Martin Heckel

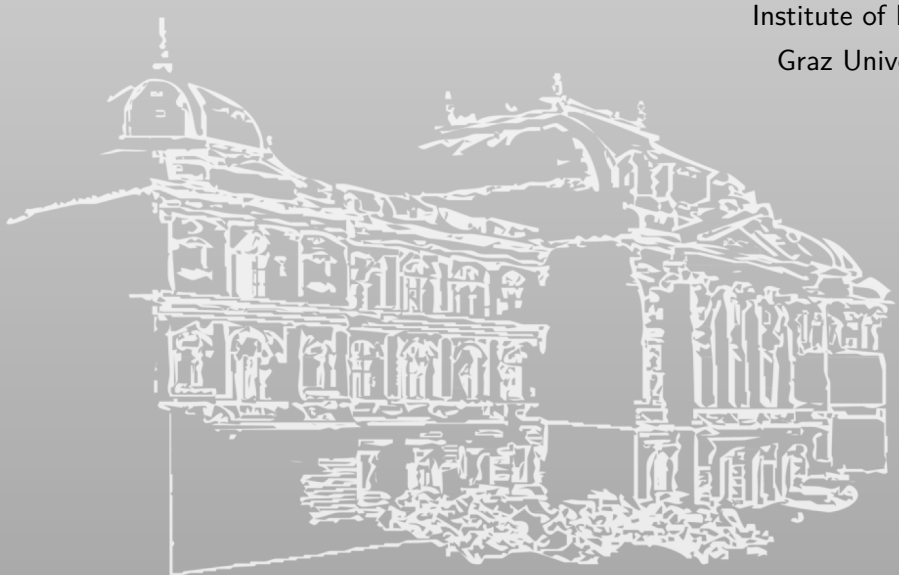
Real-World Rowhammer: Understanding and Addressing the Challenges to Rowhammer Attacks

PhD Thesis

Assessors: Florian Adamsky, Daniel Gruss, Kaveh Razavi

April 2026

Institute of Information Security
Graz University of Technology



SCIENCE ▪ PASSION ▪ TECHNOLOGY

Hochschule Hof
University of Applied Sciences

TU Graz

Abstract

The integration density of Dynamic Random-Access Memory (DRAM) increases with each generation, enabling greater storage capacity. As a result, the integration density in current DRAM is so high that frequent access to DRAM cells can cause disturbance errors in adjacent cells. This effect, known as *Rowhammer*, was initially considered a side effect, exploitable only in theory. Later, it was shown that Rowhammer can be exploited in various ways to achieve local privilege escalation.

For effective Rowhammer attacks, an attacker typically needs to know where specific addresses are spatially located in DRAM to perform accesses in a specific pattern. Only a few variants of Rowhammer do not use specific patterns. The mapping between addresses and spatial locations in DRAM is determined by the memory controller using DRAM addressing functions. In current systems, the memory controller is part of the CPU. Since vendors do not publish the DRAM addressing functions used in their memory controllers, reverse-engineering of these functions is required to run effective Rowhammer attacks. Besides Rowhammer attacks, these functions could also be used by the operating system to optimize DRAM accesses or to implement Rowhammer mitigations.

In this thesis, we present the first software-only approach for reverse-engineering DRAM bank addressing functions on systems with AMD CPUs. After reverse-engineering DRAM addressing functions, it remains unclear whether they are correct, as the process can be error-prone. To address this problem, we present an approach to verify a set of DRAM bank addressing functions and determine for individual functions within the set whether they are correct or not.

After the first Rowhammer exploits were published, vendors began developing mitigations. First, BIOS updates that reduce the DDR3 refresh window from 64 ms to 32 ms were deployed. With reduced refresh windows, cells have less time to leak, and an attacker has fewer accesses to trigger disturbance errors. We show that the number of bit flips identified within a given time can be increased by a factor of 830 when performing a multi-threaded Rowhammer attack and running additional memory accesses in parallel. These amplifications can also affect systems with the double-refresh-rate mitigation, making them vulnerable to Rowhammer.

Rowhammer is a significant problem that affects a majority of systems. While the number of Rowhammer-related publications increases annually, there is no known case of Rowhammer being used in "real" attacks, e.g., malware campaigns. Also, the prevalence of Rowhammer outside of research lab setups is unknown. We analyze multiple experimental evaluations from Rowhammer-related publications and identify six threats to the validity of Rowhammer research. Additionally, we provide seven recommendations to improve the real-world applicability of future Rowhammer research. To answer the question about Rowhammer prevalence, we conduct a large-scale study evaluating 1 006 datasets collected across 822 systems. We show that at least 12.5 % of datasets are susceptible to Rowhammer attacks. We also discuss reasons why other datasets were not affected and identify the failure of reverse-engineering DRAM addressing functions and the lack of tools supporting systems with AMD CPUs and DDR5 DRAM as the main reasons.

This thesis consists of two parts. In the first part, we contextualize our contributions within the state of the art. The second part contains my unmodified[1] first-author publications. All of these publications were peer-reviewed and accepted at international scientific conferences.

[1]The content is unmodified from the camera-ready version of the papers. The format was adjusted to fit the layout of this thesis.

Acknowledgements

First and foremost, I want to thank my advisor, Florian Adamsky. You gave me the freedom to research the things I am interested in. Whatever problems I had to overcome, you always supported me in doing so. You employed student workers to support my research. You did not require me to hold lectures, but you permitted me to do so if I wanted. I had all the freedoms I could have wished for (if there were any limits, they were set by the administration and their sometimes weird processes).

I also want to thank my other advisor, Daniel Gruss. Even though I worked primarily remotely, I received a lot of support, and you always took the time to discuss experimental results or research ideas. You always gave me advice on which ideas might be worth following and which might not be, from a strategic perspective. Even though you advised me, you never forbade or forced me to go in a specific direction.

Initially, there was no ongoing research on Rowhammer at Hof, so my advisors submitted a project proposal, which was luckily accepted. Otherwise, I would not have been able to do my PhD under such good conditions. We had many discussions about research ideas and follow-up experiments when something did not work out as expected. I especially want to thank both advisors for making the operation mode of being employed in Hof and doing my PhD in Graz possible. To conclude, I could not have wished for better advisors.

I also want to thank Kaveh Razavi for assessing my PhD thesis and Denis Kalkofen for being the chair of the defence.

I want to thank Jonas Juffinger. You taught me many things about efficient writing (and shortening) of papers and the effective presentation of results. I also want to thank Stefan Gast for the many great discussions we had, especially during my time in Graz.

I want to thank my other colleagues in Graz, Roland Czerny, Theresa Dachauer, Carina Fiedler, Simone Franza, Lukas Giner, Lena Heimberger, Andreas Kogler, Lukas Lamster, Lukas Maar, Sudheendra Raghav Neela, Fabian Rauscher, Martin Unterguggenberger, and Hannes Weissteiner. It was great working with you, discussing research-related or other topics, or complaining about weird administrative rules, unhelpful comments in reviews, or not-so-useful lectures. I also want to thank my colleagues in

Hof, Sebastian Pahl and Katharina Schiller, for the many discussions we had in Hof. I really enjoyed giving the IT Security lecture together with Katharina Schiller. I also want to thank Cookie, Carina's dog. She always lightened the mood when ideas did not work out or when we got rejection notifications for our papers.

I want to thank the people I met at conferences, especially Emiliia Gelóczi, Guillaume Didier, and Eddie Billoir. We had many great discussions, and I am happy to have met you.

Finally, I want to thank my parents, family, and friends for supporting me throughout my PhD.

Contents

Part I.

Real-World Rowhammer: Understanding and Addressing the Challenges to Rowhammer Attacks

1

Introduction

Modern computer systems are typically constructed in multiple layers. Thereby, as for all systems consisting of multiple layers, the security of the entire system is only as strong as the weakest layer. Security estimates or guarantees for components are typically done individually, since components can be used in conjunction with different other components. Therefore, these security estimates or guarantees assume that all components in the layers below function as expected, and they hold only as long as these components actually behave as assumed. Consequently, a vulnerability in the operating system would affect all applications running on it, even if the applications themselves do not have any vulnerabilities.

In contrast to directly exploitable vulnerabilities, side-channel attacks exploit metadata leakage to derive secrets. Examples of such metadata include power consumption [110, 117, 84, 67, 104, 77], execution time [82, 51, 62, 114], or memory accesses that lead to data being cached [112, 111, 25, 102, 93]. Therefore, an attack might be possible even though there are no vulnerabilities in the individual components, but only because specific components leak metadata. Side-channel attacks can affect both hardware and software components. Among others, side channels can be used to reverse-engineer DRAM bank addressing functions [82, 22, 103, 32, 46], which can be used in Rowhammer attacks.

Rowhammer is a fault attack which, in contrast to side-channel attacks, do not leak metadata to derive secret information. Instead, fault attacks trigger faults, impacting the availability and integrity of systems. Rowhammer exploits disturbance errors in DRAM [57], leading to bit flips in memory locations not mapped to the process controlled by the attacker. Initially, Rowhammer was considered a theoretical problem, as random bit flips seemed difficult to exploit. This changed when Seaborn and Dullien [92] published two Rowhammer exploits and demonstrated how Rowhammer can lead to an NaCl sandbox escape and local privilege escalation.

Since then, a cat-and-mouse game has begun between Rowhammer mitigations and novel attacks that bypass them. As a first reaction, vendors reduced the refresh windows for DDR3 DRAM from 64 ms to 32 ms through BIOS updates, as suggested by Kim et al. [57]. For example, the mitigation published by Lenovo [64, 63] doubles the refresh rate, as discussed in Chapter 6. Other proprietary Rowhammer mitigations, such as target row refresh (TRR) and pseudo target row refresh (pTRR), followed. Currently, Rowhammer attacks have been demonstrated on different DRAM generations like DDR3 [57, 10, 85, 107, 28, 27, 94, 95, 50, 60, 98, 55, 30], DDR4 [28, 43, 1, 27, 13, 66, 22, 81, 15, 45, 58, 98, 80, 108, 24, 69, 53, 71, 55, 46, 16, 30, 44, 4, 86], DDR5 [46, 75], LPDDR2 [100, 66], LPDDR3 [100, 115], LPDDR4 [100, 58], and LPDDR4X [22, 45, 58]. In addition to main memory, Rowhammer can also affect GPU memory, as shown by Lin et al. [65] on GPUs with GDDR6 memory.

1.1. Main Contributions

This section introduces the first-authored peer-reviewed papers included in this PhD thesis. I first-authored 5 papers of which one was published at an A* venue, and two at tier A venues. All of these publications are in the fields of Rowhammer and DRAM addressing functions.

Since most Rowhammer attacks need DRAM addressing functions to work properly, these functions have to be reverse-engineered first. We introduce an approach to reverse-engineer DRAM bank addressing functions on systems with AMD CPUs [32]. After reverse-engineering, it is not clear whether the functions are correct, since the time-based side channel used by most reverse-engineering tools [82, 22, 103, 32, 46] is error-prone. We introduce an approach for verifying reverse-engineered DRAM bank addressing functions [33] in software, which can verify individual functions from a submitted set of functions.

After the first Rowhammer exploits [92] were published, vendors began deploying mitigations against Rowhammer. One of these mitigations was the double refresh rate mitigation, which reduced the refresh window on DDR3 from 64 ms to 32 ms. We introduce a Rowhammer amplification approach for DDR3, achieving an amplification factor of 830 when multi-threaded Rowhammer and additional noise are used [31]. Our approach can bypass the double refresh rate mitigation.

As shown in prior work [57, 100, 45, 58, 69, 24, 46], the majority of systems are affected by Rowhammer. However, no real-world exploits, e.g., in malware campaigns, have used Rowhammer until now to the best of our knowledge. We analyze the stark discrepancy between academic publications and real-world impact by performing an analysis of threats to Rowhammer research validity [36]. Finally, we run a large-scale study on Rowhammer prevalence [35].

Reverse-Engineering Bank Addressing Functions on AMD CPUs [32]: Our approach is based on the time-based reverse-engineering of DRAM bank addressing functions by Pessl et al. [82]. We adjust their approach by adding automated detection of the number of banks and the block size (e.g., the number of consecutive addresses within the same bank). We also modify the grouping and demonstrate that our approach is capable not only of identifying DRAM bank addressing functions on systems with Intel CPUs but also on those with AMD CPUs. We evaluate our approach on two systems with Intel CPUs (i7-4800MQ and i9-10900K) and two systems with AMD CPUs (Ryzen 9 3900X and Ryzen 9 5950X), each in one configuration with a single DIMM and one with two DIMMs. We compare the results of our approach with those from Pessl et al. [82] and show that both approaches yield very similar addressing functions on Intel-based systems. This work was published at the Workshop on DRAM Security (DRAMSec) in 2023 [32] in collaboration with Florian Adamsky.

Flipper: Rowhammer on Steroids [31]: We demonstrate that two basic approaches can significantly increase the number of bit flips identified by double-sided Rowhammer on DDR3. First, we perform Rowhammer multi-threaded, with each thread exclusively hammering one DRAM bank. Second, we run another thread that performs memory accesses to increase the system's memory pressure. To generate memory pressure, we use the `cmbsp` instruction. We show that combining both approaches can increase the number of bit flips identified within a given time by a factor of 830. With our approach, bit flips occur faster, and some bit flips occur only when applying the amplification approach. Therefore, attacks can be performed faster, and (if they require bit flips at specific offsets) may become possible. Additionally, we show that our amplification approach can help to bypass the mitigation of a double refresh rate frequently used on DDR3 systems. This work was published at the Microarchitecture Security Conference (uASC) in 2025 [31] in collaboration with Florian Adamsky.

Verifying DRAM Addressing in Software [33]: When using DRAM addressing function reverse-engineering, it is often not possible to tell if the reverse-engineered addressing functions are correct. In contrast to Pessl et al. [82], who use a high-bandwidth oscilloscope to verify their addressing functions, our approach leverages the time-based side channel between row hits and row conflicts for verification. We group addresses using the functions under test and randomly select 5 000 addresses (a_1) for each group. For each of these addresses, we randomly select another address from the same group (a_2) and another address from a different group (b_1). Next, we alternately access the addresses and verify that a_1 and a_2 have the timing of a row conflict and a_1 and b_1 have the timing of a row hit. If both are correct, we consider the address a_1 as *verified*. Finally, we use the fraction of *verified* out of all addresses as a measure of how correct the set of addressing functions is. Since removing a single addressing function results in one bit of the bank number missing, 50 % of the banks cannot be addressed anymore. Therefore, the fraction of *verified* addresses is also reduced to 50 %. We use this approach to verify single DRAM bank addressing functions by removing one function at a time and verifying that the fraction of *verified* addresses drops to 50 % of the baseline. Additionally, we identify *rank addressing functions*, which are linear combinations of DRAM bank addressing functions. We reimplement the covert channel based on the timing differences between row hits and row conflicts, initially introduced by Pessl et al. [82], and evaluate it on DDR3, DDR4, and DDR5. We measure true capacities of up to 2.23 Mbit/s on DDR3, up to 0.66 Mbit/s on DDR4, and up to 1.39 Mbit/s on DDR5. We utilize the timing differences between row hits and row conflicts to perform a website fingerprinting attack, in which an attacker measures access times to addresses in different DRAM banks. These traces are then used in combination with machine learning to identify one out of 100 websites in a closed-world scenario. We achieve an F_1 score of 84 % on DDR4 and an F_1 score of 74 % on DDR5. This work was published at the European Symposium on Research in Computer Security (ESORICS) in 2025 [33] in collaboration with Florian Adamsky, Jonas Juffinger, Fabian Rauscher, and Daniel Gruss.

Epistemology of Rowhammer Attacks: Threats to Rowhammer Research Validity [36]: As shown in prior work [57, 100, 45, 58, 69, 24, 46], Rowhammer is a significant problem that affects the majority of systems or DIMMs tested. However, so far, no known case of Rowhammer being used in real-world attacks, e.g., malware campaigns, has been reported. We analyze the stark discrepancy between academic attention and

real-world relevance by systematically examining 32 offensive Rowhammer publications, which collectively conduct a total of 48 experimental evaluations. We identify 6 threats to the validity of Rowhammer research and derive 7 recommendations for future Rowhammer-related research. This work was published at the European Symposium on Research in Computer Security (ESORICS) in 2025 [36] in collaboration with Hannes Weissteiner, Florian Adamsky, and Daniel Gruss.

FlippyR.AM: A Large-Scale Study of Rowhammer Prevalence [35]: We design, perform, and evaluate a large-scale study on real-world Rowhammer prevalence. Therefore, we create an ISO image containing multiple existing Rowhammer implementations, which are executed on the system on which the ISO image is booted. We gave various talks, among others, at the 38C3 [34], and distributed our thumb drives to the audience afterwards. In total, we collected 1 006 datasets from 822 unique systems and show that 12.5 % of all systems are affected by our fully automated Rowhammer tests. Since automated reverse-engineering of addressing functions fails on approximately 50 % of datasets, we assume the actual number of affected systems to be higher, and our results to provide a lower bound. Additionally, we were unable to add the more recent *ZenHammer* [46] tool at that time, which is the first Rowhammer tool to yield good results on systems with AMD CPUs. Therefore, more or better tools and better automation could result in a higher percentage of affected systems. This work was published at the Network and Distributed System Security (NDSS) Symposium in 2026 [35] in collaboration with Nima Sayadi, Jonas Juffinger, Carina Fiedler, Daniel Gruss, and Florian Adamsky.

