RISC-H: Rowhammer Attacks on RISC-V

Michele Marazzi and Kaveh Razavi
ETH Zurich

***Abstract*—The first high-end RISC-V CPU with DDR4 support has been released just a few months ago. There are currently no Rowhammer studies on RISC-V devices and it is unclear whether it is possible to compromise systems on these newer architectures. With RISC-H, we aim to fill this gap by overcoming a number of challenges: first, the DRAM functions of the memory controller are not disclosed, which we reverse engineer via the bank-conflict side channel. Second, we discover that hammering many rows achieves a significantly low activation throughput, making Rowhammer unsuccessful. We determine that this low performance is caused by a contention in the memory subsystem when aggressor rows share certain physical address bits and slow ordering instructions. To address this challenge, we leverage different column addresses to reduce contention, and we rely on a novel approach for ordering memory accesses by inserting surgical delays in the access patterns. Combining these insights, our new Rowhammer attack, called RISC-H, can trigger the first DDR4 bit flips from a RISC-V CPU. These results show that the RISC-V ecosystem is similarly affected by Rowhammer and further highlights the need for effective mitigations.**

I. Introduction

Rowhammer on modern DRAM has mostly been successful on high-end complex CPUs from Intel [1–5] and AMD [6]. These mature products, resulting from years of improvements, render instruction execution fast and the memory controllers complex. RISC-V CPUs are comparably in their early stage and it is unclear whether it is possible to trigger Rowhammer bit flips from these platforms. Our results on the first RISC-V processor with DDR4 support [7] shows a very low activation throughput as well as expensive ordering instructions, impeding Rowhammer attacks. This paper shows how the careful selection of physical addresses and the insertion of surgical delays for ordering allows ordered activations with high throughput, enabling the first successful Rowhammer attack on RISC-V.

DRAM functions. Rowhammer causes disturbance errors on *victim* rows that are in close proximity of an *attacker* row. Memory controllers map physical addresses to DRAM banks and rows using proprietary functions, exploiting DRAM parallelism to increase performance. Hence, the success of a Rowhammer attack relies on reverse engineering these functions. In RISC-H, we use the bank-conflict side channel [8] to reverse engineer the bank and row functions. We discover that our target RISC-V CPU employs a linear mapping instead of the common XOR-based functions of Intel and AMD CPUs for bank addressing [6, 8].

Activation throughput. A key aspect for a successful Rowhammer attacks is a high activation rate generated by the memory controller to both induce bit flips and to bypass deployed mitigations [3, 6, 9, 10]. On the Sophon CPU [7], we discover that subsequent accesses to certain memory accesses are surprisingly slow. Our characterization shows that this is caused by a contention in the memory subsystem when memory addresses share columns bits. Consequently, we are able to increase the memory activation rate by distributing subsequent memory requests among different columns.

Memory ordering. Memory requests are reordered by the memory controller to reduce the number of generated activations. First, this lowers the effective activation throughput, required for Rowhammer. Second, DDR4 devices deploy Target Row Refresh (TRR) as a Rowhammer mitigation [2, 11–13], which can be bypassed by activating aggressor rows in complex patterns [1, 2]. To prevent the memory controller from reodering these patterns and making them ineffective, researchers employ fencing instructions [1] or pointer chasing [4]. We show that both these options significantly reduce the activation rate on the RISC-V CPU. Instead, we devise a novel approach to order memory requests. We make the key observation that memory ordering can be achieved by carefully delaying the requests. By exploiting the row buffer hit as a side channel, we are able to identify the right amount of delay, which we induce via NOP instructions.

We combine our insights to build RISC-H, the first Rowhammer attack on RISC-V. RISC-H is able to obtain 841 bit flips in 6 h of fuzzing on a supported DIMM. These bit flips are highly repeatable (on average, 74% of the times), enabling reliable Rowhammer exploitation.

Contributions. The following summarizes our contributions:

1. We reverse engineer the proprietary DRAM functions of the Sophon memory controller.
2. We identify, characterize, and describe how to avoid a new memory bottleneck that severely slows down the activation rate.
3. We devise and demonstrate a novel method to efficiently order memory requests by surgically-inserted delays.
4. We demonstrate Rowhammer bit flips on RISC-V for the first time.

II. Background

In this section, we introduce DRAM (§ II-A), Rowhammer (§II-B), and Rowhammer-required CPU primitives (§II-C).

