131.1K (σ= 2.8)	61.8K (σ=15.1)	77.3K (σ= 3.3)	29.8K (σ= 0.9)

ited Flush+Reload iterations is sufficient to mitigate 9-sided Rowhammer attacks on 9 out of the 10 tested modules [18].

Similarly, for the 19-address unlimited Flush+Reload, we observe significantly different numbers on the two machines: 51.8K (n=10, σ=50) (A1) and 81.9K (n=10, σ=9.7) (A2) iterations per 64 ms. With the 128 MiB/s restriction, we can restrict this down to 4.3K (n=10, σ=0.3) (A1) and 3.6K (n=10, σ=1.1) (A2) iterations, corresponding to 8.3 % (A1) and 4.4 % (A2), respectively. This memory bandwidth limit is low enough to mitigate 19-sided Rowhammer attacks on all 10 tested modules [18]. Consequently, security-wise, we consider a Memory Band-Aid proof-of-concept implementation on commodity AMD systems feasible.

Despite being undocumented, it is possible to set the memory bandwidth control MSR to 0 on AMD. However, we only observe lower Flush+Reload iteration counts for the 0-setting with low processor frequencies, while the behavior is comparable to the lowest documented restriction of 128 MiB/s for higher processor frequencies. Hence, we use 128 MiB/s as it is the lowest consistent memory bandwidth limit available.

Both tested Intel machines do not support limiting the memory bandwidth below 10 % of the system's memory bandwidth. For the unrestricted Flush+Reload loop with nine addresses, we measure 199.4K (n=10, σ=287.0) (I1) and 131.1K (n=10, σ=2.8) (I2) iterations per 64 ms. Despite these different baselines for the system's full memory bandwidth, surprisingly, the tightest, supposedly relative limit of 10 % reduces the number of iterations to similar levels: 69.7K (n=10, σ=11.6) iterations corresponding to 35.0 % on I1 and 61.9K (n=10, σ=15) iterations corresponding to 47.2 %.

In the 19-address case, Flush+Reload loop iterations are reduced from 140.0K (n=10, σ=40.7) to 32.9K (n=10, σ=16.3) (23.5 %) on I1. Equally, I2 still demonstrates a high number of iterations with 29.8K (n=10, σ=0.9) iterations with the tightest setting corresponding to 38.6 % of the unlimited number of 77.3K (n=10, σ=3.3) iterations. Hence, even the tightest restriction on Intel is **not** sufficient to mitigate many-sided Rowhammer attacks on commodity Intel systems today.

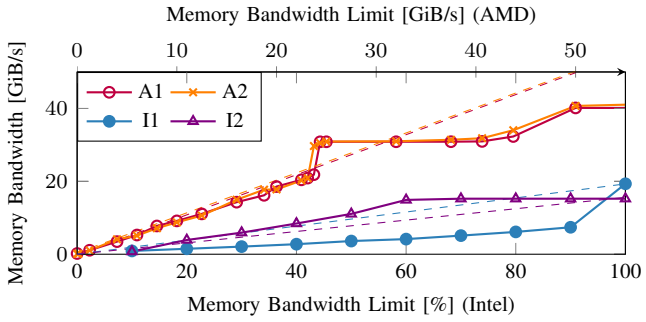


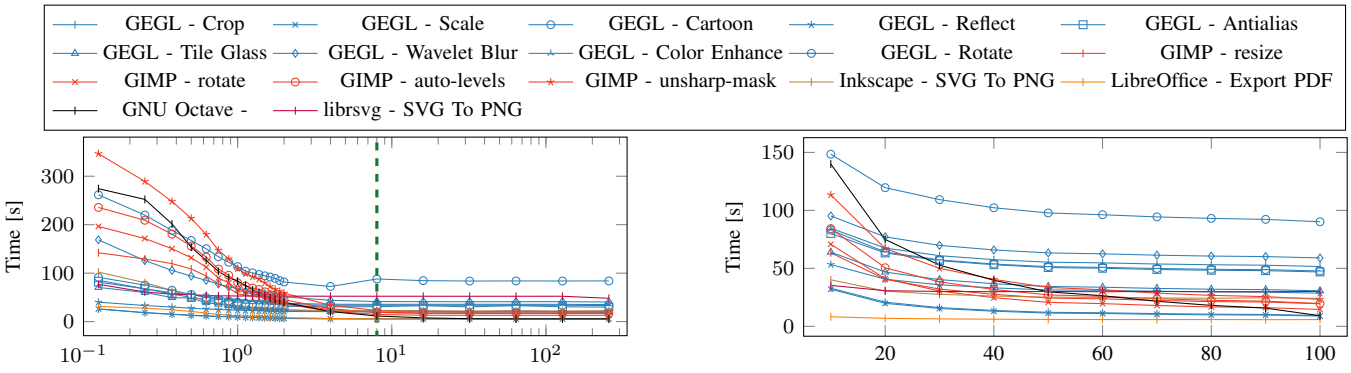
Fig. 7. Memory bandwidth of a sequential memory sweep over 1 GiB with different memory bandwidth limits on the 4 tested systems. The dashed lines show the expected values.

B. Performance with and without full bank parallelism.

DRAM functions are designed to enable software to utilize the DRAM banks statistically equally. Consequently, we can assume that the actual performance degradation caused by a full Memory Band-Aid implementation with per-bank limits is significantly lower than shown in Section VI-A, where we successfully mitigated Rowhammer attacks with a global bandwidth limit of 128 MiB/s. With secure per-bank limits, we obtain a combined bandwidth of 128 MiB/s × 32 banks = 8 GiB/s on the A1 and 128 MiB/s×96 banks = 12 GiB/s on the A2. The higher number of 128 and 256 banks on the I2 and I1 result in global limits of 16 GiB/s and 32 GiB/s, respectively.

C. Overhead in a Memory Sweep Micro-Benchmark

In this section, we evaluate the performance overhead of Memory Band-Aid in a single-core memory-sweep micro-benchmark. Unlike a tight Flush+Reload loop, a memory-sweep spreads accesses evenly across all banks and utilizes the streaming and prefetching mechanisms of the CPU. To achieve the maximum achievable per-core bandwidth, we perform multiple AVX512 loads, allowing the core to load 1 Kibit per clock cycle within a tight assembly loop on a 1 GiB huge page, ensuring no TLB misses occur. We measure the bandwidth 10 times each for different memory bandwidth limitations and present the averaged results in Figure 7.



(a). Memory Bandwidth [GiB/s], AMD Ryzen 7700X (A1). Dashed line indicates a secure Memory Band-Aid configuration.

(b) Memory Bandwidth [%], Intel Xeon 4514Y (I1)

Fig. 8. Phoronix Productivity Testsuite Results for varying memory bandwidth limitations on 2 of the evaluated systems.

Interestingly, the AMD systems have a significantly higher unrestricted per-core bandwidth of 41.1 GiB/s (n=10, σ=83 MiB/s) (A1) and 41.8 GiB/s (n=10, σ=427 MiB/s) (A2) than the Intel systems with 19 GiB/s (n=10, σ=5 MiB/s) (I1) and 15.6 GiB/s (n=10, σ=0.7 MiB/s) (I2). Both AMD systems limit the memory bandwidth close to the expected value up to 24 GiB/s, with higher limits not being enforced exactly. A full Memory Band-Aid implementation requires hard limits, however, only for significantly lower per-bank bandwidth restrictions than this. The Intel systems exhibit different behavior, even deviating significantly from the expected bandwidth values. The I1 demonstrates a sharp drop at the 90 % setting limiting the bandwidth to 39 % of the unrestricted bandwidth, corresponding to 7.4 GiB/s (n=10, σ=13 MiB/s). In contrast, the I2 does not display a significant decrease in bandwidth before reducing the limit below 60 %. Hence, I1 performs below the expected value for the majority of the settings, while I2 does not properly adhere to the bandwidth limit.

