Memory considered harmful: The Rowhammer vulnerability in DRAM

Péter Bohner[1]

Karlsruher Institut für Technologie, 76131 Kalrsuhe, Germany

Abstract. The Rowhammer vulnerability is a fundamental issue of Dynamic Random Access Memory *(DRAM)*, whereby repeated access of a row of memory may corrupt adjacent parts of the memory. Despite being known for a decade, mitigations against it are still an active area of research and newly produced DRAM modules are still affected. In this paper, we will explain the vulnerability as well as and exploitation and mitigation strategies.

Keywords: DRAM · Hardware Security · RowHammer

1 Introduction

Dynamic Random Access Memory *(DRAM)*, has long served as a fundamental cornerstone of modern computing, providing fast and volatile data storage for a myriad of applications, most notably as the main memory of almost all computers. Preservation of memory integrity is paramount for reliable and secure operation, as both computer hardware and software assume that RAM is infallible. That is, the data read from memory is the same as the data written.

However, with ever increasing memory density, faults stemming from the fundamental construction of DRAM, called *disturbance errors*, have re-emerged[8]. These can be reliably triggered and thus exploited by the *Rowhammer* vulnerability, which was publicized in 2014[8][1]. Rowhammer leverages the repeated activation of rows in memory to induce disturbances manifesting in bit flips in adjacent rows. Because adjacent rows of memory may lie in different *security contexts* (different processes, sandboxes, virtual machines *(VM)*, etc.), Rowhammer allows an attacker to violate a system's security guarantees.

Rowhammer is far more than a theoretical vulnerability, with publicly available proof-of-concept exploits, including process privilege escalation and cross-VM side channels, having been demonstrated.[12][6][15]. It is especially dangerous, because exploiting the vulnerability only relies on accessing memory, which any code no matter how unprivileged must do. For example, reference [6] introduces a functional exploit from Javascript running inside a browser. Despite being known for almost a decade, research into this vulnerability is still ongoing[2] with new mitigations being proposed in 2023[11]. As the vulnerability has been

[1] The industry has been aware of it since at least 2012[8]

[2] primarily at ETH Zürich and TU Graz, see references [7],[10],[11][13]

present in memory produced since 2010[8] and and up to the present day[3][7], and such memory is used in countless devices, Rowhammer will likely remain relevant for years to come.

This work aims to explain the rowhammer vulnerability in an accessible manner and delves into exploitation and mitigation techniques.

2 DRAM Basics

This section provides a basic overview of the operating principle and structure of DRAM, which is sufficient to understand the rowhammer vulnerability. It is mainly adapted and summarized from [8].

2.1 Operation and Structure of Dynamic RAM

DRAM stores individual bits of data in a memory *cell* comprising of a *capacitor*, which can be charged or discharged (this is the state of the cell), and an *access-transistor*.[8] These cells are arranged in a two-dimensional grid, called *subarrays*[11]. Each horizontal wire (*wordline*) can be used to *activate* a row of this grid (by pulling it high), allowing each cell's charge to be transferred through the vertical wires (*bitlines*) to the corresponding bit in the *row-buffer*. This buffer is used to fulfill all accesses to the row until another row needs to be accessed (and therefore the current row needs to be closed), at which point the data in the buffer is written back to the row (as activation destroys the data in the cell) through the bitlines, and the wordline is pulled low.[15] Closing a row is also referred to as *precharging*. A group of subarrays sharing a row-buffer is known as a *bank*. Multiple banks are grouped together into a *rank*. A DRAM *module* consists of one or more ranks. The memory controller of a system can issue commands to each rank of memory in parallel, thus increasing parallelism by allowing multiple accesses to take place at the same time.[8] In most desktop and server computer architectures, the memory controller has multiple *channels* to each of which multiple Dual Inline Memory Modules *(DIMMs)* can be connected. Each of these channels are completely independent of each other, further multiplying memory throughput. [15]

It is important to note that the memory controller addresses RAM by *channel, rank, bank, row, and column*[7] and is unaware of the physical layout of rows and columns inside the dram chips (the subarrays and thereby the physical proximity in 3D-space of DRAM rows). The memory controller uses the DDR protocol (see 1) to communicate with each rank of memory. [11][7]

2.2 Refreshing

Real electrical components, including those from which DRAM is made, are not perfect. This means that DRAM cells leak charge over time. Therefore there

[3] DDR4 is still in production

Command	Addressing	Description
ACT	Bank, Row	activate (open) a row
READ	Bank, Column	read a column
WRITE	Bank, Column	write a column
PRE	Bank	Precharge (close) a row
REF	-	Refresh multiple rows