A. DRAM

DRAM devices are used as main memory in current high-end computing systems. These devices provides fast, dense and cheap memory. The DDRx protocol [14] describes how the CPU memory controller (MC) can access DRAM memory, with chips typically mounted on Dual In-line Memory Modules (DIMMs,

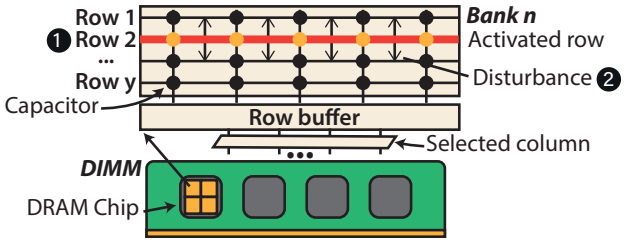

Fig. 1: **DRAM Architecture and Rowhammer.** Memory is organized in many rows in different banks. A row ACT **(1)** causes disturbance to nearby data **(2)**.

Fig. 1). Currently, the majority of DIMMs are DDR4 devices, with the new DDR5 standard [15] being released a few years ago.

DRAM Hierarchy. The MC needs to respect a logical hierarchy to access memory. Internally in a DRAM chip, memory is obtained as densely packed capacitors distributed across different DRAM banks [16, 17]. Each bank has multiple rows, and each row has many data columns. When the MC accesses memory, it needs to specify the bank, the row and the column associated to the data. Prior to accessing the specific column, the MC needs to issue an activation (ACT) to the row [18]. Each bank can only have one row activated at a time. To deactivate an active row in a bank, the MC issues a precharge (PRE). Once the row has been activated, the MC can finally read or write data by specifying a column. Further reads/writes to columns of an activated row do not require additional activations and are described as *row buffer hits*. Due to the internal DRAM circuitry, activating and precharging rows is slow. For this reason, MCs will typically try to optimize the order of generated memory requests such that ACTs and PREs are reduced. This reordering is done by using a memory request buffer.

DRAM Capacitors. DRAM is based on capacitors, which are elements capable of storing charge. A single capacitor allows to encode a single bit of data via the stored charge. Due to the manufacturing capabilities of integrated circuits (ICs), these capacitors are extremely small and compact [16]. This compactness is kept by the highly hierarchical internal structure, which enables memory chips with large addressability and cheap price. Because capacitors leak charge, MCs need to send a refresh command (REF) every tREFi (7.8 us on DDR4) such that the DRAM chip has time to restore rows to their full values. An entire chip is fully restored after 8192 refreshes (tREFW, 64 ms on DDR4). Without these refreshes, stored data would get corrupted.

B. Rowhammer

The highly-packed capacitors require very dense circuitry to provide the market with cheaply-produced memory. Unfortunately, such IC scaling has come with drawbacks in terms of memory reliability. In 2014, researchers demonstrated that aggressor-row activations can have an effect on nearby victim-rows on DDR3 devices [19]. The effect, known as Rowhammer, causes an increased charge leakage that can corrupt victim data without directly accessing it. To trigger Rowhammer, aggressor rows that are physically nearby a victim row are activated repeatedly for a large number of times, known as *Rowhammer threshold* (e.g., 50 K), before the victim row is refreshed by a REF command. This vulnerability has been subsequently exploited in many different ways from different attack vectors [4, 5, 20–23] and deeply characterized [10, 24–30]. As a response, DRAM vendors have deployed Rowhammer mitigations known as Target Row Refresh (TRR) on DDR4 devices. TRR implements Rowhammer detection mechanisms, which are followed by the refresh of the victim row [2, 11]. On DDR4 devices, TRR has been shown to be flawed when advance row activation patterns are used [1]. Research efforts have provided industry many different Rowhammer mitigations based on alternative approaches [9, 17, 18, 31–40].

C. CPU Primitives

To reliably perform Rowhammer on DDR4 devices from different vendors, researchers have relied on two key CPU primitives. First, the activation rate (i.e., how many ACTs per second the MC issues) is maximized [3, 6, 9, 10]. Second, the aggressor rows that are activated are based on complex patterns to bypass TRR [1]. To ensure the row activation order, researchers either used pointer chasing or CPU fencing instructions [1, 4, 6].

III. Overview

With our research, we aim to successfully flip DRAM bits via Rowhammer on a RISC-V CPU for the first time. To achieve this goal, we face multiple challenges.

First, the MC maps physical addresses to specific banks and rows in a way that is not disclosed. This mapping is fundamental to perform Rowhammer, as the aggressor and victim rows need to be placed in physical proximity.