The strictest setting of 128 MiB/s on AMD is not adhered to by A1, which limits the bandwidth to 222 MiB/s (n=10, σ=39 KiB/s). A2 correctly limits it to 120 MiB/s (n=10, σ=28 KiB/s). However, in Section VI-A we demonstrated that even 222 MiB/s (n=10, σ=39 KiB/s) is suitable to mitigate many-sided attacks. With a global memory bandwidth limit, this slowdown by a factor of 190 (A1) and 349 (A2) is necessarily large to limit the number of possible DRAM accesses in a refresh interval. However, a secure 128 MiB/s limit *per-bank*, would result in a global memory bandwidth of 8 GiB/s (A1) and 12 GiB/s (A2) (see Section VI-B), corresponding to 19.5 % (A1) and 28.7 % (A2) of the unrestricted per-core bandwidth. Hence, a full Memory Band-Aid implementation experiences a slowdown by a factor of 5.1 (A1) and 3.4 (A2).

The lowest setting on Intel limits the bandwidth to 924 MiB/s (n=10, σ=2.3 MiB/s) (I1) and 756 MiB/s (n=10, σ=0.8 MiB/s) (I2), corresponding to 4.7 % (I1) and 4.8 % (I2) of the per-core bandwidth. However, this setting is insufficient to mitigate Rowhammer attacks. In contrast, with the required *per-bank* limit of 128 MiB/s, Memory Band-Aid would facilitate a global limit of 32 GiB/s (I1) and 16 GiB/s (I2), exceeding the corresponding available per-core memory bandwidth. Hence restricted workloads on I1 and I2 do *not* suffer any performance loss with a full implementation of Memory Band-Aid.

D. Performance Overhead in Phoronix Macro-Benchmarks

We use the Phoronix Productivity Testsuite to evaluate the impact of Memory Band-Aid on typical consumer workloads. First, we evaluate the performance overhead of a *secure* global limit with our proof-of-concept implementation. Second, we discuss the significantly lower overheads of a full Memory Band-Aid implementation based on the combined bandwidth provided by *per-bank* limitations (see Section VI-B).

We run all benchmarks on a single core with varying memory bandwidth restrictions: On Intel, we test all 10 available memory bandwidth limits. On AMD, we focus on lower memory bandwidth limits, as these are suitable for mitigating Rowhammer attacks, even with a global limit instead of a per-bank limit, in our proof-of-concept implementation. We sample higher values at a coarser granularity. Figure 8 and Figure 11 show the benchmark results for all tested memory bandwidth limits. Table III details the performance results for the tested bandwidth settings on each machine. The performance varies by up to 2 orders of magnitude, depending on how memory-bound or compute-bound the benchmarks are. Compute-bound benchmarks are (in the best case) not affected at all, whereas memory-bound benchmarks (in the worst case) are permanently throttled due to the limit.

Global Memory Bandwidth Limit. On the I1, the strictest bandwidth limitation of 10 % induces an average overhead of 156.6 % with a geometric mean of 60.6 s across all benchmarks compared to 23.6 s with unlimited bandwidth. On the I2 only the tightest limit induces an overhead of 118.6 %, with a geometric mean of 81.5 s across all benchmarks, compared to 37.3 s with unlimited bandwidth. However, this setting is insufficient to mitigate Rowhammer attacks. In contrast, the strictest bandwidth limit on AMD mitigates 19-sided Rowhammer on all DDR4 modules. On the A1, since

TABLE III
PHORONIX PRODUCTIVITY TESTSUITE PERFORMANCE RESULTS OF OUR PROOF-OF-CONCEPT IMPLEMENTATION.

Benchmark	Intel I1			AMD A1				
	Unlim.	10 % Per-system	Insecure PoC Overhead	Unlim.	128 MiB/s Per-system	**Secure PoC Overhead**	128 MiB/s Per-bank	**Full Impl. Overhead**
GEGL - Crop	9.4	32.9	250.6 %	5.8	26.0	**346.2 %**	5.8	**0.3 %**
GEGL - Scale	8.8	32.0	261.6 %	5.5	26.1	**377.1 %**	5.5	**0.1 %**
GEGL - Cartoon	90.2	148.4	64.6 %	84.0	261.6	**211.6 %**	87.9	**4.7 %**
GEGL - Reflect	28.6	53.3	86.5 %	22.3	40.1	**79.4 %**	22.9	**2.4 %**
GEGL - Antialias	46.9	80.1	70.9 %	29.9	81.4	**172.3 %**	30.8	**2.8 %**
GEGL - Tile Glass	30.6	64.0	109.1 %	19.2	70.5	**266.9 %**	19.8	**3.0 %**
GEGL - Wavelet Blur	59.0	95.2	61.5 %	40.4	168.8	**317.9 %**	41.3	**2.4 %**
GEGL - Color Enhance	51.5	84.6	64.1 %	35.1	85.3	**143.0 %**	35.7	**1.5 %**
GEGL - Rotate 90 Degrees	47.7	83.0	73.9 %	33.0	91.4	**177.0 %**	33.7	**2.2 %**
GIMP - resize	19.4	63.4	226.0 %	17.6	142.0	**707.9 %**	19.1	**8.4 %**
GIMP - rotate	14.5	70.7	387.4 %	13.0	196.4	**1 415.8 %**	14.7	**13.3 %**
GIMP - auto-levels	19.7	84.3	328.6 %	16.9	235.6	**1 292.6 %**	19.0	**12.5 %**
GIMP - unsharp-mask	23.2	113.4	389.4 %	20.4	346.5	**1 594.8 %**	23.0	**12.5 %**
Inkscape - SVG Files To PNG	24.2	40.1	66.1 %	22.1	102.3	**363.5 %**	22.1	**0.1 %**
LibreOffice - 20 Docs To PDF	5.8	8.2	42.0 %	5.1	31.4	**513.4 %**	5.5	**7.9 %**
GNU Octave Benchmark -	9.1	139.9	1 433.2 %	5.6	274.3	**4 787.9 %**	11.6	**106.8 %**
librsvg - SVG Files To PNG	30.1	35.3	17.3 %	48.1	76.9	**59.8 %**	52.3	**8.7 %**
Mean	23.6	60.6	156.6 %	18.6	98.9	**431.4 %**	20.4	**9.4 %**

We tested our proof-of-concept on the Intel Xeon 4514Y (I1) and AMD Ryzen 7700X (A1) without bandwidth limits, at the most restrictive bandwidth limit, and the total memory system bandwidth of a full Memory Band-Aid implementation with per-bank restrictions. Lower is better.

enforcing a smaller bandwidth, this secure limit induces a significantly higher overhead of 431.4 % on average, with a geometric mean of 98.9 s across all benchmarks, compared to 18.6 s with unlimited bandwidth. The A2 demonstrates worse performance overall with an unlimited geometric mean of 30.7 s, but slighltly less relative overhead of 388.8 % with a geometric mean of 150.2 s under the 128 MiB/s limitation.

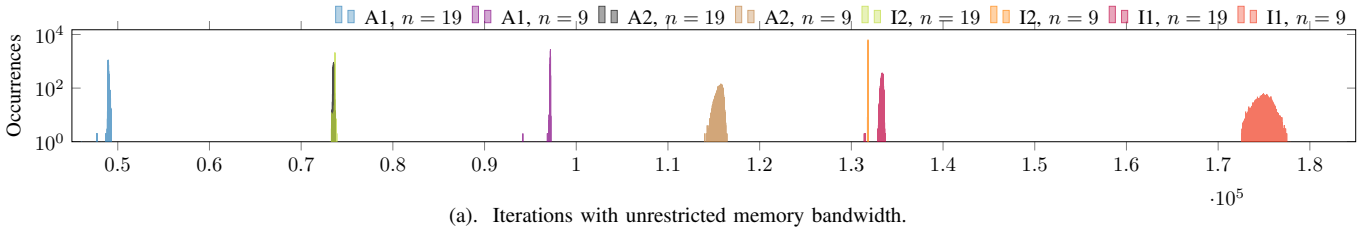
Our results show that typical consumer workloads are already not as significantly affected by global bandwidth limits like worst-case memory-intense micro-benchmarks, as they do not exercise memory as intensely. Furthermore, our micro-benchmarks deliberately bypassed the cache or exceeded its capacity significantly. In macro-benchmarks, caching contributes to maintaining a reasonable performance level.