Table 1. The DDR protocol[8][11][7]

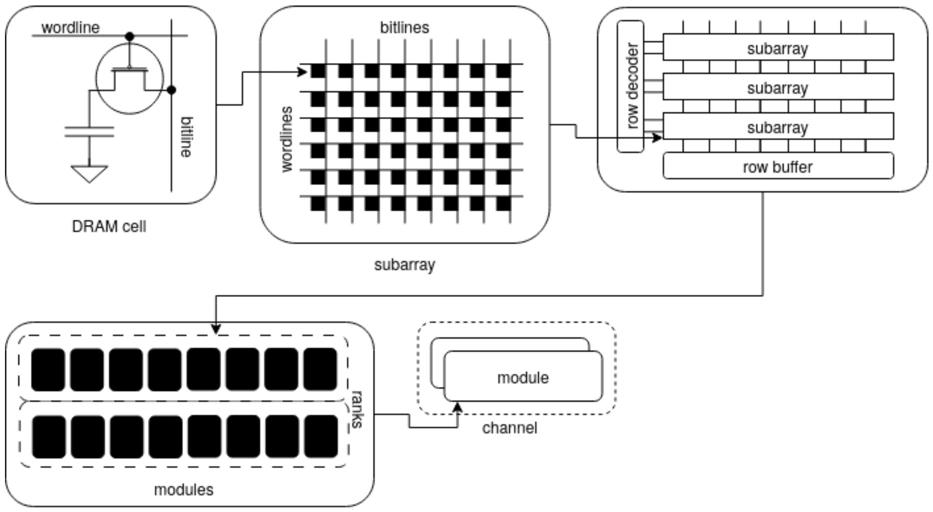

Fig. 1. Structure of DRAM, adapted from [8] and [7]

exists is a finite time, known as the *retention time*, before the capacitor voltage falls below a point where the states can be reliably differentiated (the so-called *noise-margin*), and data loss occurs. Each row needs to be periodically refreshed (activated and rewritten) in an interval shorter than the retention time. According to the JEDEC DDR3 specification, the retention time must be at least $64ms$.[8] There exists a separate REF command1, which refreshes several rows at once. It needs to be issued by the memory controller frequently enough that each row can be refreshed within the retention time. For DDR3 this requires 8192 REF commands, meaning one every $7.8\mu s$[8]

3 Disturbance Errors

An interference of a memory cell in another cell's operation is called a *disturbance*. If a disturbance is sufficiently large, the disturbed cell is pushed over it's noise margin, thereby experiencing a fault called a *disturbance error*.[8]. Disturbance errors have been known about since the first DRAM chips and have had hardware mitigations in place.[8][13]

As DRAM density has increased, three mechanisms have lead to memory cells being more susceptible to disturbance errors[8][15][5]: *Firstly,* each individual memory cell has to become smaller, which means each capacitor is smaller. The smaller a capacitor, the less charge it can hold, which reduces the cell's noise-margin, thereby increasing it's affinity for data loss. [8] *Secondly,* the distance between cells decreased, which increased the level of parasitic electrical interaction between cells, which can lead to data loss as well[15]. *Thirdly,* changes in process technology increased the number of cells susceptible to intercell crosstalk.[8]

4 The Rowhammer Bug

Yoongu, et al.[8] found that repeated toggling of a wordline (the *attacker*-row) may induce disturbance errors in memory cells located in nearby rows (the *victim*-row). [4] This can be achieved by alternating DRAM accesses to different rows of the same DRAM rank (*hammering*). To do this on a computer, a simple program can be written, which alternately reads from two memory addresses (X and Y) located on different rows, see figure 2 below. Because all modern processors have memory caches and out-of-order execution, this example needs to employ two tricks for this access pattern to occur. First, it uses the x86 *CLFLUSH* instruction to flush the caches after each write to both addresses. Second, it uses the *MFENCE* instruction to prevent the CPU from reordering memory accesses.[8]

```
loop:
mov (X), %eax
mov (Y), %ebx
clflush (X)
clflush (Y)
mfence
jmp loop
```

Fig. 2. x86 assembly Rowhammer example program from Yoongu, et al.[8]

Some cells are much more susceptible to hammering than others[13][8]. As tested by Yoongu et al [5], over 70% of cells that experienced a single error, had errors in 10 of 10 test iterations. This means, that there exist *victim cells* for rowhammer, which can be reliably attacked.[12][6]

[4] Crucially, access patterns not resulting in frequent toggling of wordlines did not cause disturbance errors. This is important for mitigation techniques.

[5] See section 7

[6] Interestingly, these are not correlated to *weak* dram cells, which are the first to lose data when not refreshed.[8]

5 Exploitation Challenges

Data loss caused by Rowhammer on its own may be undesirable, but is in itself not dangerous. It becomes a vulnerability because a program running in one security context (userspace/kernelspace, UNIX user/group, virtual machine, process sandbox, etc.) may affect memory used by another program in a different security context.[8][12], thereby violating that program's assumptions.

For example, the NaCl (native client) exploit demonstrated by [12], uses Rowhammer-induced bit-flips to alter previously validated (and deemed safe) program code. The Linux privilege escalation in the same paper uses Rowhammer to modify a processes' page table to gain write access to a SUID binary's code. These exploits are mostly built of well-known exploitation techniques (such as the latter exploit spraying kernel data structures (page tables)), using the new Rowhammer as the attack vector. Nevertheless, exploits face novel challenges described in the following sections.

5.1 Finding vulnerable addresses

Exploits need to determine memory addresses suitable for hammering, for which numerous layers of abstraction have to be reversed. The exploit needs to map memory it has access to, to the physical layout of the rows containing their data. Depending on the exploit type, this may be relatively easy, or exceptionally difficult. [14] While an exploit running on a Linux host need only use */proc/self/maps* to get page frame numbers, from which educated guesses about the physical memory layout can be made[12], a JavaScript exploit running in a browser can not even obtain the virtual addresses of it's own variables. Nevertheless, there exist multiple techniques to find vulnerable addresses from JavaScript, that have been used to develop full exploits.[14][6]

5.2 Bypassing Caches

To cause memory writes, the CPU caches must be bypassed. This can either be achieved like in 4 by directly flushing the cache, or by using an access pattern in memory that causes cache misses to the hammered addresses. The paper *Rowhammer.js*[6] goes into detail about such strategies.

6 Hammering techniques

Different ways of triggering rowhammer bit-flips have been developed over the years, including:

Random Hammering This is the basic basic variant of Rowhammer discussed in section 4, where a pair of random memory addresses located in different rows are alternatingly accessed.[8]

Double-Sided Hammering The first improvement to triggering Rowhammer comes from the first published practical exploit[12]: With double-sided hammering, instead of accessing two random rows, the row above and below the victim row are accessed alternatingly (these are the aggressor rows). This produces many more bit-flips than the random method, making the attack viable on more devices.[12] This technique does however require knowledge about the underlying memory geometry, to determine the suitable addresses.

Many-Sided Hammering The logic of double-sided hammering may be generalized to an arbitrary amount of adjacent addresses.[5]

HalfDouble HalfDouble is a hammering technique developed to bypass Rowhammer mitigation techniques which look for the aggressor-victim pairs described above. It uses four rows; the *far aggressor row*, located 2 rows away from the *victim row*, the *near aggressor row* located adjacent to the victim row and *the decoy row*, located away from the other rows. By alternatingly reading from the far aggressor row and the decoy row, with an infrequent (once every 1000 to 10000 iterations) read to the near aggressor row interspersed, bit-flips can be achieved in the victim row[9]. This technique massively complicates Rowhammer mitigations, as it disproves the assumption made by early research[8][12][6] that bit-flips may only be introduced by adjacent rows.

Blacksmith Blacksmith is an even more advanced hammering technique specifically designed to bypass TRR7.4 mechanisms/heuristics, by creating complex varying access patterns. It does this by mapping the signal properties of *phase*, *amplitude*, and *frequency* to memory accesses then fuzzing these values. The phase describes when the address is accessed (in relation to the others), the amplitude describes the amount of consecutive activations and the frequency describes the distribution of accesses over time. [7] With these access patterns, a significant number of bit-flips can be reliably introduced into DDR4 and LPDDR4X memory modules, which already have hardware-level mitigations against Rowhammer in place.[7][13]

7 Mitigations

Mitigations of Rowhammer can broadly be divided into three categories depending their place of implementation; purely software based, memory controller based, and in-DRAM approaches. The following describes and evaluates some of the most well-known mitigations in these categories.

Disabling cache flushes Looking at the code in 2, one may assume that making cache flushes a privileged instruction would mitigate the rowhammer vulnerability. Unfortunately, this is not sufficient, as multiple CLFLUSH-free attacks have been demonstrated[13], including ones made from interpreted languages (e.g., JavaScript: *Rowhammer.js*)[6]. Rowhammer has also been demonstrated on ARM devices[16], where cache flushes are privileged. Restricting cache flush instructions has however made exploitation harder by reducing the number of bit-flips that can be achieved.[6] Therefore, it *may* be a useful mitigation, being comparable to restricting the *RDTSC*(read time stamp counter) instruction following *Meltdown* and *Spectre*.[7]

Shortening refresh intervals The simplest change to reduce the frequency of Rowhammer errors is to increase the refresh rate, as this is usually configurable by firmware. This was also the first mitigation to be implemented on devices, via BIOS updates[12].

The issue with this mitigation arises from the fact that sufficient shortening of the refresh rate incurs a very large performance overhead. To eliminate rowhammer bit-flips in DDR3, a refresh rate of as low as $8.2ms$ was necessary, a 7.8 fold increase in refreshes. This would result in the memory spending 11.0–35.0% of its time refreshing (up from the current 1.4–4.5%)[8]. A $> 30\%$ performance penalty is of course unacceptable. This was with basic single-sided hammering. With double-sided hammering[12] and modern DRAM being much more vulnerable to disturbance errors[13][11], it is likely that far shorter refresh rates would be necessary to fully protect modern memory from rowhammer, if this were the only mitigation implemented. Therefore, modern systems generally use the standard $64ms$ refresh rate or have halved the refresh rate to $32ms$, and mostly rely on other mitigations.

7.1 Error Correction Code (ECC) Memory

Rowhammer is far from the first source of memory corruption. Bit-flips occur (although rarely) during normal DRAM operation. [8]. To guard against this, mission-critical workstation and server applications have long used Error Correction Code (*ECC*) memory. ECC memory modules contain additional bank(s) of memory to store redundancy (error correction) information.[4] ECC memory should prevent rowhammer attacks from working, as bit flips can be corrected or at least detected. While ECC memory is effective in preventing almost all published exploits, it can be defeated, see 8. ECC memory also has other weaknesses, which lead to it being seldom used in consumer devices. Most importantly, storing redundancy information incurs a sizeable storage overhead, resulting in higher costs as more physical DRAM chips are needed for the same capacity. [10][4][8] Most ECC implementations reside in the memory controller, which needs to explicitly support it. This support is lacking from Intel's consumer CPUs. The parity calculations of ECC also result in a slight increase in latency and power consumption[4], which is undesirable in power-constrained mobile devices.

[7] RDTSC is helpful for developing Rowhammer attacks as well[15][14]
[8] They can be caused by EMI, high temperatures and even cosmic rays[4]

7.2 ANVIL

Rowhammer attacks have peculiar memory access characteristics, notably a high cache miss rate and high spatial locality of DRAM row accesses.[2] ANVIL is a software based mitigation in form of a Linux kernel module that uses hardware performance counters found in modern CPUs to identify rowhammer attacks and preemptively issue memory reads to potential victim rows. It achieves this by monitoring the last-level cache miss count and sampling memory accesses with these counters. [2][16] In the paper that proposes it, ANVIL is effective in mitigating Rowhammer, but there is little other data on it, aside of concerns with implementing ANVIL to guard against DMA-based[9] Rowhammer attacks on ARM[16]. Judging from the working principle of ANVIl, it seems highly likely that ANVIL would fail to protect against HalfDouble and Blacksmith.

7.3 Probabilistic Adjacent Row Activation (PARA)

Probabilistic Adjacent Row Activation (*PARA*) is the first low-overhead mitigation of Rowhammer. It was proposed by *Yoongu et al* in the same paper that discovered Rowhammer[8]. PARA is a mitigation performed by the memory controller, wherein upon closing a row (PRE command), the controller sometimes (with a low probability) opens one of the adjacent rows of the closed row. With this, it is unlikely that a victim row may be subject to sufficient hammering to cause bit-flips, without being refreshed. The paper showed that a probability of 0.1% for opening a row is sufficient, which resulted in a worst-case performance overhead across multiple benchmarks of $< 0.8\%$.[8]

PARA's other advantage is that it is stateless, requiring no additional metadata to be stored. And unlike TRR implementations discussed later, it can not be defeated by adversarial attack patterns, because there is no attack detection algorithm to trick. Unfortunately, PARA has a lot of downsides and is not a sufficient mitigation against Rowhammer. First, it requires the memory controller to know the physical layout of the memory to determine what rows are physically *adjacent* to each other. With most memory this is however not possible, as the geometry of the DRAM subarrays is hidden from the memory controller.[13] Second, PARA only protects from hammering coming from direct neighbours. New attacks such as *HalfDouble*[9] (see 6) have shown that Rowhammer attacks with a *blast-radius* of > 1 [11] are possible.

