Software-Based RowHammer Mitigation with Randomized Memory Allocation

Kento Murata, Soramichi Akiyama
Ritsumeikan University, Osaka, Japan

ACM Reference Format:
Kento Murata, Soramichi Akiyama. 2024. Software-Based RowHammer Mitigation with Randomized Memory Allocation. In *Proceedings of Asia-Pacific Workshop on Systems (APSys'24).* ACM, New York, NY, USA, 1 page. https://doi.org/XXXXXXX.XXXXXXX

1 Background and Problem

RowHammer [1] is an attack where an attacker can rewrite memory regions inaccessible to them. It is based on electromagnetic inference inside DRAM chips and thus hard to prevent. RowHammer can cause privilege escalation and arbitrary binary execution.

Using Buddy system for memory management makes a system vulnerable to RowHammer. This is because Buddy system (1) tries to allocate as contiguous free pages as possible and (2) is deterministic. This enables an attacker (i) to allocate a physically contiguous memory region and (ii) to return a single page from the region and make the victim reuse that page, which is now physically sandwiched between attacker-controlled pages.

Existing hardware-based mitigation techniques count frequently accessed memory locations and trigger themselves based on some thresholds. The problem is that the higher the memory capacity, the lower the threshold due to stronger electromagnetic inference. Experimental results show up to 600% overhead when the threshold is very small [2].

2 Proposal and Early Results

We propose a memory management mechanism that allocates random pages to mitigate RowHammer. The main idea is that allocating random pages prevents an attacker from being able to physically sandwich a victim memory page. Our system is designed not to incur many memory accesses when allocating random free pages and Figure 1 shows how it works. We divide all memory pages into N blocks of M pages ($M \geq N$) and select `r[i]`$^{\text{th}}$ page from each block (`i = 0, ..., N`), where `r` consists of the first N elements of a random permutation of an array `{0, ..., M-1}`. We update `r` to a new permutation that does not match any previously used one after it is used by rotating the permutation to one direction. This guarantees that every allocation attempt

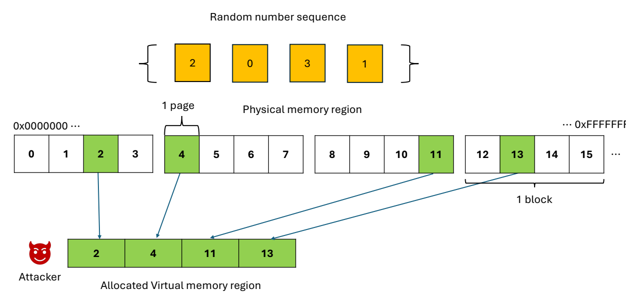

Figure 1: How our system allocates pages randomly

can return "new" pages (no overlaps with any previously-allocated ones) without trial-and-errors.

To confirm that an attacker cannot physically sandwich a victim page in our system, we implement our system and Buddy system on top of the SE mode of gem5 and conduct the following experiment. (1) An attacker function allocates a memory region of 2 MiB with `mmap` and returns a single page in the region with `munmap`. (2) A victim function allocates a single page using `mmap`. (3) If the physical address of the page allocated to the victim is the same as that of the page returned by the attacker, the attacker succeeds to physically sandwich a victim page. The results is that the victim was forced to reuse the returned page (address `0xca000`) in Buddy system, while in our system the physical addresses of the attacker-returned and victim-allocated pages were completely different (`0x11db5000` and `0x18c1a000`).

3 Future work

Future work includes evaluating the performance overhead of our system. We are especially interested in how modern cache prefetchers are affected by random memory allocation.

References

[1] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ISCA*. 361–372.

[2] Anish Saxena, Saurav Mathur, and Moinuddin Qureshi. 2024. Rubix: Reducing the Overhead of Secure Rowhammer Mitigations via Randomized Line-to-Row Mapping. In *ASPLOS*. 1014–1028.

APSys'24, September 4 - 5, 2024, Kyoto, Japan
2024. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX