Presshammer: Rowhammer and Rowpress without Physical Address Information

Jonas Juffinger[1[0009−0002−0569−1704]], Sudheendra Raghav Neela[1[0009−0009−4845−868X]], Martin Heckel[2[0009−0006−9739−0587]], Lukas Schwarz[1[0009−0008−6145−3296]], Florian Adamsky[2[0009−0002−6642−6904]], and Daniel Gruss[1[0000−0002−7977−3246]]

[1] Graz University of Technology, Graz, Austria
[2] Hof University of Applied Sciences, Hof, Germany

Abstract. Modern DRAM is susceptible to fault attacks that undermine the entire system's security. The most well-studied disturbance effect is Rowhammer, where an attacker repeatedly opens and closes (*i.e.*, hammers) different rows, which can lead to bitflips in adjacent rows. Different hammering strategies include double-sided, hammering two rows sandwiching a victim row, and one-location, hammering a single row. One-location Rowhammer requires no physical address information, as any location in memory is mapped to a DRAM row, and no relation between rows is required for hammering. The recently discovered Rowpress differs from Rowhammer by not hammering rows but keeping them open longer, evident by a disjoint set of affected memory locations.

In this paper, we examine the differences between four attack variants: one-location Rowhammer, a one-location Rowpress variant we developed, double-sided Rowhammer, and double-sided Rowpress on a set of 12 DDR4 modules. Our methodology is to hammer and press the exact same set of physical memory locations in all attack variants. Surprisingly, our results show that on 4 out of 12 DDR4 modules, we were only able to reproduce double-sided Rowhammer but none of the other attack variants. On 2 DDR4 modules, we were able to reproduce all attack variants. We find that the number of unique bitflip locations ranges from 161 to 15 612, when hammering the exact same set of physical memory locations. Our one-location Rowhammer attack induces roughly the same amount of bitflips as double-sided Rowhammer, however, only 61.8 % of bitflip locations overlap. We explain this by one-location Rowhammer inducing bitflips due to the Rowhammer as well as the Rowpress effect, making the differentiation of both methods difficult, therefore, calling it Presshammer. Based on our observed bitflips, we develop the first end-to-end one-location Rowpress attack. One-location Rowpress requires only minimal physical address information that an attacker can acquire through a same-row same-bank side-channel attack. Our end-to-end attack escalates to kernel privileges within less than 10 minutes.

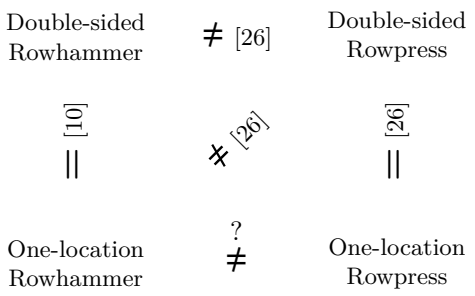


Fig. 1: Claims and observations from different papers. The equal sign indicates that these variants exploit the same root cause, and the unequal sign indicates that these variants have a different root cause. The reference indicates which work this claim is based on.

1 Introduction

A fundamental assumption for secure software systems is the correctness of the underlying hardware. A long line of work has shown that physical glitching could violate this assumption. However, for a long time [2], it was not clear whether it could also be violated in scenarios without a physical attacker. The discovery of Rowhammer as a security issue in 2014 [21] showed that this assumption can be fatally flawed. Rowhammer is a disturbance effect in the DRAM, leading to data corruption that can undermine the entire system's security. To trigger the Rowhammer effect, an attacker repeatedly opens and closes (*i.e.*, hammers) different rows, leading to row activation operations in the DRAM [21]. Different Rowhammer strategies have been discovered, such as single-sided hammering [21], double-sided decoys [9, 17] which all open and close different rows in the same bank, triggering row activations. One-location Rowhammer [10] works slightly differently, as it does not activate multiple rows but instead performs frequent accesses to one single row. This approach has the advantage of not requiring physical address information, as any location in memory maps to a DRAM row, and no relation between rows is required for hammering.

Recently Luo et al. [26] discovered an effect in DDR4, called Rowpress. Rowpress, instead of accessing a single location in a row and then another one in a different row, accesses a larger number of offsets in one row before switching to another row. The goal of this access pattern is to keep the row open as long as possible, as this can induce bitflips due to RAS clobber, *i.e.*, the passing gate effect [12, 15, 52]. Luo et al. [26] report that Rowpress and Rowhammer bitflips belong to largely disjoint sets of memory locations. Rowpress uses physical address information to identify the rows and the offsets within the rows to target.

Comparing the open-source implementations of Rowpress [26] and one-location Rowhammer [10], it is interesting to see that both implementations `main-algo1`

and `main-algo2` by Luo et al. indeed hammer two rows[3], albeit multiple offsets within a row, in a Flush+Reload [21] loop, whereas Gruss et al. only run a Flush+Reload loop on a single address. Hence, following the definition of Rowpress, the attack loop by Gruss et al. should, similarly, actually press a row and not hammer it. However, it is not clear whether the one-location hammering and Rowpress implementations on CPUs, exploit the same or different physical root causes: Gruss et al. [10] reported a difference in bitflip rates between single-sided, double-sided and one-location hammering but no significant difference in bitflip locations, indicating that they all exploit the same root cause. Combining these reports leads to a contradiction, as illustrated in Figure 1. Thus, the relation between the different Rowhammer and Rowpress variants remains unclear.

In this paper, we examine the different Rowhammer and Rowpress variants and find that these effects are not easy to separate. We run Rowhammer and Rowpress tests on 12 DDR4 modules to assess the prevalence of Rowhammer and Rowpress effects on current systems. We find that double-sided Rowhammer works on only 6 out of 12 DDR4 modules. We implement a one-location Rowpress variant, which strictly presses only a single row in contrast to published proof of concepts. However, we find that Rowhammer and Rowpress variants other than double-sided Rowhammer only work on two DDR4 modules in our setup, both with the published proof of concept (PoC) as well as our own implementations. Both results pose a significant difference from the numbers reported by Jattke et al. [17] and Luo et al. [26].

We compare the four attack variants on all DDR4 modules with bitflips in any of the attacks. We hammer and press the exact physical memory locations on the same system in all attack variants by using a setup with fixed memory mappings so that physical addresses remain identical across different Rowhammer and Rowpress tools and runs. The number of unique bitflip locations ranges from 161 to 15 612 on the affected modules when hammering the exact same set of physical memory locations. Our one-location Rowhammer attack induces roughly the same amount of bitflips (14 264) as double-sided Rowhammer (15 612), however, only 61.8 % of bitflip locations overlap. This could be explained by the one-location Rowhammer pattern inducing bitflips due to the Rowhammer as well as the Rowpress effect, making it actually a *Presshammer* attack. We did observe significantly less bitflips with the original Rowpress pattern by Luo et al. [26] that accesses multiple offsets per row than with one-location Rowhammer, even though both keep a row open for a prolonged time.