7.4 Targeted Row Refresh (TRR)

Targeted Row Refresh (*TRR*) is an umbrella term for all techniques that selectively refresh rows that are considered to be by some metric to likely be under a rowhammer attack. This includes memory controller based implementations such as Intel's *pTRR*, of which very little is known[5][13], but which suffers from the same issues lacking necessary information about the physical DRAM layout

[9] Direct Memory Access

as PARA implementations, and is therefore not fully able to protect against Rowhammer[5].

In-DRAM TRR was introduced into the DDR4 specification and is the "official" solution to Rowhammer[2][5] It consits of two parts: 1. The maximum activation count (MAC) is a value reported by the module to the memory controller, which sets the maximum number of times a row may be activated in a refresh interval. 2. The sampler, a part inside the module, which determines and keeps track of potential aggressor rows. If the MAC is exceeded for a row, the memory controller must issue activations to adjacent rows.[13][5] The implementation of TRR is DRAM manufacturer specific and little was known about the inner workings of this mechanism until the publication of TRRespass[5]. That paper reverse-engineered the secret algorithms used by DRAM multiple vendors and constructed multi-sided hammering patterns that reliably defeated these TRR implementations.

Therefore, even though TRR is the primary solution adopted by the industry to fix Rowhammer, current implementations have proven to insufficient to protect against Rowhammer attacks.

8 Defeating ECC

ECCploit is an exploit capable of inducing bit-flips in ECC protected systems. In modern systems ECC is generally handled at the memory controller level, which uses of the DRAM to store additional error correcting information, to be able to detect and correct bit-flips. One can think of storing k bits of data and r redundancy bits in a code-word of size $n = k + r$. The error correcting algorithm can then reconstruct the valid data word, even if the code-word has some errors.[4] The redundancy depends on the algorithm used, but all have a maximum amount of errors they can reliably detect and correct. For example, *Extended Hamming Codes* (commonly taught in computer science) are *Single Error Correcting, Double Error Detecting* (SECDEC), meaning any single bit-flip can be corrected and two bit-flips may be detected. Three or more bit-flips in a code-word may not be noticed.[4]

The key for exploiting Rowhammer in an ECC system is to cause bit-flips in the victim row, such that it remains a valid ECC codeword. This is not feasible to exploit by random-chance. To achieve this, the authors reverse-engineered the proprietary ECC algorithms of several CPUs and found a novel side-channel to determine if a bit had been flipped and corrected: When an error correction occurs, the access latency is higher. Using this side-channel, Rowhammer-vulnerable bits can be identified through exhaustive hammering of the available address space. Combined with the now known ECC algorithm, ECCploit can find a set of bit-flips that result in a valid code word when triggered.[4]

ECCploit is not a fast attack, requiring half an hour when the bit-flips can be directly observed or a week otherwise. Nevertheless, it has proven that ECC alone is not sufficient to fully protect a system from Rowhammer.[4]

9 Impact

After almost a decade since the discovery of Rowhammer, research into it is still ongoing, with new attacks and mitigations having been published recently[13]. Therefore, Rowhammer is still a relevant class of vulnerability. There are only four CVEs (see table 2) directly associated with Rowhammer attacks, only one of which is a full software exploit (*CVE-2022-42961*), even though many more attacks have been published in academia[6][14][12][13][16].[10] Despite proof-of-concept exploits being available for a lot of different platforms and targets [6][16][12], there are no reports of any malware exploiting rowhammer in the wild.[3][11] This is in stark contrast to other high-profile hardware vulnerabilities, like Spectre and Meltdown, where malware development was quick and widespread.[1]

All in all, Rowhammer is a fundamental flaw in a ubiquitous technology, which has proven to be exceptionally difficult to comprehensively mitigate, which is why it is still a great threat to information security.