Per-Bank Memory Bandwidth Limit. Due to bank parallelism, a full Memory Band-Aid implementation with per-bank limits can achieve 2 orders of magnitude smaller performance overheads. A secure 128 MiB/s per-bank limit corresponds to a general memory bandwidth of 8 GiB/s to 32 GiB/s on the evaluated systems due to the different number of banks (see Section VI-B). Since this combined memory bandwidth exceeds the achievable per-core limit on both tested Intel machines (see Section VI-C), restricted workloads are **not** throttled by Memory Band-Aid due to the high bank parallelism. With the significantly smaller number of 64 banks on the A1, a 8 GiB/s memory bandwidth limit induces an overhead of 9.4 % on average with a geometric mean of 20.4 s across all benchmarks compared to 18.6 s with unlimited bandwidth. The increased number of 96 banks on the A2 results in a reduced overhead of 6.7 % on average with a geometric mean of 32.8 s compared to the unlimited mean of 30.7 s. Hence, the number of banks plays a crucial role in the performance overhead of a full Memory Band-Aid implementation with high bank parallelism allowing for negligible performance overheads for typical consumer workloads. The performance overhead observed in these realistic benchmarks, even for the global bandwidth limits implemented by current-generation Intel and AMD processors, is acceptable. Furthermore, the performance overheads with our Memory Band-Aid prototype using current AMD L3BE or Intel MBA implementations are significantly worse than with a full Memory Band-Aid implementation and a per-bank implementation of the limit. This shows the practicality of Memory Band-Aid with per-bank limits as a defense in depth, with overheads of a full implementation in the range of 0 % to 9.4 %.

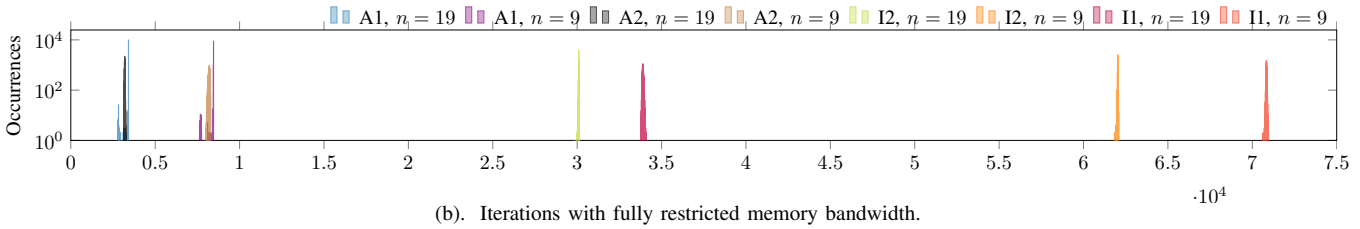
E. Consistency of Memory Bandwidth Limits

Some quality of service features only provide approximate guarantees. Hence, we evaluate how much the Flush+Reload iterations vary across multiple 64 ms intervals to determine whether this approach is a viable Rowhammer mitigation. For this purpose, we again run Flush+Reload loops with 9 and 19 addresses on our 4 test systems with unlimited memory bandwidth as well as the tightest available limit. Figure 9 shows the histogram of the iterations we achieved in 10^4 64 ms long measurements for each setting and tested machine.

All distributions are concentrated in one large spike. For an unlimited 9-address loop, we observe wider connected distributions on 2 of the tested systems with a standard deviation of 883 Flush+Reload iterations (I1) and 350 iterations (A2). All other distributions demonstrate small standard deviations below 160 Flush+Reload iterations. We observe a small peak of outliers below when accessing 9 or 19 addresses on the A1 and 19 addresses on the I1. With a secure bandwidth limit, the A1 demonstrates a considerable number of outliers below of the major peak. The distributions of the A2 for the same settings fall in between the two peaks of the A1, demonstrating different precision in targeting the limit.



(a). Iterations with unrestricted memory bandwidth.



(b). Iterations with fully restricted memory bandwidth.

Fig. 9. Distribution of Flush+Reload iterations per 64 ms refresh interval on our 4 test systems. We access $n = 9$ or $n = 19$ addresses in the Flush+Reload loop, like corresponding Rowhammer attacks.

Most importantly, we did not see any outliers above the limit on any machine. This shows that both Intel and AMD indeed enforce bandwidth limitations as an upper bound for the respective tightest bandwidth limits. However, we need to consider a security margin to account for the width of the distribution when choosing appropriate bandwidth limits. Furthermore, the observed number of outliers *below* on some machines highlights the need for vendors to improve their CPUs, ensuring they do not significantly underperform below configured limits when implementing Memory Band-Aid.

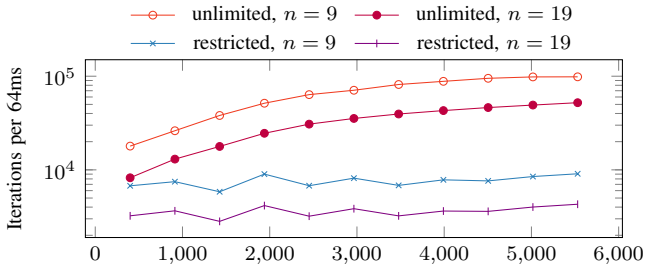
F. Impact of Processor Frequency

Since the memory bandwidth limit is implemented in the processor, we investigated whether the processor's frequency affects the accuracy of the bandwidth limit, particularly given features such as frequency scaling and turbo boost. For this purpose, we run the Flush+Reload loop either restricted to the tightest supported memory bandwidth or with no bandwidth limitation. Then, we vary the processor frequency of all processor cores in equally sized steps between the available hardware limits. We execute a Flush+Reload hammering loop for 10^6 iterations for each processor frequency, and measure the execution time via the `rdtsc` instruction. Finally, we compute the average number of Flush+Reload iterations per refresh interval. We repeat this measurement 10 times for each bandwidth and frequency pair and average the results.

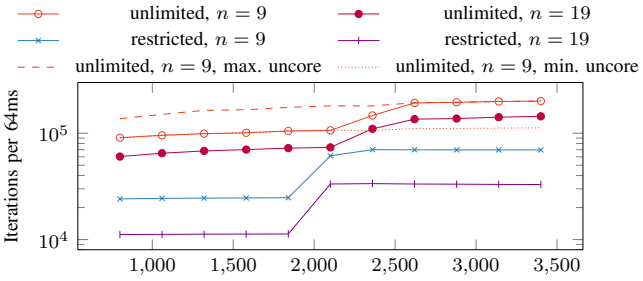
Figure 10 and Figure 12 show the results for 9- and 19-address Flush+Reload loops on 2 test systems each. On the A1, the behavior is consistent in all cases: With unrestricted bandwidth, low processor frequencies are a bottleneck for the Flush+Reload iterations. The iterations per refresh interval grow with increasing processor frequency until the system's memory bandwidth capacity is reached. At that point, the DRAM becomes a performance bottleneck, *i.e.*, further processor frequency increases do not significantly increase the number of Flush+Reload iterations. With low memory bandwidth limits, the available memory bandwidth is already a bottleneck at low processor frequencies. Therefore, increasing the processor frequency does not systematically increase the number of iterations per 64 ms. However, we observe fluctuations in the number of loop iterations per refresh interval, which vary by several thousand accesses. While A2 offers only 3 frequency settings, we still observe the same trend as on the A1: Without a memory bandwidth limit, increased CPU frequency results in a significantly increased number of Flush+Reload iterations. With a tight memory bandwidth limit, the number of iterations only fluctuates slightly.

On the I1, we observe a different effect: With unlimited memory bandwidth, we observe a shallow increase in throughput throughout the frequency range, of about 3.3 % on average (slightly higher at lower frequencies) per 260 MiHz. We conclude that the increase in the observed throughput is not primarily dependent on the core frequency. Instead, we observe a jump at 2 GHz that roughly doubles the throughput. We discovered that this effect is due to uncore frequency scaling, *i.e.*, the bottleneck on the Intel I1 is the memory controller. The frequency at which the memory controller runs, in turn, is influenced by the processor frequency of the cores [98]. We verify uncore frequency scaling as the root cause of the observations by setting the uncore frequency to the lowest and highest available settings, 800 MHz and 2.4 GHz, respectively, and repeating the measurement without a memory bandwidth limit. With fixed uncore frequency, the number of Flush+Reload iterations behaves similarly to AMD, with slight increases with the processor core frequency. On the I2, we do not observe significant changes with varying CPU frequency. In particular, we do not observe a sudden jump due to uncore frequency scaling. This can be attributed to the fact that the system is already running at a high uncore frequency even when the system load is low. Therefore, increasing the CPU frequency and, therefore, uncore frequency further does not lead to a significant increase in Flush+Reload iterations.

From our experiments, we conclude that despite variations in the number of Flush+Reload iterations and, thus, maximum



(a). CPU Frequency [MHz], AMD A1



(b). CPU Frequency [MHz], Intel I1. Dashed and dotted lines with minimum and maximum uncore frequency.

Fig. 10. Impact of the processor frequency on Flush+Reload loops with $n = 9$ and $n = 19$ addresses. The plot shows the iterations achieved per refresh interval on the AMD A1 and Intel I1.

hammer counts possible, Memory Band-Aid can reliably limit the number of row activations within a refresh interval regardless of the processor frequencies. We configure the limits of Memory Band-Aid based on the measurements with the maximum uncore frequency, hence uncore frequency scaling cannot be exploited to circumvent the limit.

VII. Discussion and Related Work

Memory Band-Aid is an effective defense in depth against Rowhammer. With minimal hardware changes and even limited support on AMD commodity systems today, it is the next line of defense necessary to tackle the Rowhammer problem.

Our evaluation showed that several limitations on current hardware limit our proof-of-concept Memory Band-Aid implementation. The flexible configuration of memory bandwidth limits for specific workloads already allows for restricting any performance penalty to untrusted workloads alone. However, coarse bandwidth granularities makes it impossible to achieve the necessary slowdowns on the investigated Intel systems when assuming the lowest ever reported minimum hammer count to induce bit flips. The lack of per-bank limits induces a need for stricter global limits, impacting the performance overhead of Memory Band-Aid on commodity hardware. Still, we see an acceptable overhead for most workloads and only intense micro-benchmarks, e.g., a loop of Flush+Reload, see a significant slowdown. We conclude that deployment on current AMD hardware via a kernel patch is already a feasible (but costly) Rowhammer mitigation today.

While previous work [3], [99] explored restricting accesses to memory as a Rowhammer mitigation, we are the first to investigate per-bank bandwidth limiting as a practical Rowhammer mitigation with acceptable overheads. Closest to our work is a Linux Kernel Mailing List proposal [99] to use performance counters to trigger interrupts every 10 000 cache misses and **busy wait** the thread when cache misses exceed 97 600. This not only limits memory but also processor throughput and was already deemed too restrictive and slow for real-world workloads to be a plausible Rowhammer mitigation at the time. Furthermore, not all performance counters are precise [100], and they can incur performance overheads [101]. Additionally, our numbers show that they would have to increase the interrupt frequency and lower the miss threshold to busy-wait the processor by one order of magnitude each. We address several limitations in this work: Memory Band-Aid directly utilizes per-thread bandwidth limits, thereby achieving acceptable overheads with our current proof-of-concept, which only throttles memory throughput and untrusted workloads. Furthermore, our proposal of per-bank memory bandwidth limits enables a 2-order-of-magnitude increase in performance.

Blockhammer [3] detects per-thread row activations that are likely suspicious, temporarily blocklisting accesses to these rows. It then determines a quota for in-flight memory requests of this thread to **shared caches or memory** (see [3]-3.2.2), effectively delaying every single memory operation. Similar to MBA 1.0 on older Intel CPUs, which uses delays instead of actual rate-limiting [17], this throttles workloads even if they do not reach the memory bandwidth limit within a time slice. In contrast, our mitigation only limits memory accesses and only when the memory bandwidth limit is reached, not earlier and not probabilistically by utilizing quality-of-service features that are introduced into processors independent of Rowhammer, e.g., MBA/L3BE, which are already sufficient for first prototypes. A clear advantage of Memory Band-Aid over hardware-centric mitigations is the possibility to quickly roll out necessary updates, due to changes in the attack surface, that update the configuration of Memory Band-Aid to secure default parameters, accounting for new information on minimum hammer counts. Furthermore, the environment, e.g., temperature, also affect Rowhammer thresholds [18], as may aging of DRAM chips. Acting on the operating system level, Memory Band-Aid can know these parameters and consider this for slight adjustments in the memory bandwidth limits.

Vendors introduced bandwidth limits as a quality-of-service feature, not for Rowhammer, continuously improving performance and granularity [17]. We envision future systems where Memory Band-Aid becomes a hardware-agnostic framework relying merely on already present features. The need for tighter bandwidth, per-bank, and per-logical-core limits, to achieve complete security and performance benefits, contributes to understanding what vendors should change to create a plat-

form for a hardware-agnostic framework. With these changes, Memory Band-Aid is a practical defense in depth.

VIII. Conclusion

Memory Band-Aid is a defense-in-depth against Rowhammer, using memory bandwidth limits to enforce that no DRAM bank reaches the minimum hammer count necessary to induce a bit flip. Hence, even when deployed hardware mitigations fail, Memory Band-Aid can still reliably mitigate Rowhammer attacks. We observe that overheads on current systems, with global limits, may be prohibitively high but future system with per-bank bandwidth limits will remain in the range of $0\,\%$ to $9.4\,\%$ overhead in realistic use cases. We conclude that vendors still need to implement these minimal hardware changes and that with these changes, Memory Band-Aid is a practical defense-in-depth that should be deployed in practice as a building block to mitigate Rowhammer sustainably.

Acknowledgment

We thank the SAFARI research group, and specifically Ataberk Olgun and Onur Mutlu, for providing the minimum hammer count data that we based our analysis on. This research is supported in part by the European Research Council (ERC project FSSec 101076409), the Austrian Science Fund (FWF SFB project SPyCoDe 10.55776/F85 and FWF project NeRAM 10.55776/I6054), the Deutsche Forschungsgemeinschaft (grant no. 503876675), and the European Union (grant no. ROF-SG20-3066-3-2-2). Additional funding was provided by generous gifts from Red Hat, Google and Intel. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

References

[1] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. 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 *ISCA*, 2014.

[2] M. Seaborn, "Exploiting the DRAM rowhammer bug to gain kernel privileges," 2015. [Online]. Available: http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

[3] A. G. Yaglikci, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi, S. Ghose, and O. Mutlu, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," in *HPCA*, 2021.

[4] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ISCA*, 2020.

[5] L. Orosa, A. G. Yaglikci, H. Luo, A. Olgun, J. Park, H. Hassan, M. Patel, J. S. Kim, and O. Mutlu, "A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses," in *MICRO*, 2021.

[6] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-Double: Hammering From the Next Row Over," in *USENIX Security*, 2022.

[7] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "BLACKSMITH: Rowhammering in the Frequency Domain," in *S&P*, 2021.