Based on our observed Presshammer bitflips, we develop a novel one-location Presshammer approach, which requires almost no physical address information and show that an attacker can mount an end-to-end attack by using the same-bank same-row side channel. We demonstrate that our one-location Presshammer attack is practical by obtaining kernel privileges through a page-table corruption attack [44]. The attack runtime with one-location Presshammer is less than 10 minutes on average.

[3] `https://github.com/CMU-SAFARI/RowPress/blob/main/demonstration/main-algo1.cpp#L143`

In summary, we make the following contributions in this work:

1. We systematically compare one-location and double-sided Rowhammer and Rowpress.
2. We study the prevalence of both effects across 12 DDR4 modules and find that 6 of these modules are affected by double-sided Rowhammer, but only 2 by all attack variants, with double-sided Rowhammer flipping the most locations and one-location Rowpress the fewest.
3. We show that our one-location Rowhammer pattern flip bits at locations only partly overlapping with the locations flipped by double-sided Rowhammer. 61.8 % of bitflip locations flipped by one-location Rowhammer also flipped with double-sided Rowhammer, while flipping roughly the same number of bits overall, 14 264 and 15 612 respectively.
4. We present the first one-location Rowpress end-to-end attack, reliably escalating to kernel privileges within less than 10 minutes of attack time.

Outline. Section 2 provides background on DRAM, Rowhammer, and Rowpress. Section 3 presents a systematic comparison between Rowhammer and Rowpress, including our prevalence study. Section 4 shows how one-location Rowpress can be mounted without physical address information. Section 5 presents a PoC attack on page tables, allowing an attacker to escalate to kernel privileges. We discuss mitigations in Section 6 before concluding in Section 7.

2 Background

This section provides background on DRAM, Rowhammer, and Rowpress.

2.1 DRAM

DRAM is the main memory used in modern computers. DRAM capacities have increased substantially with shrinking process sizes from a few kilobytes in the 1990s to multiple gigabytes in personal computers and even terabytes in larger server systems today. The DRAM has a very high latency compared to the processor, and a single DRAM chip also has a limited bandwidth. Consequently, modern DRAM has a significant amount of parallelism to improve the bandwidth, positively influencing the average latency. DDR4 has 16 DRAM banks that operate in parallel, combined in a rank. There are single- and dual-rank DDR4 modules. Finally, the DRAM modules are connected to the processor via so-called channels. Today, dual-channel is the most widely used configuration in personal computers. Server CPUs typically have four or eight channels. Banks are divided into *rows* of *cells*, *i.e.*, the transistors and capacitors storing the data. As DRAM cell charges deplete over time, frequent refreshes are required to prevent data loss [18]. Typically, each cell is refreshed every 64 ms. Rowhammer mitigations like TRR can trigger additional row refreshes if they detect Rowhammer attacks. To access data, the memory controller has to activate the row, which moves the data from the row into the row buffer from where it can be read and written.

2.2 Rowhammer

In 2014, Kim et al. [21] discovered the *Rowhammer effect*. An attacker can induce bitflips in the DRAM from software through disturbance errors triggered by frequent row activations. Rowhammer attacks have been demonstrated in sandboxed environments [44, 34], in native environments [44, 10, 45], in virtual machines [53, 37, 16], in JavaScript [11, 3, 38], on mobile devices [48, 8, 23], over the network [46, 25], to influence or steal neural networks [55, 47, 36], and to steal cryptographic keys [24, 28, 7]. Rowhammer is getting worse with every new DRAM generation and increasing density. LPDDR4 requires less than a 10th of the activations than DDR3 memory to hammer [30, 20].

The Rowhammer effect induces a bitflip by discharging the capacitor in a neighboring cell. Once a cell has lost sufficient charge, the value read during the next load will no longer be the original value [21, 24, 50]. A a correlation exists between the data in the aggressor rows and the resultant bitflips; typically, the victim row adopts the values of the aggressor row.

There are different variants of Rowhammer: double-sided Rowhammer [44], single-sided Rowhammer [21], one-location Rowhammer [10] and half-double Rowhammer [23]. Double-sided Rowhammer uses two rows called aggressors in the same DRAM bank, which sandwich another row in between, inducing more bitflips than single-sided Rowhammer. Half-double Rowhammer hammers one row further apart than double-sided Rowhammer and is dependent on the additional refreshes performed by the TRR mitigation to induce bitflips.

Gruss et al. [10] introduced one-location Rowhammer intending to defeat the mitigations published at the time. The significant difference from the other Rowhammer variants is that one-location hammering only accesses a single location in the DRAM, which should not cause row conflicts. Instead, the aggressor is kept open through repeated accesses to the exact location until it gets closed by the memory controller. When the row is closed, one-location Rowhammer reopens the row immediately. Gruss et al. [10] hypothesized that this reopening might be the underlying trigger to the disturbance effect they discovered.

Rowpress. Rowpress was discovered by Luo et al. [26]. While it is similar to Rowhammer, it induces bitflips in rows through continuous and sustained activation of the same rows. Luo et al. [26] verified that bitflips are induced by holding the rows open for a long time using a specialized FPGA platform. As the memory controller of a CPU would normally instead close the row soon after the memory access, Luo et al. [26] use a unique access pattern with different offsets in the same row. This causes the memory controller to delay precharge commands up to a maximum of $9 \times t_{REFI}$. In a Rowpress attack, the attacker chooses offsets and repetitions such that the $9 \times t_{REFI}$ delay is approached. Luo et al. [26] find that the Rowpress read disturbance phenomenon is widespread across numerous DRAM modules and manufacturers. Their paper also analyzed and compared the flipped bits caused by Rowpress and the ones caused by Rowhammer. Their data shows that the flipped bits differ for both attacks; therefore, Luo et al. [26] conclude that Rowpress is caused by another effect than Rowhammer. This effect is either called *RAS clobber* [15, 52] or the *passing gate effect* [12].

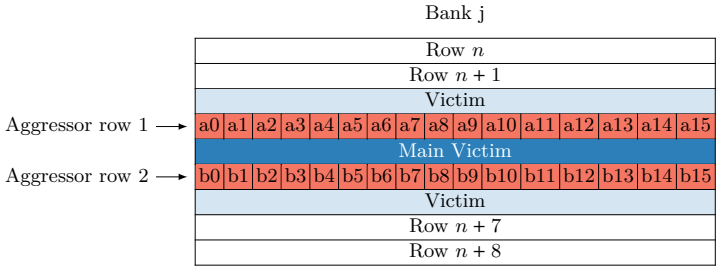


Fig. 2: Rowpress activates the 16 addresses in the first aggressor row (a0–a15) one after another. Then, all 16 addresses in the second aggressor row (b0–b15) are accessed. This keeps the rows open for a long period of time, causing bitflips due to the passing gate effect.