CVE Designation	CVSS-3.1 Score	Description
CVE-2015-0565	10.0 critical	NaCl allows clflush instruction [12]
CVE-2020-10255	9.0 critical	TRRespass, see **??**
CVE-2021-42114	8.3 high	BLACKSMITH, see 6
CVE-2022-42961	5.3 medium	ECDSA key disclosure from wolfSSL

Table 2. CVEs related to Rowhammer[3]

References

1. Armasu, L.: Hundreds of meltdown, spectre malware samples found in the wild. https://www.tomshardware.com/news/meltdown-spectre-malware-found-fortinet,36439.html (2018), [Online; Accessed on 29.12.2023]
2. Aweke, Z.B., Yitbarek, S.F., Qiao, R., Das, R., Hicks, M., Oren, Y., Austin, T.: Anvil: Software-based protection against next-generation rowhammer attacks. In: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems. p. 743–755. ASPLOS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2872362.2872390, https://doi.org/10.1145/2872362.2872390
3. Byers, R., Turner, C., Brewer, T.: National vulnerability database. Tech. rep., National Institute of Standards and Technology (2024). https://doi.org/10.18434/M3436

[10] This is one of the reasons why the CVE system is frequently criticised as a metric for problem severity

[11] The author's search did not turn up any results for such malware

4. Cojocar, L., Razavi, K., Giuffrida, C., Bos, H.: Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In: S&P (May 2019), Paper=https://download.vusec.net/papers/eccploit_sp19.pdfSlides=https://www.ieee-security.org/TC/SP2019/SP19-Slides-pdfs/Lucian_Cojocar_Exploiting_Correcting_Codes_slides-ecc-new.pdfWeb=https://www.vusec.net/projects/eccploitPress=https://bit.ly/2UcucNv, best Practical Paper Award, Pwnie Award Nomination for Most Innovative Research
5. 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 (May 2020), Paper=https://download.vusec.net/papers/trrespass_sp20.pdfSlides=https://download.vusec.net/slides/trrespass_sp20.pdfWeb=https://www.vusec.net/projects/trrespassCode=https://github.com/vusec/trrespassPress=https://bit.ly/2UXWKJ4, best Paper Award, Pwnie Award for Most Innovative Research, IEEE Micro Top Picks Honorable Mention, DCSR Paper Award
6. Gruss, D., Maurice, C., Mangard, S.: Rowhammer.js: A remote software-induced fault attack in javascript. In: Caballero, J., Zurutuza, U., Rodríguez, R.J. (eds.) Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 300–321. Springer International Publishing, Cham (2016)
7. Jattke, P., Van Der Veen, V., Frigo, P., Gunter, S., Razavi, K.: Blacksmith: Scalable rowhammering in the frequency domain. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 716–734 (2022). https://doi.org/10.1109/SP46214.2022.9833772
8. 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: 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA). pp. 361–372 (2014). https://doi.org/10.1109/ISCA.2014.6853210
9. 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: Butler, K.R.B., Thomas, K. (eds.) 31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022. pp. 3807–3824. USENIX Association (2022), https://www.usenix.org/conference/usenixsecurity22/presentation/kogler-half-double
10. Marazzi, M., Jattke, P., Solt, F., Razavi, K.: Protrr: Principled yet optimal in-dram target row refresh. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 735–753 (2022). https://doi.org/10.1109/SP46214.2022.9833664
11. Marazzi, M., Solt, F., Jattke, P., Takashi, K., Razavi, K.: Rega: Scalable rowhammer mitigation with refresh-generating activations. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1684–1701 (2023). https://doi.org/10.1109/SP46215.2023.10179327
12. Mark Seaborn, T.D.: Exploiting the dram rowhammer bug to gain kernel privileges. https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html (2015), [Online; Accessed on 01.12.2023]
13. Mutlu, O., Olgun, A., Yağlıkcı, A.G.: Fundamentally understanding and solving rowhammer. p. 461–468. ASPDAC '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3566097.3568350, https://doi.org/10.1145/3566097.3568350
14. Oren, Y., Kemerlis, V.P., Sethumadhavan, S., Keromytis, A.D.: The spy in the sandbox: Practical cache attacks in javascript and their implications. In: Pro-

ceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. p. 1406–1418. CCS '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813708, https://doi.org/10.1145/2810103.2813708

15. Pessl, P., Gruss, D., Maurice, C., Schwarz, M., Mangard, S.: Drama: Exploiting dram addressing for cross-cpu attacks. p. 565–581. SEC'16, USENIX Association, USA (2016)
16. van der Veen, V., Lindorfer, M., Fratantonio, Y., Padmanabha Pillai, H., Vigna, G., Kruegel, C., Bos, H., Razavi, K.: Guardion: Practical mitigation of dma-based rowhammer attacks on arm. In: Giuffrida, C., Bardin, S., Blanc, G. (eds.) Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 92–113. Springer International Publishing, Cham (2018)