Challenge 1. Reverse engineer the MC DRAM functions to link physical addresses to DRAM addresses.

We solve challenge 1 in Section § IV by using the bank conflict side channel [8]. For this purpose, we implemented a multi-thread counter to measure time.

A high activation rate by the MC is a key Rowhammer primitive. We seek to understand if the RISC-V CPU is capable of generating a high activation throughput while allowing for complex row activations patterns.

Challenge 2. Maximize the DRAM activation rate (i.e., ACT/s) without losing aggressors pattern generality.

We address this challenge in Section § V. We identify a memory bottleneck that substantially slows down the ACTs rate. In particular, subsequent memory accesses that share particular bits of the physical address create contention on the memory subsystem. By distributing temporally-close memory accesses across different columns, we are able to heavily increase the activation throughput.

The last challenge is to keep a high row activation rate without losing control over the order of row activations. As the RISC-V CPU might not be as complex as Intel and AMD counterparts, we identified that fencing and other common

Tbl. I: Hardware used in RISC-H.

Processor		Cores		DRAM	
Vendor	Sophon	Vendor	T-Head	Vendor	X
Model	SG2042	Model	C920	Memory	8 GB
Memory	DDR4	Frequency	2GHz	Total Banks	16
System Cache	64MB	L2 cache	1MB	Rows/Bank	64 K
Core Clusters	16	Cores/clusters	4	Production Yr.	2018

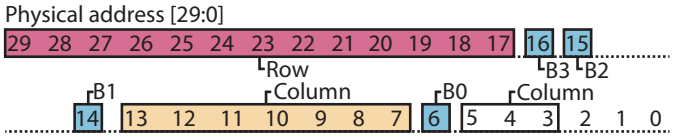

Fig. 2: **Reverse engineered DRAM Functions.** The memory controller applies a linear mapping of the bits 6, 14, 15, and 16 to the DRAM banks.

ways to hold activation order intact have a huge toll on the activation throughput.

Challenge 3. Maintain the order of row activations, without relying on fencing or pointer chasing.

We assume that the MC scheduling policy is based on some variation of the First-Ready First-Come-First-Serve (FR-FCFS) strategies as these are considered standard [41–44], and we solve challenge 3 by carefully delaying memory requests, forcing activation ordering. By exploiting the row hit as a side channel, we prove that our approach is keeping the intended activation order.

By combining all these aspects, we are able to produce Rowhammer bit flips in a fast and reproducible way, obtaining 841 bit flips in 6 hours of fuzzing on a supported DIMM.

IV. Reverse Engineering of DRAM Functions

We report the configuration of our system in Table I. We use the well-known bank conflict side channel to determine the DRAM functions [8, 45, 46]. As this is a timing side channel, we require an accurate and precise method to measure time. RISC-V provides an instruction called `rdcycle`, which is supposed to return the number of cycles executed by the CPU. However, the RISC-V instruction `rdcycle` returns a constant value on our system. We build a counting thread, similar to previous work [47], obtaining a timer resolution of 4.85 ns.

Results. We use the recently released tool AMDRE [6] to reverse engineer the DRAM functions allocating a GiB super page. We adapt AMDRE to rely on shared memory as a counter and to use the RISC-V `fence` instruction. Further, we modify it to use average instead of the minimum and to use more repetitions to reduce noise. We report the results in Fig. 2. In the RISC-V CPU under evaluation, the mapping functions are linear. This differs from the complex XOR-based functions reported for Intel and AMD [6, 8]. We identify column bits by exploiting the row buffer hit, accessing couples of addresses of the same bank where one bit differs. We categorize the bit as column if the access is fast and classify the remaining as rows bits.

V. Maximizing the Activation Rate

A key aspect of successful Rowhammer bit flips is a high ACT rate generated by the memory controller. Given the DDR4

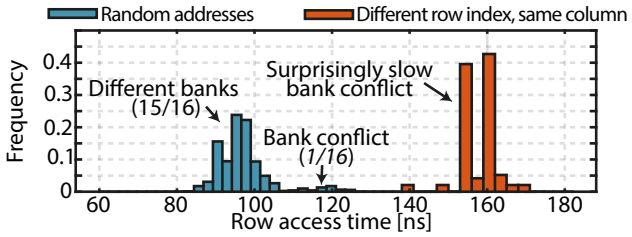

Fig. 3: **Histogram of access times.** We report the row access time for couple of random addresses and for addresses that differ only for the row index.

standard, the minimum time between different ACTs to the same bank is given by `tRC` [14], which for our system is 46.5 ns. We seek to understand if the RISC-V CPU is capable of generating such a high amount of activations per second.