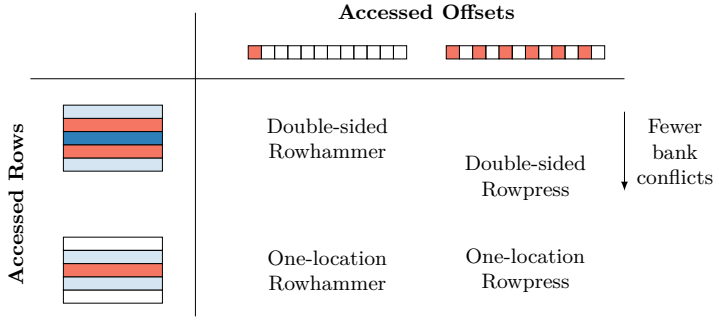


Fig. 3: Overview of the tested variants. Rowhammer attacks access only a single offset per DRAM row. Typically, this is offset 0. Rowpress accesses multiple offsets per DRAM row to keep the row open longer. The double-sided variants access two rows with the victim sandwiched between, however, with a lot lower frequency than double-sided Rowhammer. The one-location variants access only a single row, not explicitly causing bank conflicts.

3 Systematic Comparison of Rowhammer and Rowpress

The key difference separating one-location hammering from other Rowhammer methods is that it does not provoke bank conflicts with other rows in the same DRAM bank, *i.e.*, it is hammering only one location [10]. Similarly, Rowpress also works by keeping a row open as long as possible. With double-sided Rowpress, the open row is switched after some time, causing row conflicts but less than double-sided Rowhammer. One-location Rowpress keeps, like one-location Rowhammer, one row open. Figure 3 visualizes these similarities and differences.

Luo et al. [26] reported that different cells are affected by Rowpress than Rowhammer, hinting at another underlying cause. Rowpress exploits the passing gate effect [12] not the Rowhammer effect. Comparing the cells flipped by Rowpress, double- and one-location Rowhammer shows a less clear picture. While one-

location Rowhammer and double-sided Rowhammer flip roughly the same number of bits, the flip locations only overlap by 61.8 %. This hints that one-location Rowhammer can induce bitflip with both effects, Rowhammer and Rowpress.

While the Rowpress paper suggested that one-location Rowpress is possible, it is essential to note that the published Rowpress PoC only contains the double-sided Rowpress variant [26]. Therefore, we used the double-sided Rowpress PoC to develop a one-location Rowpress PoC which presses in only one single location.

3.1 Differences Between One-Location Rowhammer and Rowpress

There are multiple differences between the shown one-location Rowhammer by Gruss et al. [10] and Rowpress by Luo et al. [26]. Listings 1.1, 1.2, 1.3, and 1.4 show pseudo-codes of the different Rowhammer and Rowpress methods.

Number of Accessed Rows. One-location Rowhammer accesses only a single row in one DRAM bank. While Luo et al. [26] mention Rowpress with a single row "one-location Rowpress", their PoC code performs only double-sided Rowpress. We modify this double-sided Rowpress code to support one-location pressing.

Number of Cache Block Reads. One-location Rowhammer accesses only one cache block in a row repeatedly. Typically, it is the first offset in the row. Rowpress accesses multiple cache blocks in a row.

TRR Evasion. Gruss et al. [10] did not actively evade TRR in their one-location implementation. Luo et al. [26] employ a simple TRR evasion similar to TRRespass [9]. They synchronize their hammer loop to `REF` and access dummy rows on the same bank to decrease the chance that TRR tracks the correct aggressor rows. However, as we show in Section 3.6, we were not able to reproduce Rowpress with this simple TRR evasion method.

3.2 Comparison Methodology

The goal is to record the exact cells that flip when hammering or pressing the same physical range in memory. We run the following four Rowhammer and Rowpress methods to find out whether the same or different cells flip with these methods: one-location Rowhammer (OL RH), single-sided Rowpress from now on called one-location Rowpress (OL RP), double-sided Rowhammer (DS RH), double-sided Rowpress (DS RH).

To be able to compare the flipped cells we have to ensure that we always hammer the exact same physical memory range. We use a 1 GiB huge page configured with Linux kernel command-line parameters. From this huge page we use the first 100 MiB region for all our tests. We used multiple CPUs to hammer and press all our DIMMs: Intel Core i7-6700K, Intel Core i9-9900K, Intel Core i9-10900K. Our study comprises 12 different DRAM DIMMs without ECC. Like Luo et al. [26] we disabled the cache prefetcher for all experiments.

Our goal is not to find the best hammer method in terms of row accesses or time passed for the attack. We want to find all cells flippable with each method to be able to measure the overlap between methods as an indication of how

separated the effects are that they trigger. Therefore, we hammer our 100 MiB region multiple times with enough accesses per hammer loop to ensure we found the bulk of all flippable cells for each method. For each bitflip, we store the physical address, exact bit location, and data before and after the flip.

To find out whether one-location Rowpress flips different bits than one-location Rowhammer, we run the latter with every page offset that we also use for one-location Rowpress. We verify that the hammered offset within a row accessed does not influence the flips in the neighboring victim rows.

3.3 One-Location Rowhammer

```
1 for (iter = 0 ; iter < NUM_ITER ; iter++):
2   for (i = 0 ; i < NUM_AGGR_ACTS ; i++):
3     // access a single cache block of the aggressor row
4     // to keep the aggressor row open longer
5     for (j = 0 ; j < NUM_READS ; j++):
6       *AGGRESSOR[x];
7       clflushopt (AGGRESSOR[x]);
```

Listing 1.1: One-location Rowhammer code.

One-location Rowhammer repeatedly accesses one offset `x` in a single row. Varying this offset does not influence the flipped bits. We were only able to reproduce one-location Rowhammer on two identical DIMMs. They are two DDR4 modules with with an early TRR implementation.