[8] H. Luo, A. Olgun, A. G. Yağlıkçı, Y. C. Tuğrul, S. Rhyner, M. B. Cavlak, J. Lindegger, M. Sadrosadati, and O. Mutlu, "RowPress: Amplifying Read Disturbance in Modern DRAM Chips," in *ISCA*, 2023.

[9] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," in *S&P*, 2018.

[10] F. de Ridder, P. Jattke, and K. Razavi, "Posthammer: Pervasive Browser-based Rowhammer Attacks with Postponed Refresh Commands," in *USENIX Security*, 2025.

[11] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *S&P*, 2019.

[12] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *S&P*, 2020.

[13] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript," in *USENIX Security*, 2021.

[14] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms," in *USENIX Security*, 2024.

[15] AMD, "AMD64 Architecture Programmer's Manual," 2024.

[16] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3 (3A, 3B & 3C): System Programming Guide," 2024.

[17] P. Sohal, M. Bechtel, R. Mancuso, H. Yun, and O. Krieger, "A Closer Look at Intel Resource Director Technology (RDT)," in *Real-Time Networks and Systems (RTNS)*, 2022.

[18] A. Olgun, F. N. Bostanci, I. E. Yuksel, O. Canpolat, H. Luo, G. F. Oliveira, A. G. Yaglikci, M. Patel, and O. Mutlu, "Variable Read Disturbance: An Experimental Analysis of Temporal Variation in DRAM Read Disturbance," in *HPCA*, 2025.

[19] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector," in *S&P*, 2016.

[20] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *USENIX Security*, 2016.

[21] Y. Xiao, X. Zhang, Y. Zhang, and R. Teodorescu, "One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation," in *USENIX Security*, 2016.

[22] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," in *DIMVA*, 2016.

[23] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating software mitigations against rowhammer: a surgical precision hammer," in *RAID*, 2018.

[24] A. Tatar, R. Krishnan, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer Attacks over the Network and Defenses," in *USENIX ATC*, 2018.

[25] S. Ji, Y. Ko, S. Oh, and J. Kim, "Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks," in *AsiaCCS*, 2019.

[26] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading Bits in Memory Without Accessing Them," in *S&P*, 2020.

[27] Y. Tobah, A. Kwong, I. Kang, D. Genkin, and K. G. Shin, "SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks," in *S&P*, 2022.

[28] I. Kang, W. Wang, J. Kim, S. van Schaik, Y. Tobah, D. Genkin, A. Kwong, and Y. Yarom, "Sledgehammer: Amplifying rowhammer via bank-level parallelism," in *USENIX Security*, 2024.

[29] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-Bomb: Locking Down the Processor via Rowhammer Attack," in *SysTEX*, 2017.

[30] M. T. Aga, Z. B. Aweke, and T. Austin, "When good protections go bad: Exploiting anti-DoS measures to accelerate Rowhammer attacks," in *HOST*, 2017.

[31] M. Lipp, M. T. Aga, M. Schwarz, D. Gruss, C. Maurice, L. Raab, and L. Lamster, "Nethammer: Inducing Rowhammer Faults through Network Requests," in *SILM Workshop*, 2020.

[32] L. Orosa, U. Rührmair, A. G. Yaglikci, H. Luo, A. Olgun, P. Jattke, M. Patel, J. Kim, K. Razavi, and O. Mutlu, "SpyHammer: Using RowHammer to Remotely Spy on Temperature," *arXiv:2210.04084*, 2022.

[33] L. Gerlach, F. Thomas, R. Pietsch, and M. Schwarz, "A Rowhammer Reproduction Study Using the Blacksmith Fuzzer," in *European Symposium on Research in Computer Security*, 2023.

[34] J. Juffinger, S. R. Neela, M. Heckel, L. Schwarz, F. Adamsky, and D. Gruss, "Presshammer: Rowhammer and Rowpress without Physical Address Information," in *DIMVA*, 2024.

[35] M. Marazzi and K. Razavi, "RISC-H: Rowhammer Attacks on RISC-V," in *DRAMSec Workshop*, 2024.

[36] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS*, 2016.

[37] Z. Zhang, Z. Zhan, D. Balasubramanian, X. Koutsoukos, and G. Karsai, "Triggering Rowhammer Hardware Faults on ARM: A Revisit," in *ASHES Workshop*, 2018.

[38] Y. Yarom and K. Falkner, "Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX Security*, 2014.

[39] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," in *S&P*, 2018.

[40] Apple Inc., "About the Security Content of Mac EFI Security Update 2015-001," https://support.apple.com/en-us/HT204934, 2015.

[41] Lenovo, "Row Hammer Privilege Escalation," 2015. [Online]. Available: https://support.lenovo.com/us/en/product_security/row_hammer

[42] JEDEC, "JESD235B: High Bandwidth Memory DRAM (HBM1, HBM2) Standard," 2020.

[43] JEDEC Solid State Technology Association, "DDR4 SDRAM Standard," 2021. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-4a

[44] S. Hong, D. Kim, J. Lee, R. Oh, C. Yoo, S. Hwang, and J. Lee, "DSAC: Low-Cost Rowhammer Mitigation Using In-DRAM Stochastic and Approximate Counting Algorithm," *arXiv preprint*, 2023.

[45] JEDEC Solid State Technology Association, "DDR5 SDRAM Standard," 2024. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-5c01

[46] O. Canpolat, A. G. Yağlıkçı, G. F. Oliveira, A. Olgun, O. Ergin, and O. Mutlu, "Understanding the Security Benefits and Overheads of Emerging Industry Solutions to DRAM Read Disturbance," *DRAMSec*, 2024.

[47] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime," in *MICRO*, 2022.

[48] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, "Graphene: Strong yet Lightweight Row Hammer Protection," in *MICRO*, 2020.

[49] M. J. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. J. Ham, J. W. Lee, and J. H. Ahn, "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh," in *HPCA*, 2022.

[50] J. M. You and J.-S. Yang, "MRLoc: Mitigating Row-hammering based on memory Locality," in *Design Automation Conference (DAC)*, 2019.

[51] I. Kang, E. Lee, and J. H. Ahn, "CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention," *IEEE Access*, 2020.

[52] A. G. Yauglikcci, A. Olgun, M. Patel, H. Luo, H. Hassan, L. Orosa, O. Ergin, and O. Mutlu, "HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips," in *MICRO*, 2022.

[53] H. Hassan, A. Olgun, A. G. Yağlıkçı, H. Luo, and O. Mutlu, "Self-Managing DRAM: A Low-Cost Framework for Enabling Autonomous and Efficient DRAM Maintenance Operations," in *MICRO*, 2024.

[54] R. Zhou, S. Tabrizchi, A. Roohi, and S. Angizi, "LT-PIM: An LUT-Based Processing-in-DRAM Architecture with RowHammer Self-Tracking," *CAL*, 2022.

[55] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM Stronger Against Row Hammering," in *Design Automation Conference (DAC)*, 2017.

[56] G. Saileshwar, B. Wang, M. Qureshi, and P. J. Nair, "Randomized row-swap: mitigating Row Hammer by breaking spatial correlation between aggressor and victim rows," in *ASPLOS*, 2022.

[57] S.-W. Ryu, K. Min, J. Shin, H. Kwon, D. Nam, T. Oh, T.-S. Jang, M. Yoo, Y. Kim, and S. Hong, "Overcoming the Reliability Limitation in the Ultimately Scaled DRAM Using Silicon Migration Technique by Hydrogen Annealing," in *International Electron Devices Meeting (IEDM)*, 2017.

[58] S. Saroiu, A. Wolman, and L. Cojocar, "The Price of Secrecy: How Hiding Internal DRAM Topologies Hurts Rowhammer Defenses," in *IEEE IRPS*, 2022.