Baseline performance. To evaluate the activation throughput of the CPU, we measure the time to access an array of 256 different rows that target the same bank. The rows are different, hence we do not require pointer chasing or fencing to avoid memory requests reordering. Such reordering would inflate the throughput by reducing the number of activations, exploiting row buffer hits. We discover that the average access time per row is severely slow, around 210 ns. We repeat the experiment to understand if there is a dependency to the number of accessed rows. The access time saturates to 210 ns with a high number of accessed rows, while it is slightly faster (180 ns/ACT) when very few rows are accessed (e.g., 12 rows). All these values are much higher than the `tRC` of the system, making a Rowhammer attack unlikely to succeed.

We speculate that the design of the CPU might not be as optimized as for Intel and AMD devices, resulting in contention on a particular microarchitectural resource. Specifically, we formulate the hypothesis that this contention is address-dependent, for example, due to accessing internal cache blocks, other sub-blocks, or due to the internal MC design. We devise the following experiments to investigate this effect, in which we use contiguous memory obtained from a GiB super page.

Identifying the memory bottleneck — first experiment. Our aim is to identify if two subsequent memory requests have resource contention, and if this contention can be avoided by varying their addresses. To this end, we first generate addresses that only differ for the row index. Given the previous results, we expect their combined access time to result in a bank conflict. We show the results in Fig. 3, which further includes the histogram of the bank-conflict side channel previously used. Surprisingly, the access time of different rows is always slower than the bank-conflict time. Note that the access time is slightly lower than in the previous experiment (160 ns compared to 180 ns), as we directly dereference registers for a more controlled experiment instead of accessing an array in a loop.

Identifying the memory bottleneck — second experiment. The results from AMDRE are generated by using random couples of addresses. These random couples cause bank conflicts, yet their access time is faster than our experiment. Therefore, as we are now using almost identical addresses, we wish to see if particular parts of them are causing contention that

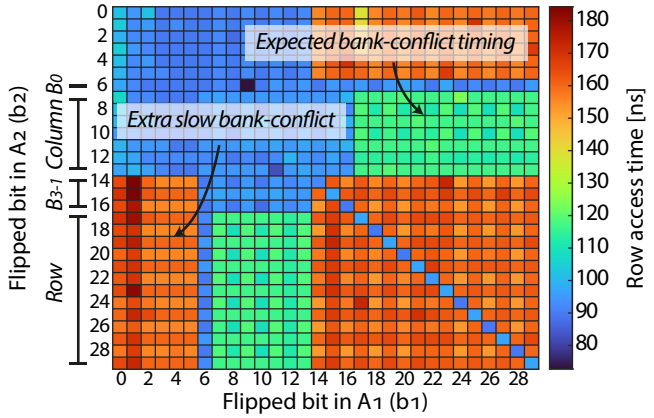

Fig. 4: **Time access experiment.** We access two addresses (A_1, A_2), where $A_1 = base \oplus b_1$ and $A_2 = base \oplus b_2$, and we report the row access time.

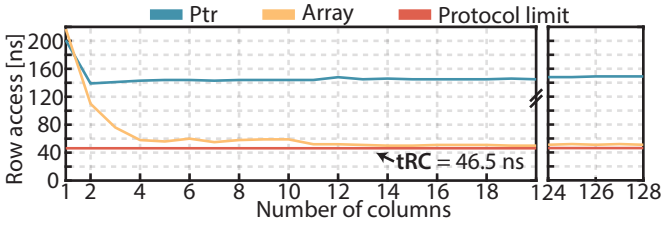

Fig. 5: **Average access time when using different columns.** Pointer chasing (Ptr) access time is severely slower compared to Array access (Array). When multiple columns are used, the access time of Array becomes close to tRC.

increases the slowdown. In what follows, instead of changing only the row index, we vary each address bit at the time. Starting from a base address, we measure the combined access time of two addresses, A_1 and A_2, where $A_1 = base \oplus b_1$ and $A_2 = base \oplus b_2$. We test all combinations of b_1 and b_2 between $b_0 - b_{29}$, and report the results in Fig. 4. From the results we identify three categories of access speed: (i) fast, (ii) semi-slow, and (iii) slow. Semi-slow timings correspond to the bank-conflict found during the DRAM function reverse engineering. Instead, slow timings correspond to the "extra" slow access.