3.4 Rowpress

We use the improved Rowpress algorithm published in the extended version [27] of Rowpress [26] in Appendix G. It interleaves the aggressor accesses and flushes. Luo et al. [27] measured four times as many induced bitflips than with the original version that first accessed all aggressors and then flushed all of them.

```
1 for (iter = 0 ; iter < NUM_ITER ; iter++):
2   for (i = 0 ; i < NUM_AGGR_ACTS ; i++):
3     // access multiple cache blocks of the aggressor row
4     // to keep the aggressor row open longer
5     for (j = 0 ; j < NUM_READS ; j++):
6       *AGGRESSOR[j];
7       clflushopt (AGGRESSOR[j]);
```

Listing 1.2: The one-location Rowpress code used for this evaluation

One-location Rowpress accesses only offsets within a single row, as shown in Listing 1.2 and Figure 4. We were only able to reproduce one-location Rowpress on two identical DIMMs, the same where we reproduced one-location Rowhammer.

```
1 for (iter = 0 ; iter < NUM_ITER ; iter++):
2   sync_with_ref();
3
4   for (i = 0 ; i < NUM_AGGR_ACTS ; i++):
5     // access multiple cache blocks in each aggressor row
6     // to keep the aggressor row open longer
7     for (j = 0 ; j < NUM_READS ; j++):
```

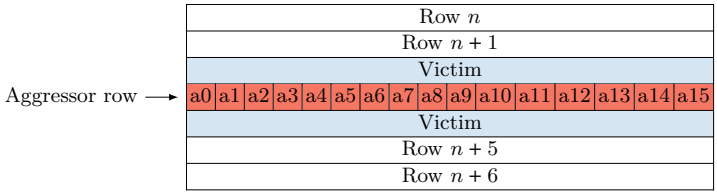


Fig. 4: Abstract representation of one-location Rowpress. We activate addresses a0 to a15 repeatedly to induce bitflips without provoking row conflicts.

```
8       *AGGRESSOR1[j];
9       clflushopt (AGGRESSOR1[j]);
10    for (j = 0 ; j < NUM_READS ; j++):
11      *AGGRESSOR2[j];
12      clflushopt (AGGRESSOR2[j]);
13
14  perform_dummy_row_accesses();
```

Listing 1.3: The double-sided Rowpress code used for this evaluation

The double-sided Rowpress code accesses offsets in two rows in the same bank. The code by Luo et al. [26] performs simple TRR evasion. They are syncing the row accesses to `REF` and access some dummy rows after the main row accesses, as shown in Listing 1.3. We found that this evasion is not enough for all but the same two DIMMs where we were able to reproduce the one-location methods.

3.5 Double-Sided Rowhammer

```
1 for (iter = 0 ; iter < NUM_ITER ; iter++):
2   for (i = 0 ; i < NUM_AGGR_ACTS ; i++):
3     // frequently open and close each aggressor row
4     for (j = 0 ; j < NUM_READS ; j++):
5       *AGGRESSOR1[0];
6       *AGGRESSOR2[0];
7       clflushopt (AGGRESSOR1[0]);
8       clflushopt (AGGRESSOR2[0]);
```

Listing 1.4: The double-sided Rowhammer code used for this evaluation

To hammer DRAM with the double-sided pattern, we used Blacksmith by Jattke et al. [17]. Blacksmith is the most sophisticated TRR evasion method to date. With Blacksmith we successfully hammered 6 of our 12 tested DIMMs.

3.6 Results

Table 1 shows an overview of all tested DIMMs and the successful hammer and press methods. We were only able to induce bitflips with Rowpress patterns on two of our tested DIMMs, D1 and D2. On DIMMs D3, D4, D6, and D9, Blacksmith [17] found double-sided Rowhammer patterns that induced bitflips. We did not try the one-location variants of Rowhammer and Rowpress if we did not achieve bitflips with both double-sided variants.

Table 1: The table shows whether a Rowhammer or Rowpress method was able to cause bitflips in any of our tested DIMMs. We indicate success with (✓), not causing bitflips with(✗) or not tested with (~). We did not test the one-location (OL) variants if double-sided Rowpress (DL RP) was unsuccessful.
* On D3 to D12, we used Blacksmith to hammer double-sided with TRR evasion.

	DIMM	Size	MT/s	Double Sided		One Location	
				RH*	RP	RH	RP
DDR4	D1	8 GB	2133	✓	✓	✓	✓
	D2	8 GB	2133	✓	✓	✓	✓
	D3	8 GB	2400	✓	✗	~	~
	D4	8 GB	2400	✓	✗	~	~
	D5	8 GB	2133	✗	✗	~	~
	D6	8 GB	2666	✓	✗	~	~
	D7	8 GB	2666	✗	✗	~	~
	D8	8 GB	2666	✗	✗	~	~
	D9	8 GB	2666	✓	✗	~	~
	D10	8 GB	2666	✗	✗	~	~
	D11	8 GB	3000	✗	✗	~	~
	D12	8 GB	2133	✗	✗	~	~

Our theory on why we only saw bitflips induced by the Rowpress pattern on 2 DIMMs is that TRR on modern DDR4 DIMMs has evolved to be more difficult to evade. Blacksmith can only do it by accessing a very large number of aggressor rows. For example, the best pattern on D3 repeatedly accesses 60 different rows during one refresh interval to evade TRR. However, as Rowpress tries to keep a row open for a long time, fewer row activations are possible per refresh interval. Luo et al. [26] caused Rowpress bitflips on a DIMM manufactured in 2018. Our DIMMs D1 & D2 are of similar age. Whether a blacksmith-like TRR evasion with Rowpress is possible on modern DDR4 memory, is up to future work.

Table 2 shows the detailed results of all DIMMs where we induced bitflips. On D1, we hammered the same physical address space with all methods listed. On D3, D4, D6, and D9, we used the best pattern found by Blacksmith.

On DIMM D1 and D2, we were able to flip bits with all tested methods. Most importantly, we induced bitflips in the same physical address range with all methods. This allows us to analyze precisely whether the tested Rowhammer and Rowpress methods flipped bits in the same cells on this DIMM. We also verified that the offset we access within a row does not influence the bitflips we see when hammering. Figure 5 shows a confusion matrix of all bitflips caused by Rowhammer and Rowpress. Because Rowpress with 8 accesses per row also causes bitflips with 16 offsets, we use its numbers for this analysis.

Based on our results it remains inconclusive, whether one-location Rowhammer [10] is, in fact, Rowpress. We measure roughly the same amount of bitflips when hammering with a double-sided and one-location pattern. However, only 61.8 % of bitflip locations found by one-location Rowhammer flipped also

Table 2: The number of bitflips per 100 MB of memory. RP-8 and RP-16 donate Rowpress patterns with 8 and 16 cache line accesses per row.
* On D3 to D11, we used Blacksmith to hammer double-sided for TRR evasion.

DIMM	Double Sided			One Location		
	RH*	RP-8	RP-16	RH	RP-8	RP-16
D1 & D2	15 612	2174	4264	14 264	161	0
D3	13 656	0	0	0	0	0
D4	15 786	0	0	0	0	0
D6	157	0	0	0	0	0
D9	3040	0	0	0	0	0

		RH		RP	
		DS	OL	DS	OL
RH	DS	100	56.5	13.7	1.0
	OL	61.8	100	14.5	1.1
RP	DS	98.1	95.3	100	1.0
	OL	96.3	98.8	81.4	100

Fig. 5: Detailed results of DIMM D1 & D2. Each number represents the fraction in percent of exact bitflip locations of one method (y-axis) found in the other method (x-axis). For example, 61.8 % of bitflip locations flipped by one-location Rowhammer is also in the set of locations flipped by double-sided Rowhammer.

with double-sided Rowhammer. The comparison between double-sided and one-location Rowpress also shows a reduction in the number of bitflips and a difference in bitflip locations. Comparing Rowpress and Rowhammer variants, we find that virtually all Rowpress bitflip locations are in the sets of both Rowhammer variants. However, this is likely also influenced by the smaller number of bitflips we found with all Rowpress variants. A possible explanation for our results could be that one-location Rowhammer causes both effects simultaneously very effectively, actually making it a *Presshammer* attack. It keeps a single row open, inducing bitflips due to RAS clobber in some cells, while the opening and closing of the row that is still happening is enough to induce bitflips in other cells.