[59] J. Han, J. Kim, D. Beery, K. D. Bozdag, P. Cuevas, A. Levi, I. Tain, K. Tran, A. J. Walker, S. V. Palayam *et al.*, "Surround Gate Transistor With Epitaxially Grown Si Pillar and Simulation Study on Soft Error and Rowhammer Tolerance for DRAM," *TED*, 2021.

[60] S. Saroiu and A. Wolman, "How to Configure Row-Sampling-Based Rowhammer Defenses," DRAMSec, 2022.

[61] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: preventing row-hammering by exploiting time window counters," in *ISCA*, 2019.

[62] Z. Greenfield and T. Levy, "Throttling support for row-hammer counters," 2014, US Patent 9251885.

[63] F. Devaux and R. Ayrignac, "Method and Circuit for Protecting a DRAM Memory Device from the Row Hammer Effect," U.S. Patent 10,885,966, 2021.

[64] G.-H. Lee, S. Na, I. Byun, D. Min, and J. Kim, "CryoGuard: A Near Refresh-Free Robust DRAM Design for Cryogenic Computing," in *ISCA*, 2021.

[65] B. K. Joardar, T. K. Bletsch, and K. Chakrabarty, "Learning to Mitigate RowHammer Attacks," in *DATE*, 2022.

[66] ——, "Machine Learning-Based Rowhammer Mitigation," *TCAD*, 2022.

[67] M. Qureshi, A. Rohan, G. Saileshwar, and P. J. Nair, "Hydra: Enabling Low-Overhead Mitigation of Row-Hammer at Ultra-Low Thresholds via Hybrid Tracking," in *ISCA*, 2022.

[68] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Mitigating Wordline Crosstalk using Adaptive Trees of Counters," in *ISCA*, 2018.

[69] A. Naseredini, M. Berger, M. Sammartino, and S. Xiong, "ALARM: Active LeArning of Rowhammer Mitigations," https://users.sussex.ac.uk/~mfb21/rh-draft.pdf, 2022.

[70] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural Support for Mitigating Row Hammering in DRAM Memories," *CAL*, 2015.

[71] J. Woo, G. Saileshwar, and P. J. Nair, "Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems," in *HPCA*, 2023.

[72] C. Yang, C. K. Wei, Y. J. Chang, T. C. Wu, H. P. Chen, and C. S. Lai, "Suppression of RowHammer Effect by Doping Profile Modification in Saddle-Fin Array Devices for Sub-30-nm DRAM Technology," *TDMR*, 2016.

[73] H. Gomez, A. Amaya, and E. Roa, "DRAM Row-Hammer Attack Reduction Using Dummy Cells," in *NORCAS*, 2016.

[74] F. N. Bostanci, I. E. Yüksel, A. Olgun, K. Kanellopoulos, Y. C. Tuğrul, A. G. Yağliçi, M. Sadrosadati, and O. Mutlu, "CoMeT: Count-Min-Sketch-Based Row Tracking to Mitigate RowHammer at Low Cost," in *HPCA*, 2024.

[75] A. Olgun, Y. C. Tugrul, N. Bostanci, I. E. Yuksel, H. Luo, S. Rhyner, A. G. Yaglikci, G. F. Oliveira, and O. Mutlu, "ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation," in *USENIX Security*, 2024.

[76] A. G. Yaglikci, Y. C. Tuğrul, G. F. De Oliviera, I. E. Yüksel, A. Olgun, H. Luo, and O. Mutlu, "Spatial Variation-Aware Read Disturbance Defenses: Experimental Analysis of Real DRAM Chips and Implications on Future Solutions," in *HPCA*, 2024.

[77] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "Protrr: Principled yet optimal in-dram target row refresh," in *S&P*, 2022.

[78] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "Rega: Scalable rowhammer mitigation with refresh-generating activations," in *S&P*, 2023.

[79] T. Bennett, S. Saroiu, A. Wolman, and L. Cojocar, "Panopticon: A Complete In-DRAM Rowhammer Mitigation," DRAMSec, 2021.

[80] H. Hassan, M. Patel, J. S. Kim, A. G. Yağlıkçı, N. Vijaykumar, N. Mansouri Ghiasi, S. Ghose, and O. Mutlu, "CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability," in *ISCA*, 2019.

[81] O. Mutlu, A. Olgun, and A. G. Yağlıkcı, "Fundamentally Understanding and Solving RowHammer," in *Asia and South Pacific Design Automation Conference*, 2023.

[82] Y. Wang, Y. Liu, P. Wu, and Z. Zhang, "Discreet-PARA: Rowhammer Defense with Low Cost and High Efficiency," in *ICCD*, 2021.

[83] O. Canpolat, A. G. Yağlıkçı, G. F. Oliveira, A. Olgun, N. Bostancı, I. E. Yuksel, H. Luo, O. Ergin, and O. Mutlu, "Chronus: Understanding and Securing the Cutting-Edge Industry Solutions to DRAM Read Disturbance," in *HPCA*, 2025.

[84] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "ANVIL: Software-based protection against next-generation Rowhammer attacks," *ACM SIGPLAN Notices*, 2016.

[85] F. Brasser, L. Davi, D. Gens, C. Liebchen, and A.-R. Sadeghi, "CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory," in *USENIX Security*, 2017.

[86] V. van der Veen, M. Lindorfer, Y. Fratantonio, H. P. Pillai, G. Vigna, C. Kruegel, H. Bos, and K. Razavi, "GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM," in *DIMVA*, 2018.

[87] C. Bock, F. Brasser, D. Gens, C. Liebchen, and A.-R. Sadeghi, "RIP-RH: Preventing Rowhammer-based Inter-Process Attacks," in *Asia CCS*, 2019.
[88] R. K. Konoth, M. Oliverio, A. Tatar, D. Andriesse, H. Bos, C. Giuffrida, and K. Razavi, "ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks," in *USENIX OSDI*, 2018.
[89] Z. Zhang, Y. Cheng, D. Liu, S. Nepal, Z. Wang, and Y. Yarom, "PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses," in *MICRO*, 2020.
[90] A. Di Dio, K. Koning, H. Bos, and C. Giuffrida, "Copy-on-flip: Hardening ecc memory against rowhammer attacks." in *NDSS*, 2023.
[91] J. Juffinger, L. Lamster, A. Kogler, M. Eichlseder, M. Lipp, and D. Gruss, "CSI: Rowhammer - Cryptographic Security and Integrity against Rowhammer," in *S&P*, 2023.
[92] Intel, ""Noisy Neighbors" Problem in Kubernetes," 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/noisy-neighbors-problem-in-kubernetes.html
[93] Intel, "Intel Resource Director Technology (Intel RDT) Architecture Specification," 2025.
[94] M. Wi, J. Park, S. Ko, M. J. Kim, N. Sung Kim, E. Lee, and J. H. Ahn, "SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling," in *IEEE HPCA*, 2023.
[95] W. He, Z. Zhang, Y. Cheng, W. Wang, W. Song, Y. Gao, Q. Zhang, K. Li, D. Liu, and S. Nepal, "WhistleBlower: A System-Level Empirical Study on RowHammer," *IEEE Transactions on Computers*, 2023.
[96] S. Baek, M. Wi, S. Park, H. Nam, M. J. Kim, N. S. Kim, and J. H. Ahn, "Marionette: A RowHammer Attack via Row Coupling," in *ASPLOS*, 2025.
[97] C. Maurice, N. Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters," in *RAID*, 2015.
[98] Y. Guo, D. Cao, X. Xin, Y. Zhang, and J. Yang, "Uncore Encore: Covert Channels Exploiting Uncore Frequency Scaling," in *MICRO*, 2023.
[99] J. Corbet, "Defending against Rowhammer in the kernel," 10 2016. [Online]. Available: https://lwn.net/Articles/704920/
[100] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "SoK: The challenges, pitfalls, and perils of using hardware performance counters for security," in *S&P*, 2019.
[101] M. Schwarzl, P. Borrello, A. Kogler, K. Varda, T. Schuster, D. Gruss, and M. Schwarz, "Robust and Scalable Process Isolation against Spectre in the Cloud (Extended Version)," 2022. [Online]. Available: https://martinschwarzl.at/media/files/robust_extended.pdf

APPENDIX

A. Additional Systems Data

In this section, we provide additional experimental data for AMD Epyc 8024P (A2) and Intel Xeon 4410T (I2).

a) Performance Overhead in Phoronix Macro-Benchmarks: Figure 11 shows the results of the Phoronix benchmarks for different memory bandwidth configurations on the 2 additional systems A2 and I2.

b) Impact of Processor Frequency: Figure 12 shows the variation of Flush+Reload iterations per refresh interval with varying CPU frequencies on the A2 and the I2.

B. Browser Evaluation

To evaluate the overhead of Memory Band-Aid on browser applications, we restricted a logical core to varying memory bandwidth limits and measured the duration of launching Google Chrome headless and accessing `https://google.com`. On our system, this simple test already involves more than 10 processes and many threads and thus, a significant amount of inter-process communication. Figure 13 shows the averaged results of 20 measurements on the four test systems. The strictest, yet insecure global setting on Intel demonstrates a slowdown by a factor of 1.5 (I1) and 1.75 (I2). On AMD, we measure a slowdown by a factor of 7.4 (A1) and 8.9 (A2) for the secure global limit of 128 MiB/s. On all 4 machines, a secure per-bank limit of 128 MiB/s as employed in a full Memory Band-Aid implementation does not induce any statistically significant overhead. If not applied generally to all user processes, *i.e.*, if not all user processes are untrusted, a potential application of Memory Band-Aid to browsers can limit specifically the sandboxed processes, while leaving other browser processes unaffected. While this may yield a more favorable security-performance trade-off, it also opens attack surface to sandbox escapes.

C. Security Formalization

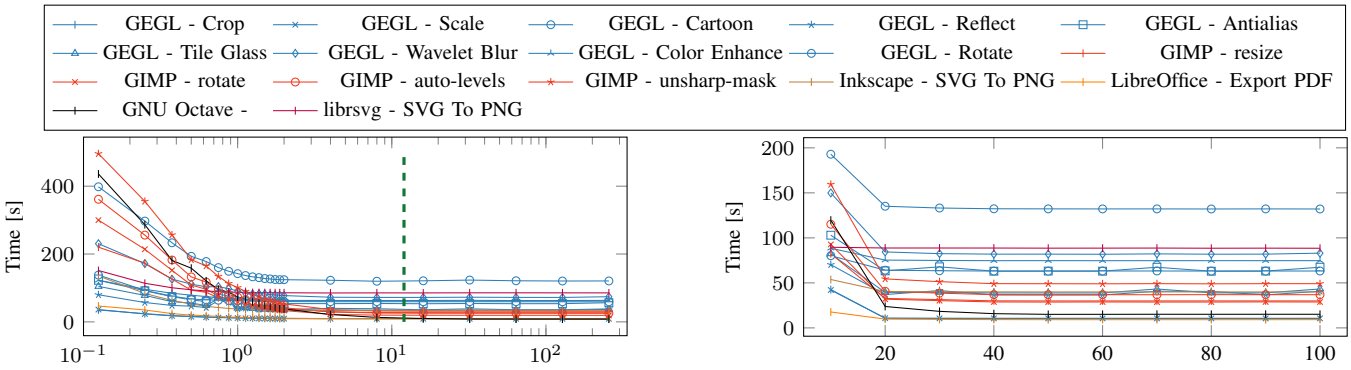
We can also model Rowhammer behavior and our mitigation formally using Linear Temporal Logic (LTL). We define a Rowhammer bit flip as a violation of a temporal safety property within a refresh interval due to excessive row activations to aggressor rows, with $\texttt{ACT}(r)$ being true *when* the row r is activated; $\texttt{Flip}(v)$ being true *when* a Rowhammer bit flip occurred in victim row v; $\texttt{Aggr}(v)$, the set of aggressor rows adjacent to a victim row v; $\texttt{Bank}(v)$, the DRAM bank containing victim row v; T_{refresh}, the DRAM refresh interval (e.g., 64 ms); θ, the empirical Rowhammer threshold (*i.e.*, maximum safe number of activations within T_{refresh} the kernel or user will configure); and L, the per-bank row activation limit enforced by *Memory Band-Aid*. Additionally we define ε as the natural charge leakage per cycle, γ as the additional charge drain per activation, $c_v(t)$ as the remaining charge in v at time t, and C_{flip} as the remaining charge threshold where a bit flip can occur if the remaining charges drops *below* this threshold. The initial charge is after the refresh is $c_v(0)$. The charge decays due to natural leakage at rate ε and is additionally reduced by γ per activation of any aggressor row. Thus, the remaining charge in the victim row v at a point in time t in a refresh interval $t \in [0, T_{\text{refresh}}]$ is:

$$c_v(t) = c_v(0) - t \cdot \varepsilon - \gamma \cdot \sum_{r \in \texttt{Aggr}(v)} \texttt{ACT}(r).$$

$\texttt{Flip}(v)$ can only be true if $c_v(t) < C_{\text{flip}}$ at any point in time. However, the charge within a refresh interval can only decrease, *i.e.*, $c_v(T_{\text{refresh}}) \leq c_v(t)$ for all $t \in [0, T_{\text{refresh}}]$. This implies that $\texttt{Flip}(v)$ can only be true in a refresh interval if $c_v(T_{\text{refresh}}) < C_{\text{flip}}$ in this refresh interval. That is, a system is secure if

$$\begin{aligned}
&\square\neg\texttt{Flip}(v) \\
\Leftrightarrow &\square\neg(c_v(t) < C_{\text{flip}}) \\
\Leftrightarrow &\square\neg\left(c_v(0) - T_{\text{refresh}} \cdot \varepsilon - \gamma \cdot \sum_{r \in \texttt{Aggr}(v)} \texttt{ACT}(r) < C_{\text{flip}}\right).
\end{aligned}$$

With Memory Band-Aid, the total number of activations to any rows in $\texttt{Bank}(v)$ within a refresh window is limited to $L \cdot T_{\text{refresh}}$. Consequently, we now can use L as an upper bound



(a). AMD Epyc 8024P (A2). The dashed line indicates a secure Memory Band-Aid configuration mitigating many-sided attacks.

(b) Intel Xeon 4410T (I2)

Fig. 11. Phoronix Productivity Testsuite Results for varying memory bandwidth limitations on 2 of the evaluated systems.

TABLE IV
PHORONIX PRODUCTIVITY TESTSUITE PERFORMANCE RESULTS OF OUR PROOF-OF-CONCEPT IMPLEMENTATION.