Address dependency of the bottleneck. By flipping a bank bit (e.g., b_{14}) only in one address, we would expect the timing to be always fast, as A_1 and A_2 would target different banks. Instead, in many of the combinations the access time is slow. By varying bits between $b_7 - b_{13}$, different bank accesses result (as expected) in fast memory accesses. By varying row bits (e.g., b_{19}) and any bit between $b_7 - b_{13}$, we obtain the expected bank conflict timing. We conclude that subsequent accesses of addresses with identical bits $b_7 - b_{13}$ are severely slowed down. Coincidentally, these bits correspond to the column bits of a row. We now investigate how many different columns, in an access loop, are required to obtain a high ACT rate.

Columns dependency. We distribute from 1 to 128 different columns to 256 different rows accessed in a loop. For each setup, we report the average access time *per row* in Fig. 5. After a few columns, the access time saturates close to the protocol limit (`tRC`). For simplicity, in the remainder of the paper, we will always linearly distribute the columns of accessed memory

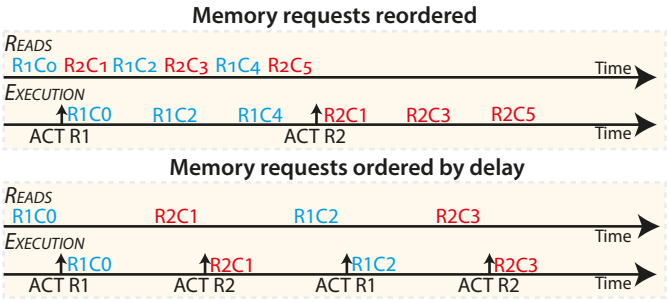

Fig. 6: **Memory requests ordering.** In the first case, no ordering is enforced and different memory requests are merged in single activations. In the second case, delayed memory requests induce multiple activations.

region across all possibilities (i.e., 128). The reader should note that this is required *independently* from the accessed rows (i.e., if the pattern is accessing the same row or different ones).

VI. Enforcing Memory Requests Order

We now analyze the impact of enforcing memory requests order on the activation rate.

Effect of memory requests reordering. MCs try to reduce activations by reordering memory requests that would target the same row. For Rowhammer attacks, this has two damaging effects. First, the *effective* ACT throughput will be reduced, as rows will be kept open longer than required. Second, state-of-the-art Rowhammer patterns are complex and require precise ordering of aggressor rows to fool the deployed TRR mechanisms [6]. If requests are reordered, these pattern would get scrambled, reducing the possibility of Rowhammer success.

Speed of pointer chasing and fence. The two main ways to order memory requests is to use pointer chasing and the `fence` instruction. We now evaluate their speed on our RISC-V CPU, by measuring the average access time of 256 addresses. Accesses divided by `fence` have an average speed of 210 ns/ACT, regardless of the number of columns used. Pointer chasing increases its speed when temporally-close accesses are distributed among different columns, however, the access is still severely slow saturating at around 145 ns/ACT (Fig. 5).

In Section §V, we observed that the system is capable of reaching high access speed (55 ns/ACT). Therefore, we cannot rely on `fence` and pointer chasing as strategies for preserving the order of memory requests. With our novel delayed memory requests, we now demonstrate how it is possible to reach high speed while ensuring ordering.

A. Memory ordering via delayed accesses

We assume the MC scheduling policy to be based on a variation of First-Ready First-Come First-Serve (FR-FCFS), as these are considered standard strategies [41–44]. When these policies are employed, the memory request buffer will be used to batch requests that go to the same rows, incrementing row hit and decreasing the number of ACTs (Fig. 6) [41].

We aim to enforce memory ordering by delaying memory requests. If the request buffer does not (yet) contain requests that can be merged, it should eventually perform the activation. Likewise, if a request has been in the queue for a long time (i.e.,

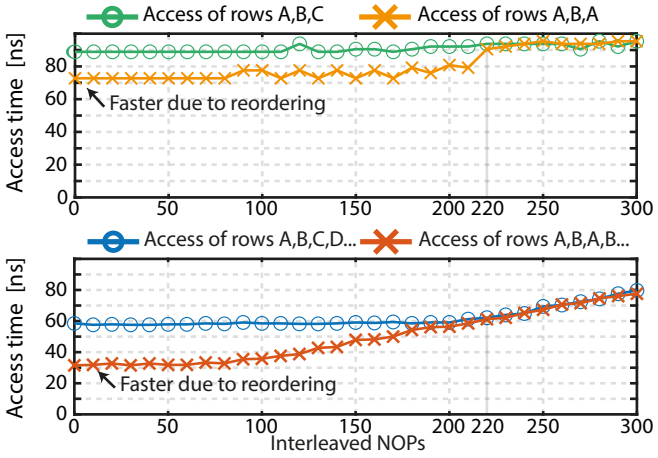

Fig. 7: **Delayed access experiment.** We measure 3 ACTs (A,B,C) and compare it to 3 requests that can be sped-up by reordering (A,B,A). When enough NOPs are inserted, the MC does not reorder accesses anymore. We repeat the experiment with 16 different rows compared to the access pattern (A,B)×8.

old request), it should be prioritized over new memory requests to avoid starvation [42]. We explain this concept in Fig. 6.