4 Unprivileged One-Location Presshammer Attack

As our one-location Rowpress follows the same semantics as one-location Rowhammer, we can similarly reduce the requirements. Most Rowhammer attacks require at least partial knowledge of physical address bits above the page offset. Therefore, the OS hides physical address information from user programs to respond to these Rowhammer attacks [22]. While it is possible to reverse engineer

Fig. 6: The offsets within a DRAM row are parts of different memory pages `A` to `D`. Each part has the size of two cache lines, 128 bytes. This example is based on the DRAM addressing functions of a Skylake CPU [33].

the full DRAM addressing functions [1, 33, 51] and additionally uncover row addressing and physical address bits [23, 24, 43, 45] using the bank conflict timing side-channel, these methods can be tedious and take time.

For our one-location Presshammer attack, we only need the addresses of all offsets that map to the same row in the same bank. Due to the DRAM addressing functions, one page does not map to one row. On virtually all recent systems, different logical pages are interleaved on one row, as shown in Figure 6, based on the DRAM addressing functions [33], to increase memory throughput. Thus, the missing parts of each page are simply mapped to other rows in other banks. To perform a Rowpress attack, we must access multiple offsets within one row, spread across various pages. Instead of using the physical address information to find these offsets in the correct locations, we use the bank-conflict timing side-channel in a reduced form.

Taking a random page `A`, we want to find the pages `B` to `D` that map to the same row. Because we work on virtual memory and do not have any knowledge about the underlying mapping to the physical memory, close and "far away" pages based on the virtual address are not necessarily reflected in the physical addresses. We increase this chance by allocating a large chunk of memory. After filling single-page holes in the physical memory, the buddy allocator in Linux typically returns chunks of physically contiguous memory [23]. This increases the performance of our approach but is not strictly necessary.

Two assumptions for our algorithm hold on all current x86 systems:

A1 A DRAM row is 8 KiB (2^{13} B) large.

A2 The physical address bits used for row addressing start high enough to have a single page always in the same row across banks, *i.e.*, a page is never in two rows in the same bank.

The algorithm also works with 4 KiB rows with tiny modifications.

First, we have to find a page `X` that is in the same bank as `A`. Whether page `A` and `X` are in the same bank is easily detectable with the bank-conflict timing side channel. To not interfere with the following search for pages `B` to `D`, we start the search for page `X` a few hundred virtual pages away from page `A`. We denote current bank-conflict candidate page as `S`. We repeatedly flush and reload `A[0]` and `S[0]` while measuring the timing to detect a bank conflict. With page `A` and `X`, finding pages `B` to `D` is based on two requirements, also shown in Figure 7:

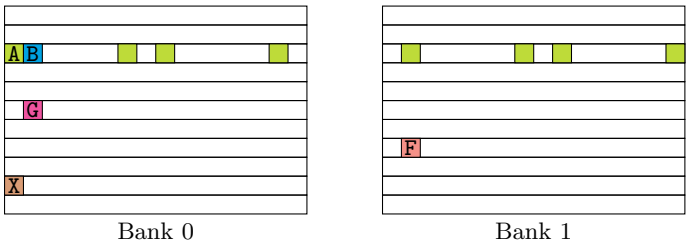


Fig. 7: Visualization of the two requirements to find page offsets in the same row. `F[1]` does not have a bank conflict with `X[0]`, violating **R1**. `G[1]` does have a bank conflict with `X[0]` but also with `A[0]`, violating **R2**. `B` fulfills both requirements.

R1 Because `B` to `D` must be in the same row as page `A`, they must be in the same bank as `X`. We search for pages that have a bank conflict (same bank, but a different row) with page `X`.

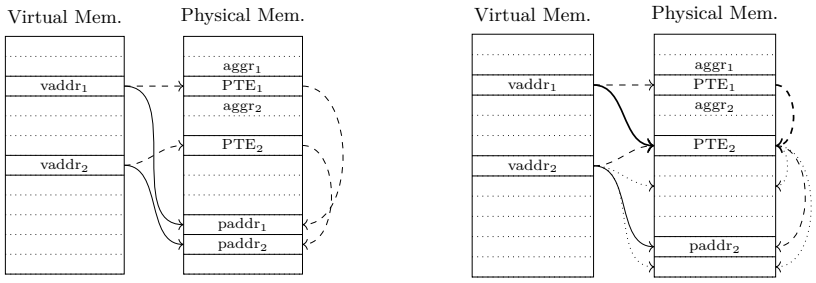
R2 Because `B` to `D` are in the same row as page `A`, there must not be a bank conflict between `B` to `D` and `A`.

We iterate over chunks of 64 B to find `B` to `D`, starting with an offset of 64. We check **R1** by flushing and reloading `S[offset]` and `X[0]`. We try pages above and below `A` until we find the page conflicting with `X`. Then, we check **R2** by flushing and reloading `S[offset]` and `A[0]`. If there is no conflict, we found the page mapped to the first offset. We repeat these steps until all offsets are filled. If we found `B` to `D` once, checking the following offsets is quick because we no longer have to search for new pages. During the search, we also mark already "used" offsets to speed up the search the more rows we already found.

Because we exclusively work with virtual memory, there is a chance that any of the required pages to fill a row is not mapped in our address space. However, as shown by Luo et al. [26], Rowpress does not need all offsets to press efficiently, and the offsets we find are sufficient to mount Rowpress. However, the rate of bitflips will be reduced as we do not control the data for the entire row, which is known to influence the number of bitflips [5, 24].

5 Proof-of-Concept Presshammer Attack on Page Tables

Luo et al. [26] did not present an end-to-end Rowpress exploit, as their focus was on the scientific evaluation of the effect, not the security impact. Consequently, they did not discuss a threat or attacker model and simply used 1 GiB huge pages to demonstrate the effect on real systems. We bridge this gap by showing that Rowpress can be used to attack a system without using 1 GiB or 2 MiB huge pages to get additional physical address information or any knowledge of the DRAM mapping functions.



(a) Correct Mapping (b) Successful Exploit

Fig. 8: A simplified example of page table flipping with only one level of page tables is shown. After successful exploitation (b) by hammering PTE_1, PTE_2 is writeable through $\texttt{vaddr}_1$. This makes the whole physical memory accessible by changing PTE_2 and then accessing the new page through $\texttt{vaddr}_2$.