Benchmark	**Intel I2**			**AMD A2**				
	Unlim.	10 % Per-system	Insecure PoC Overhead	Unlim.	128 MiB/s Per-system	**Secure PoC Overhead**	128 MiB/s Per-bank	**Full Impl. Overhead**
GEGL - Crop	10.6	42.6	303.4 %	9.7	35.1	**263.2 %**	10.1	**4.2 %**
GEGL - Scale	10.6	42.1	298.2 %	9.0	36.3	**301.9 %**	9.5	**5.2 %**
GEGL - Cartoon	132.1	192.9	46.1 %	120.6	398.2	**230.1 %**	122.4	**1.5 %**
GEGL - Reflect	40.7	70.2	72.6 %	38.2	80.0	**109.3 %**	39.3	**2.7 %**
GEGL - Antialias	67.4	103.0	52.8 %	56.5	122.5	**116.8 %**	57.4	**1.6 %**
GEGL - Tile Glass	43.2	82.0	89.7 %	37.0	104.7	**182.5 %**	38.1	**2.9 %**
GEGL - Wavelet Blur	83.0	149.9	80.5 %	74.2	230.5	**210.9 %**	77.1	**3.9 %**
GEGL - Color Enhance	74.7	88.5	18.4 %	63.3	122.8	**94.0 %**	63.3	**0.0 %**
GEGL - Rotate 90 Degrees	63.2	80.2	26.9 %	59.2	138.0	**133.2 %**	59.5	**0.5 %**
GIMP - resize	30.0	82.5	175.2 %	26.8	220.2	**721.0 %**	28.8	**7.4 %**
GIMP - rotate	28.8	92.7	222.5 %	19.2	299.9	**1 464.8 %**	21.1	**10.2 %**
GIMP - auto-levels	36.9	115.2	212.3 %	25.3	361.4	**1 327.7 %**	28.0	**10.8 %**
GIMP - unsharp-mask	49.1	159.4	224.6 %	31.2	495.9	**1 488.6 %**	33.7	**8.0 %**
Inkscape - SVG Files To PNG	39.9	53.8	34.6 %	34.0	135.4	**298.1 %**	34.9	**2.5 %**
LibreOffice - 20 Docs To PDF	9.2	17.7	92.4 %	8.7	46.7	**438.0 %**	8.9	**2.8 %**
GNU Octave Benchmark -	15.1	119.7	691.2 %	8.1	436.0	**5 312.0 %**	13.1	**62.8 %**
librsvg - SVG Files To PNG	88.5	89.6	1.2 %	85.4	151.4	**77.2 %**	85.5	**0.1 %**
Mean	37.3	81.5	118.6 %	30.7	150.2	**388.8 %**	32.8	**6.7 %**

We tested our proof-of-concept on the Intel Xeon 4410T (I2) and AMD Epyc 8024P (A2) without bandwidth limits, with the most restrictive bandwidth limit, and the total memory system bandwidth of a full Memory Band-Aid implementation with per-bank limits. Lower is better.

for the sum of activations to aggressor rows to simplify the condition to

$$\square\neg\left(c_v(0) - T_{\text{refresh}} \cdot \varepsilon - \gamma \cdot L < C_{\text{flip}}\right).$$

We further assume that the memory without any hammering remains stable, *i.e.*,

$$\square\neg\left(c_v(0) - t \cdot \varepsilon \leq C_{\text{flip}}\right).$$

We define $C_{\text{flip}} + T_{\text{refresh}} \cdot \varepsilon$ as the minimal initial charge C_{minimal} required to remain stable without hammering by the end of the refresh interval. With this assumption, we can rewrite the condition as

$$\square\neg\left(c_v(0) - T_{\text{refresh}} \cdot \varepsilon - \gamma \cdot L < C_{\text{minimal}} - T_{\text{refresh}} \cdot \varepsilon\right)$$

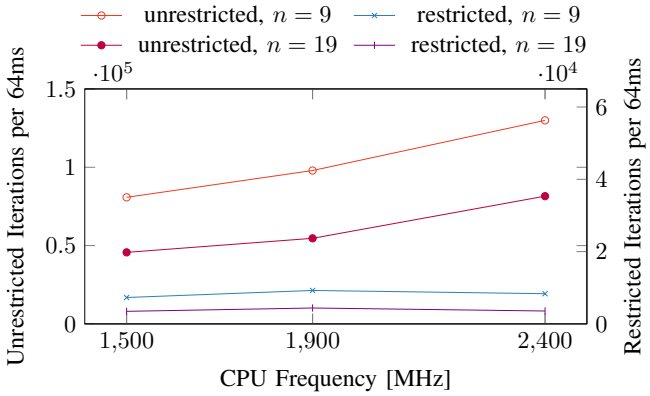
and simplify to

$$\square c_v(0) - \gamma \cdot L > C_{\text{minimal}}$$

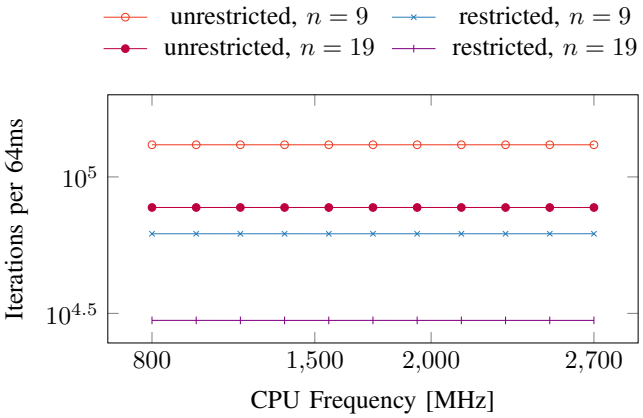
Hence, we can conclude that if the per-bank row activation limit L is chosen such that

$$L < \frac{c_v(0) - C_{\text{minimal}}}{\gamma}$$

then it is guaranteed that the Rowhammer condition can never be reached, *i.e.*, a threshold L exists such that a system is provably secure against Rowhammer bit flips. Note that this holds at all times and with all possible Rowhammer attack patterns, including bursts and novel ones that may be discovered in the future as it does not depend on the specific aggressor rows or activation patterns. Since the limit L is independent of the processor frequency, this guarantee holds for all refresh intervals, even if the processor frequency changes. Effects other than Rowhammer that do not leak due to row activations, such as physical defects, are outside the scope of this formalization. Finally, should new attacks arise that hammer from multiple cores, applying the limit L on a

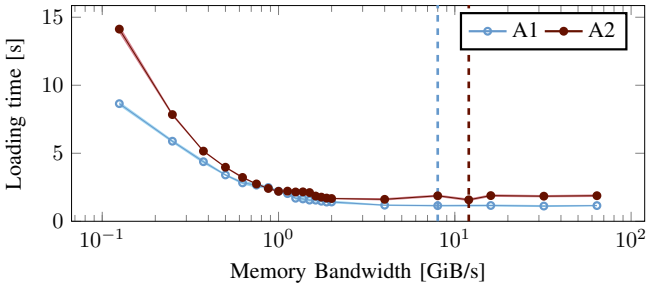


(a). AMD Epyc 8024P (A2). Restricted and unrestricted iterations are scaled differently.

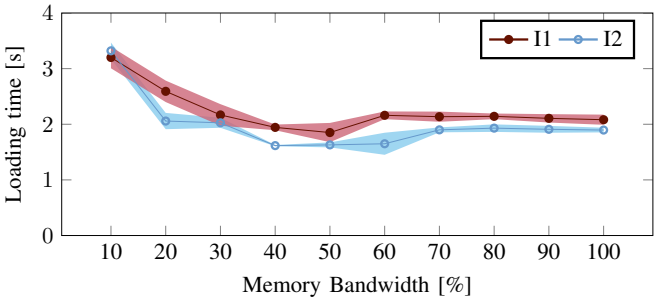


(b). Intel Xeon 4410T (I2)

Fig. 12. Impact of the processor frequency on Flush+Reload loops with $n = 9$ and $n = 19$ addresses. The plot shows the iterations achieved per refresh interval on the AMD Epyc 8024P (A2) and Intel Xeon 4410T (I2).



(a). AMD Ryzen 7700X (A1) and Epyc 8024P (A2). The dashed line indicates a secure Memory Band-Aid configuration mitigating many-sided attacks.



(b). Intel Xeon 4514Y (I1) and Xeon 4410T (I2)

Fig. 13. Average time to launch Chrome headless and access `https://google.com`. $N = 20$. The area shows the standard deviation.

per security-domain basis, e.g., per user, per virtual machine, or per container, the formal security guarantees still hold.

While our security analysis shows that a correctly chosen limit L is sufficient to prevent Rowhammer bit flips, a misconfigured limit L may still lead to bit flips if the limit is too high or to unnecessary performance overhead if the limit is too low. However, since accesses are delayed rather than blocked when the limit is hit, Memory Band-Aid does not introduce a denial-of-service risk.