We perform our technique by placing NOPs in between the different requests. As the CPU does not expose any relevant performance counters (e.g., issued activations), we must rely on a carefully crafted experiment to validate our method and to understand for how long the requests should be delayed for.

NOPs-delayed requests experiment. An access pattern with 3 different rows (e.g., `A-B-C`) forces the MC to issue 3 ACTs. Instead, a pattern that targets the same row twice (e.g., `A-B-A`) allows the MC to reorder requests, resulting in only 2 ACTs and a row buffer hit. We perform an experiment to measure this difference. In Fig. 7, we report the average time per memory request of the two cases. The pattern `A-B-A` is evidently faster due to memory reordering. If our hypothesis is correct, the pattern `A-B-A` should generate 3 ACTs when memory requests are delayed long enough. We interleave the accesses with a varying number of NOPs, from 0 to 300, and report the results in Fig. 7. When around 220 NOPs are inserted, the pattern `A-B-A` starts behaving as expected.

Now, we generalize the experiment and measure the effect when *multiple* reordering are possible, as this is the typical Rowhammer case. We repeat the experiment by accessing 16 different rows (e.g., `A-B-C-D...`) compared to eight times the couple `A-B`. As previously found, around 220 NOPs the behavior converges to the expected timing (Fig. 7).

Finally, we evaluate the overhead of our technique. Repeating the experiment of Fig. 5, we measure the average access time for a loop of different rows with 220 NOPs interleaved. The result is an average slow down of only 3 ns per access, which shows that our new ordering technique has very low overhead and is viable to perform Rowhammer on the RISC-V CPU.

VII. RISC-H

We combine all the previous observations and results to perform Rowhammer on the first high-end RISC-V CPU. We now explain the setup and the results.

Setup. We allocate 1 GiB of memory as transparent huge pages (2 MiB) and verify that all the pages result in bank conflict. This is expected, as the highest bank bit is the 16th (i.e., lower than 2 MiB, Fig. 2). Then, we initialize the 1 GiB of memory with 32 different row patterns of 8 Bytes, selected by hashing the memory address. This allows faster execution times compared to the performance toll of using randomized memory to check for bit flips. The patterns are based on classic repetitions of `0xAA`, `0xEE`, `0x00`, `0xFF`, and variations.

We perform Rowhammer by generating non-uniform patterns, as they represent state-of-the-art [1, 6]. We do not use Blacksmith as a dependency (asmjit) is not yet ported to RISC-V, but instead rely on our own implementation called RISC-H. All memory requests are interleaved by 220 NOPs and linearly distributed across 128 columns. We fuzz each pattern for a duration of $4 \times tREFW$. After fuzzing, we check for bit flips for only the addresses that correspond to the same targeted bank and whose rows are nearby the aggressors. This further improves the performance of RISC-H. Once we identify a bit flip, we test its repeatability by testing the same pattern 10 times for double the time. We further test hammering by using `fence`, pointer chasing, and without any ordering. All the experiments take place in a controlled environment at 23° C.

Results. We discover that on the tested DIMM, we are able to trigger bit flips with double-sided patterns (i.e., frequency-based patterns are not necessary) of which we report the results. With our complete approach to RISC-H, we obtain 841 unique bit flips in only 6 h of fuzzing. The first bit flip occurs within one minute after the fuzzing is started (i.e., after data has been initialized to DRAM). Bit flips have a high repeatability, with an average success rate of 7.4 out of 10 times.

We confirm that we did not obtain bit flips when the ordering was absent (i.e., no NOPs), or when it was enforced by `fence` or by pointer chasing. In conclusion, our delayed-request ordering is necessary to get Rowhammer bit flips on the CPU.