Table 3: Results of our row finding algorithm using a different number of measurements for the bank-conflict side channel. There are 128 offsets per row. We search for 10 000 rows and discard the first 100. The search radius is $\pm$50 pages.

# Measurements	Duration per Row	Found Cache Lines	Correct Cache Lines
250	40 ms, σ = 17	124, σ = 4.9	119, σ = 14
500	45 ms, σ = 15	128, σ = 0.8	125, σ = 16
1000	116 ms, σ = 40	125, σ = 4.2	123, σ = 4.3
2500	260 ms, σ = 109	125, σ = 4.9	124, σ = 5.2

We use Presshammer to attack operating system page tables based on the exploit first shown by Seaborn et al. [44] and many following [5, 11, 17, 23, 48, 53, 57]. The proof-of-concept exploit of Seaborn et al. [44] still relied on full access to the virtual to physical address translations, while we use the algorithm from Section 4 to get all the information we need. The attack aims to use a bitflip to change a page frame number (PFN) in a page table to point to another page table. This gives the attacker write access to one of its own page tables and, through it, access to the whole memory. Figure 8 shows the mappings before and after a successful exploitation. The exploitation technique for Presshammer does, at large, not change from the Rowhammer exploit. We ran the exploit on an Intel Core i7-6700K with Ubuntu 20.04.4 LTS using Linux version 5.4.210.

DRAM Row Finding. We use the algorithm described in Section 4 to find the aggressor rows. On our machine with 16 GiB of memory, 4 GiB were used by the operating system, and the exploit allocates the remaining 12 GiB of memory. When allocating enough memory, the buddy allocator of the Linux kernel starts to return chunks of physically contiguous memory [48, 23].

A single measurement is insufficient to get a definitive result because of system noise when using the bank conflict side-channel. Software on other cores of

the CPU also accesses the memory. However, more measurements take longer, which is relevant if we want to check many rows for bitflips.

We ran the algorithm with 250 to 2500 measurements per address and analyzed the correctness of the result, which can be seen in Table 3. Taking 500 measurements gives the best results. We see that the standard deviation for the duration increases significantly for a higher number of measurements. This suggests more interrupts from the operating system that worsen the result.

Memory Templating. We verified that Rowpress bitflips are repeatable, like Rowhammer bitflips [21]. That means if an aggressor row flips a bit in a neighboring victim row, pressing this aggressor will flip the same bit again if the flip direction stays the same. This helps because we can template the memory in a first step, the exploit searches for aggressors that flip bits in exploitable locations. Bitflips are exploitable if they flip bits in the PFN range of a page table entry (PTE). We count bits 12 to 32 of a PTE (bits 0 to 20 of the PFN) as exploitable; this corresponds to a neighboring page up to a page 4 GiB away.

The exploit searches 10 aggressor victim pairs before continuing. This gives a higher exploit probability later because Rowpress can, like Rowhammer, only flip bits in one direction. Due to that, there is only a 50 % chance that an aggressor-victim pair works when filled with an actual page table entry.

The cache prefetcher in the CPU [14] tries to predict which addresses will be loaded next and fetches them into the cache for increased performance. Luo et al. [26] turned off the prefetcher for their experiments. However, this is not possible with our threat model. We evade the prefetcher by shuffling the offsets we access. This decreases the chance that the prefetcher can detect a pattern in our accesses.

Page Table Spraying. Filling most of the memory with page tables is the easiest way to increase the chances of successful exploitation in two ways. First, it increases the chance of putting a page table in one of the previously found victim rows. Second, it increases the chance of a changed PFN pointing to another page table. Van der Veen et al. [48] used a more deterministic technique to place the page table called *Phys Feng Shui*. They place a page table in a victim page by precisely massaging the Linux page allocator. However, for this Presshammer PoC exploit, we chose the more straightforward spraying method.

Page table spraying is performed by mapping the same shared memory file repeatedly. This puts the shared memory file's content into the memory only once, while having to create new page tables for every mapping. After finding the aggressor and victim pairs, the exploit still has most of the physical memory reserved from the templating phase. Every time before mapping the shared memory file again, the exploit frees exactly as many random pages as required for the newly to-be-created page tables. After creating around 80 % of all page tables, the exploit frees the victim rows one after another. When the page table spraying is finished, as much memory as possible is filled with page tables.

Page Table Flipping. After spraying the page tables, the exploit presses the aggressor rows one after another to induce bitflips in the victim PFNs. Because the victim rows are not mapped anymore, we cannot simply check for a bitflip. Instead, the exploit checks all mappings of the shared memory file if the content

is the expected content. If not, a PFN was changed. If the content looks like a page table, the exploit changes one page-table entry of the page table and checks through all mappings again. When finding another page with changed content, the exploit found $\texttt{vaddr}_1$ and $\texttt{vaddr}_2$, as shown in Figure 8b.

Exploitation. With all previous steps performed successfully, the exploit now has full read and write access to the whole physical address space. The original proof-of-concept exploit code from Seaborn et al. [44] used this access to patch a victim binary and change its behavior. For a real exploit, this could be any binary owned by root with the `setuid` sticky bit set to run code with superuser privileges. Without access to the virtual to physical address translations, the exploit has to scan the memory for the victim data it wants to attack. The exploit can also search for exploitable kernel structures to, e.g., give the process or user higher privileges or directly inject attacker-controlled code into the kernel. For our proof-of-concept exploit, we just dump the physical memory into a file.

While reading the whole physical memory, it is crucial to evict the TLB entry of the PTE modified by the exploit. If the exploit does not evict the TLB entry, all reads from $\texttt{vaddr}_2$ will return the same data. To evict it, we keep half of the mappings we used during the exploit and iterate over them. We free the other half to relieve the memory pressure on the system and prevent an out-of-memory situation. Dumping the memory to disk is limited by the disk write bandwidth.

We ran the exploit 5 times on our machine. Of these 5 runs, 3 were successful. It took, on average, 513 s (n=3, σ=125) to gain full access to the physical memory. One run was unsuccessful because the operating system did not put a page table in a victim row or no bit flipped in the PFN. The other run was unsuccessful because it did not point to another of our page tables after changing a PFN by row pressing it. If unsuccessful, the exploitation can just be tried again.