Limitations and future work. We evaluated RISC-H only on one DIMM. This is due to the extremely limited memory support provided by the CPU. We tested 85 DDR4 DIMMs present in our lab, of which only one resulted in the system booting. We tried to clone the SPD values of the working DIMM to different modules with the same timings and geometry, and we further tried 22 SODIMMs connected via an adapter. Unfortunately, this also did not result in booting. Future studies should assess RISC-H on multiple devices in case the DIMM support will be extended.

VIII. Conclusion

With RISC-H, we proved that Rowhammer bit flips are possible on the first high-end RISC-V CPU. To perform Rowhammer, we had to overcome multiple challenges. First, we identified the undisclosed DRAM functions by exploiting the bank-conflict side channel. Then, we discovered a memory bottleneck related to the address of subsequent memory requests, which made the ACT rate insufficient to trigger bit flips. As a solution, we distributed the activations across the many row columns. Lastly, we devised a novel technique to

enforce memory requests order with high performance. By carefully delaying accesses with NOP instructions, we enforce row activations without relying on slow fence instructions or pointer chasing. To the best of our knowledge, we are the first to demonstrate bit flips on a RISC-V CPU.

Acknowledgment

We thank our anonymous reviewers and Patrick Jattke for their valuable feedback. This work was supported by the Swiss National Science Foundation under NCCR Automation, grant agreement 51NF40 180545, and the Swiss State Secretariat for Education, Research and Innovation under contract number MB22.00057 (ERC-StG PROMISE).

References

[1] P. Jattke, V. van der Veen, P. Frigo, S. Gunter, and K. Razavi, "BLACKSMITH: Scalable Rowhammering in the Frequency Domain," in *IEEE S&P*, 2022, pp. 716–734. [Online]. Available: https://ieeexplore.ieee.org/document/9833772/

[2] P. Frigo, E. Vannacc, H. Hassan, V. v. der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *IEEE S&P*, 2020, pp. 747–762. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9152631

[3] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers," in *IEEE S&P*, 2020, pp. 712–728. [Online]. Available: https://ieeexplore.ieee.org/document/9152654/

[4] 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 '21*, Aug. 2021, pp. 1001–1018.

[5] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading Bits in Memory Without Accessing Them," in *IEEE S&P '20*, May 2020, pp. 695–711.

[6] P. Jattke, M. Wipfli, F. Solt, M. Marazzi, M. Bölcskei, and K. Razavi, "Zenhammer: Rowhammer attacks on amd zen-based platforms," in *33rd USENIX Security Symposium (USENIX Security 2024)*, 2024.

[7] SOPHGO, "Sophon SG2042," 2024. [Online]. Available: https://en.sophgo.com/product/introduce/sg2042.html

[8] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in *USENIX Security '16*, Aug. 2016, pp. 565–581.

[9] 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*, vol. 51, no. 4, pp. 743–755, 2016.

[10] J. S. Kim, M. Patel, A. G. Yaglikci, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques," in *ISCA*, 2020, pp. 638–651. [Online]. Available: https://ieeexplore.ieee.org/document/9138944/

[11] H. Hassan, Y. C. Tugrul, J. S. Kim, V. van der Veen, K. Razavi, and O. Mutlu, "Uncovering In-DRAM RowHammer Protection Mechanisms: A New Methodology, Custom RowHammer Patterns, and Implications," in *MICRO*, 2021, pp. 1198–1213. [Online]. Available: https://dl.acm.org/doi/10.1145/3466752.3480110

[12] 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 arXiv:2302.03591*, 2023.

[13] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang *et al.*, "A 1.1 v 16gb ddr5 dram with probabilistic-aggressor tracking, refresh-management functionality, per-row hammer tracking, a multi-step precharge, and core-bias modulation for security and reliability enhancement," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 1–3.

[14] JEDEC Solid State Technology Association, "DDR4 SDRAM," Sep. 2012. [Online]. Available: https://www.jedec.org/sites/default/files/docs/JESD79-4.pdf

[15] "JESD79-5B: Double Data Rate 5 (DDR5) SDRAM," 2022. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-5b

[16] M. Marazzi, T. Sachsenweger, F. Solt, P. Zeng, K. Takashi, M. Yarema, and K. Razavi, "Hifi-dram: Enabling high-fidelity dram research by uncovering sense amplifiers with ic imaging," in *51st IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2024.

[17] M. Marazzi, F. Solt, P. Jattke, K. Takashi, and K. Razavi, "REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations," in *IEEE S&P*, 2023. [Online]. Available: https://comsec.ethz.ch/wp-content/files/rega_sp23.pdf

[18] M. Marazzi, P. Jattke, F. Solt, and K. Razavi, "ProTRR: Principled yet Optimal In-DRAM Target Row Refresh," in *IEEE S&P*, 2022, pp. 735–753. [Online]. Available: https://ieeexplore.ieee.org/document/9833664

[19] 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, pp. 361–372. [Online]. Available: http://ieeexplore.ieee.org/document/6853210/

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

[21] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1–18.

[22] 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 *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 245–261.

[23] K. Mus, Y. Doröz, M. C. Tol, K. Rahman, and B. Sunar, "Jolt: Recovering tls signing keys via rowhammer faults," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 1719–1736.

[24] Z. Lang, P. Jattke, M. Marazzi, and K. Razavi, "Blaster: Characterizing the blast radius of rowhammer," in *3rd Workshop on DRAM Security (DRAMSec) co-located with ISCA 2023*. ETH Zurich, 2023.

[25] H. Nam, S. Baek, M. Wi, M. J. Kim, J. Park, C. Song, N. S. Kim, and J. H. Ahn, "Dramscope: Uncovering dram microarchitecture and characteristics by issuing memory commands," *arXiv preprint arXiv:2405.02499*, 2024.

[26] 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.

[27] A. Olgun, M. Osseiran, A. G. Yağlıkçı, Y. C. Tuğrul, H. Luo, S. Rhyner, B. Salami, J. G. Luna, and O. Mutlu, "An experimental analysis of rowhammer in hbm2 dram chips," in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE, 2023, pp. 151–156.

[28] 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, pp. 1182–1197. [Online]. Available: https://doi.org/10.1145/3466752.3480069

[29] A. G. Yağlıkçı, H. Luo, G. F. De Oliviera, A. Olgun, M. Patel, J. Park, H. Hassan, J. S. Kim, L. Orosa, and O. Mutlu, "Understanding rowhammer under reduced wordline voltage: An experimental study using real dram devices," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2022, pp. 475–487.

[30] 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 *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3807–3824.

[31] T. Bennett, S. Saroiu, A. Wolman, L. Cojocar, and A.-G. Llc, "Panopticon: A Complete In-DRAM Rowhammer Mitigation," 2021. [Online]. Available: https://dramsec.ethz.ch/papers/panopticon.pdf

[32] M. Wi, J. Park, S. Ko, M. J. Kim, N. S. Kim, E. Lee, and J. H. Ahn, "SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling," in *HPCA*, 2023, pp. 333–346. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10070966

[33] J. Woo, G. Saileshwar, and P. J. Nair, "Scalable and secure row-swap: Efficient and safe row hammer mitigation in memory systems," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 374–389.

[34] 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," *arXiv preprint arXiv:2310.09977*, 2024.

[35] M. Kim, J. Park, Y. Park, W. Doh, N. Kim, T. Ham, J. W. Lee, and J. Ahn, "Mithril: Cooperative Row Hammer Protection on Commodity DRAM Leveraging Managed Refresh," in *HPCA*, 2022, pp. 1156–1169. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HPCA53966.2022.00088

[36] 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 *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 699–710.

[37] Y. Park, W. Kwon, E. Lee, T. J. Ham, J. H. Ahn, and J. W. Lee, "Graphene: Strong yet lightweight row hammer protection," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1–13.

[38] A. G. Yağlikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi *et al.*, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 345–358.

[39] A. Saxena, G. Saileshwar, P. J. Nair, and M. Qureshi, "Aqua: Scalable rowhammer mitigation by quarantining aggressor rows at runtime," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 108–123.

[40] 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 *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 1056–1069.

[41] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 146–160.

[42] P. K. Valsan and H. Yun, "Medusa: a predictable and high-performance dram controller for multicore based embedded systems," in *2015 IEEE 3rd international conference on cyber-physical systems, networks, and applications*. IEEE, 2015, pp. 86–93.

[43] W. K. Zuravleff and T. Robinson, "Controller for a synchronous dram that maximizes throughput by allowing memory requests and commands to be issued out of order," May 13 1997, uS Patent 5,630,096.

[44] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 128–138, 2000.

[45] 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 '16*, Aug. 2016, pp. 19–35.

[46] M. Wang, Z. Zhang, Y. Cheng, and S. Nepal, "DRAMDig: A Knowledge-assisted Tool to Uncover DRAM Address Mapping," in *DAC '20*, Jul. 2020.

[47] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, "Aslr on the line: Practical cache attacks on the mmu." in *NDSS*, vol. 17, 2017, p. 26.