6 Discussion of Mitigations and Related Work

The large number of Rowhammer attacks published [3, 5, 7, 8, 10, 11, 16, 23–25, 28, 34, 36–38, 48, 45–47, 44, 53, 55] motivated research on mitigations for DRAM disturbance errors. While many have been proposed in the academic community [4, 13, 19, 29, 31, 32, 39–42, 49, 54, 56], only a few, like TRR [9], have found their way into practice. As Luo et al. [26] noted, Rowpress is also mitigated by TRR and other mitigations. However, they also note that similar to Rowhammer attacks bypassing TRR [9, 17, 23], Rowpress can also bypass it. And the same is true for many other proposed countermeasures. As CPUs cannot keep a row open longer than $9 \times t_{REFI}$, Rowpress always also causes a high number of row activations similar to Rowhammer. We show in this paper that this makes the two attacks hard to distinguish in practice, but this also means that we expect most countermeasures that are effective against current Rowhammer attacks to be also effective against Rowpress. In the worst case, with some additional tweaking to the detection threshold values. However, up to now, every deployed Rowhammer mitigation has been evaded eventually, and there is no guarantee that this will not be the case for Rowpress as well.

Our findings on the overlaps between Rowhammer and Rowpress effects indicate again [10] that mitigations must be principled not to be bypassable. That is, they cannot focus on the specific effect triggering the disturbance but must focus on preventing the effect of the disturbance. Mitigations that use this approach are, for instance, Error Correction Code (ECC) [5] and Chipkill [6]. ECC uses 8 redundancy bits per 64 bits (DDR4) or 32 bits (DDR5) of data. Chipkill can correct up to 8 bitflips, e.g., due to a chip failure [6]. However, ECC has already been bypassed with Rowhammer attacks, and Chipkill suffers from a significant overhead. As Qureshi [35] noted, it is essential to take an entirely new approach to mitigate Rowhammer and other upcoming DRAM disturbance effects. The earliest work in this direction is IVEC [13], which uses a hash instead of an error-correcting code and flips bits in corrupted memory until the hash matches again. With a similar approach, Saileshwar et al. [39] also propose a modification to ECC memory using a MAC instead of the error-correcting code to detect and correct a broader range of bitflips. Juffinger et al. [19] extended this approach into an error-correction system based on lightweight cryptography that effectively transforms DRAM disturbance errors from reliability issues into performance penalties, which, in the worst case, lead to a denial of service. Recently, Olgun et al. [29] introduced a Rowhammer mitigation based on a single per-row hardware counter to track row activations across all banks. This minimizes the hardware overhead while solving the issue of a small number of counters in TRR.

7 Conclusion

Many works analyzed DRAM disturbance effects and reported a wide range of prevalence estimations. In our set of 12 DDR4 modules, we can reproduce double-sided Rowhammer on 6 modules. All other Rowhammer and Rowpress variants only worked on 2 modules. For a fair comparison between the attack variants, we evaluated the number of unique bitflip locations for all variants, ranging from 161 to 15 612. However, 95.3 % to 98.8 % of Rowpress unique bitflip locations are also flipped by double-sided and one-location Rowhammer. This surprising result raises questions about the difference between the effects these attack implementations trigger and indicates that they might partially exploit the same root cause. However, this result also allows us to craft the first unprivileged end-to-end one-location Rowpress attack, which uses the same-row same-bank side channel to escalate to kernel privileges within less than 10 minutes. Our work shows that further research is necessary to better understand the differences between Rowhammer and Rowpress implementations on real systems and their implications for security.

Acknowledgments

We thank Lukas Gerlach, Stefan Gast, Patrick Rademacher, Sarah Rinderer and the reviewers for their feedback and help. 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), the Deutsche Forschungsgemeinschaft (DFG) (503876675) and partly funded by the European Union (ROF-SG20-3066-3-2-2). Additional funding was provided by generous gifts from Red Hat and Google. 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. Barenghi, A., Breveglieri, L., Izzo, N., Pelosi, G.: Software-only reverse engineering of physical DRAM mappings for rowhammer attacks. In: International Verification and Security Workshop (IVSW) (2018)
2. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: CRYPTO (1997)
3. Bosman, E., Razavi, K., Bos, H., Giuffrida, C.: Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In: S&P (2016)
4. Brasser, F., Davi, L., Gens, D., Liebchen, C., Sadeghi, A.R.: CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In: USENIX Security (2017)
5. Cojocar, L., Razavi, K., Giuffrida, C., Bos, H.: Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In: S&P (2019)
6. Dell, T.J.: A white paper on the benefits of chipkill-correct ecc for pc server main memory. Tech. rep., IBM Microelectronics (1997)
7. Fahr Jr, M., Kippen, H., Kwong, A., Dang, T., Lichtinger, J., Dachman-Soled, D., Genkin, D., Nelson, A., Perlner, R., Yerukhimovich, A., et al.: When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer. In: CCS (2022)
8. Frigo, P., Giuffrida, C., Bos, H., Razavi, K.: Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. In: S&P (2018)
9. Frigo, P., Vannacci, E., Hassan, H., van der Veen, V., Mutlu, O., Giuffrida, C., Bos, H., Razavi, K.: TRRespass: Exploiting the Many Sides of Target Row Refresh. In: S&P (2020)
10. Gruss, D., Lipp, M., Schwarz, M., Genkin, D., Juffinger, J., O'Connell, S., Schoechl, W., Yarom, Y.: Another Flip in the Wall of Rowhammer Defenses. In: S&P (2018)
11. Gruss, D., Maurice, C., Mangard, S.: Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA (2016)
12. Hong, S., Kim, D., Lee, J., Oh, R., Yoo, C., Hwang, S., Lee, J.: DSAC: Low-Cost Rowhammer Mitigation Using In-DRAM Stochastic and Approximate Counting Algorithm. arXiv preprint (2023)
13. Huang, R., Suh, G.E.: IVEC: Off-Chip Memory Integrity Protection for Both Security and Reliability. In: ISCA (2010)
14. Intel: Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture (2016)
15. Ito, Y., He, Y.: Apparatus and Methods for Refreshing Memory (2019), u.S. Patent 11062754B2
16. Jang, Y., Lee, J., Lee, S., Kim, T.: SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In: SysTEX (2017)
17. Jattke, P., van der Veen, V., Frigo, P., Gunter, S., Razavi, K.: BLACKSMITH: Rowhammering in the Frequency Domain. In: S&P (11 2021)

18. JEDEC Solid State Technology Association: Low Power Double Data Rate 4 (2017), http://www.jedec.org/standards-documents/docs/jesd209-4b
19. Juffinger, J., Lamster, L., Kogler, A., Eichlseder, M., Lipp, M., Gruss, D.: CSI: Rowhammer - Cryptographic Security and Integrity against Rowhammer. In: S&P (2023)
20. Kim, J.S., Patel, M., Yağlıkçı, A.G., Hassan, H., Azizi, R., Orosa, L., Mutlu, O.: Revisiting RowHammer: An Experimental Analysis of Modern DRAM Devices and Mitigation Techniques. In: ISCA (2020)
21. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: ISCA (2014)
22. Kirill A. Shutemov: Pagemap: Do Not Leak Physical Addresses to Non-Privileged Userspace (2015), https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=ab676b7d6fbf4b294bf198fb27ade5b0e865c7ce
23. Kogler, A., Juffinger, J., Qazi, S., Kim, Y., Lipp, M., Boichat, N., Shiu, E., Nissler, M., Gruss, D.: Half-Double: Hammering From the Next Row Over. In: USENIX Security (2022)
24. Kwong, A., Genkin, D., Gruss, D., Yarom, Y.: RAMBleed: Reading Bits in Memory Without Accessing Them. In: S&P (2020)
25. Lipp, M., Aga, M.T., Schwarz, M., Gruss, D., Maurice, C., Raab, L., Lamster, L.: Nethammer: Inducing Rowhammer Faults through Network Requests. In: SILM Workshop (2020)
26. Luo, H., Olgun, A., Yağlıkçı, A.G., Tuğrul, Y.C., Rhyner, S., Cavlak, M.B., Lindegger, J., Sadrosadati, M., Mutlu, O.: RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In: ISCA (2023)
27. Luo, H., Olgun, A., Yağlıkçı, A.G., Tuğrul, Y.C., Rhyner, S., Cavlak, M.B., Lindegger, J., Sadrosadati, M., Mutlu, O.: RowPress: Amplifying Read Disturbance in Modern DRAM Chips. arXiv:2306.17061 (2024)
28. Mus, K., Doröz, Y., Tol, M.C., Rahman, K., Sunar, B.: Jolt: Recovering TLS Signing Keys via Rowhammer Faults. In: S&P (2023)
29. Olgun, A., Tugrul, Y.C., Bostanci, N., Yuksel, I.E., Luo, H., Rhyner, S., Yaglikci, A.G., Oliveira, G.F., Mutlu, O.: ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation. In: USENIX Security (2024)
30. Orosa, L., Yaglikci, A.G., Luo, H., Olgun, A., Park, J., Hassan, H., Patel, M., Kim, J.S., Mutlu, O.: A Deeper Look into RowHammer's Sensitivities: Experimental Analysis of Real DRAM Chips and Implications on Future Attacks and Defenses. In: MICRO (2021)
31. Park, J.H., Kim, S.Y., Kim, D.Y., Kim, G., Park, J.W., Yoo, S., Lee, Y.W., Lee, M.J.: Row Hammer Reduction Using a Buried Insulator in a Buried Channel Array Transistor. IEEE Transactions on Electron Devices (2022)
32. Park, Y., Kwon, W., Lee, E., Ham, T.J., Ahn, J.H., Lee, J.W.: Graphene: Strong yet Lightweight Row Hammer Protection. In: MICRO (2020)
33. Pessl, P., Gruss, D., Maurice, C., Schwarz, M., Mangard, S.: DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security (2016)
34. Qiao, R., Seaborn, M.: A New Approach for Rowhammer Attacks. In: HOST (2016)
35. Qureshi, M.: Rethinking ECC in the Era of Row-Hammer. In: DRAMSec (2021)
36. Rakin, A.S., Chowdhuryy, M.H.I., Yao, F., Fan, D.: DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories. In: S&P (2022)
37. Razavi, K., Gras, B., Bosman, E., Preneel, B., Giuffrida, C., Bos, H.: Flip Feng Shui: Hammering a Needle in the Software Stack. In: USENIX Security (2016)

38. de Ridder, F., Frigo, P., Vannacci, E., Bos, H., Giuffrida, C., Razavi, K.: SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript. In: USENIX Security (2021)
39. Saileshwar, G., Nair, P.J., Ramrakhyani, P., Elsasser, W., Qureshi, M.K.: Synergy: Rethinking secure-memory design for error-correcting memories. In: HPCA (2018)
40. Saileshwar, G., Wang, B., Qureshi, M., Nair, P.J.: Randomized row-swap: mitigating Row Hammer by breaking spatial correlation between aggressor and victim rows. In: ASPLOS. pp. 1056–1069 (2022)
41. Saxena, A., Saileshwar, G., Juffinger, J., Kogler, A., Gruss, D., Qureshi, M.: PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks. In: DSN (2023)
42. Saxena, A., Saileshwar, G., Nair, P.J., Qureshi, M.: AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime. In: MICRO) (2022)
43. Schwarz, M., Gruss, D., Weiser, S., Maurice, C., Mangard, S.: Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA (2017)
44. Seaborn, M., Dullien, T.: Exploiting the DRAM Rowhammer bug to gain kernel privileges. In: Black Hat USA (2015)
45. Tatar, A., Giuffrida, C., Bos, H., Razavi, K.: Defeating software mitigations against rowhammer: a surgical precision hammer. In: RAID (2018)
46. Tatar, A., Krishnan, R., Athanasopoulos, E., Giuffrida, C., Bos, H., Razavi, K.: Throwhammer: Rowhammer Attacks over the Network and Defenses. In: USENIX ATC (2018)
47. Tol, M.C., Islam, S., Adiletta, A.J., Sunar, B., Zhang, Z.: Don't Knock! Rowhammer at the Backdoor of DNN Models. In: DSN (2023)
48. van der Veen, V., Fratantonio, Y., Lindorfer, M., Gruss, D., Maurice, C., Vigna, G., Bos, H., Razavi, K., Giuffrida, C.: Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In: CCS (2016)
49. van der Veen, V., Lindorfer, M., Fratantonio, Y., Pillai, H.P., Vigna, G., Kruegel, C., Bos, H., Razavi, K.: GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM. In: DIMVA (2018)
50. Walker, A.J., Lee, S., Beery, D.: On DRAM Rowhammer and the Physics of Insecurity. IEEE Transactions on Electron Devices (2021)
51. Wang, M., Zhang, Z., Cheng, Y., Nepal, S.: Dramdig: A Knowledge-assisted Tool to UncoverDRAM Address Mapping. In: Design Automation Conference (DAC) (2020)
52. Wolff, G.D.: Word line cache mode (2019), u.S. Patent 10366733B1
53. Xiao, Y., Zhang, X., Zhang, Y., Teodorescu, R.: One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: USENIX Security (2016)
54. Yaglikci, A.G., Patel, M., Kim, J.S., Azizi, R., Olgun, A., Orosa, L., Hassan, H., Park, J., Kanellopoulos, K., Shahroodi, T., Ghose, S., Mutlu, O.: BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows. In: HPCA (2021)
55. Yao, F., Rakin, A.S., Fan, D.: DeepHammer: Depleting the Intelligence of Deep Neural Networks through Targeted Chain of Bit Flips. In: USENIX Security (2020)
56. Yauglikcci, A.G., Olgun, A., Patel, M., Luo, H., Hassan, H., Orosa, L., Ergin, O., Mutlu, O.: HiRA: Hidden Row Activation for Reducing Refresh Latency of Off-the-Shelf DRAM Chips. In: MICRO (2022)
57. Zhang, Z., Cheng, Y., Liu, D., Nepal, S., Wang, Z., Yarom, Y.: PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses. In: MICRO (2020)