1.2. Other Contributions

This section introduces the peer-reviewed papers I co-authored during my PhD studies. In total, I co-authored 2 papers, one of which was published at an A* venue. Both of them are in the field of Rowhammer.

Presshammer: Rowhammer and Rowpress without Physical Address Information [53]: In Presshammer, the difference between the single-sided and double-sided variants of Rowhammer [57] and Rowpress [69] is analyzed. Therefore, we implemented a novel variant of one-location Rowpress. We perform our experimental evaluation on a set of 12 DDR4 DIMMs. This work was published at the Conference on

Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA) in 2024 [53] in collaboration with Jonas Juffinger, Sudheendra Raghav Neela, Lukas Schwarz, Florian Adamsky, and Daniel Gruss.

Memory Band-Aid: A Principled Rowhammer Defense-in-Depth [19]: Memory Band-Aid reduces the memory bandwidth per thread and bank when a Rowhammer attack is detected. Thereby, the accesses of the hammering thread get slower, and the number of accesses might become insufficient to trigger bit flips. Since current hardware does not support per-bank bandwidth limits, but only limits for an entire logical CPU core, these limits are used in our experimental evaluation. It is shown that the overhead of using Memory Band-Aid across multiple Phoronix macro benchmarks ranges from 0 % to 9.4 %. Assuming hardware support for per-thread limits, Memory Band-Aid would be a practically usable second line of defense against Rowhammer. This work was published at the Network and Distributed System Security (NDSS) Symposium in 2026 [19] in collaboration with Carina Fiedler, Jonas Juffinger, Sudheendra Raghav Neela, Hannes Weissteiner, Abdullah Giray Yağlıkçı, Florian Adamsky, and Daniel Gruss.

1.3. Outline

Chapter 2 provides background information on DRAM and Rowhammer. Chapter 3 provides an overview of the state of the art in the area of Rowhammer attacks and mitigations. Chapter 4 concludes Part I. In Part II, a list of all first-authored and co-authored papers is shown. Additionally, the camera-ready versions of the main contributions of this thesis are provided in Chapters 5 to 9.

2 Background

This chapter describes the general concept of DRAM, DRAM architecture, and DRAM addressing functions. Additionally, it contains information on DRAM disturbance effects, such as Rowhammer.

2.1. DRAM

DRAM is the main memory technology used for volatile random-access memory (RAM) in current computer systems. In this context, *dynamic* means that the cells, consisting of capacitors and transistors, need to be actively refreshed periodically. Refreshing the DRAM cells is necessary because the capacitors lose charge over time. So, after some time, all capacitors would be discharged, and the information stored in the DRAM cells would be lost. The DDR standards specify how often cells are refreshed, and therefore, how long a single cell must maintain a charge level that is clearly detectable as *charged* or *uncharged*. In DDR3 [47] and DDR4 [48], the default refresh window is 64 ms, while it is 32 ms for DDR5 [49].

2.1.1. DRAM Architecture

DRAM cells are organized into arrays consisting of rows and columns. All cells within the same column are connected to a *bitline* [56] in a way that the data can be read from and written to all cells in that column using a single bitline. Similarly, all cells within the same row are connected to a *wordline* [56] in a way that only entire rows can be activated. When a row is activated, the transistors connect the bitlines to the capacitors, and thereby, the state of the capacitor can be read, or the capacitor can be charged or discharged to write a specific value. There are *true-cells*, in which the charged state represents a logical 1 and *anti-cells*, in which the

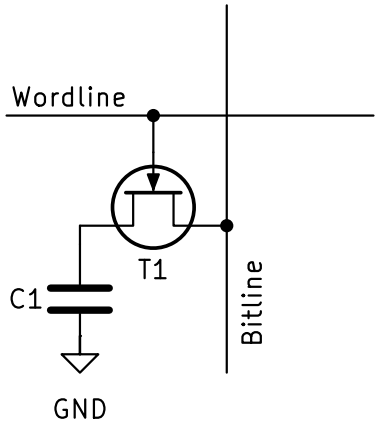


Figure 2.1.: Single DRAM cell consisting of a capacitor (C_1) and a transistor (T_1). The transistor T_1 is connected to the wordline (horizontal) and the bitline (vertical). When the wordline is activated, the value of the cell can be read by measuring the voltage level on the bitline. To write a value, the voltage level corresponding to the value that should be written is applied to the bitline when the wordline is activated.

charged state represents a logical 0 [57]. See Figure 2.1 for the general schematic of a DRAM cell.

The capacitors have very small capacities, and *sense amplifiers* are used to amplify the voltage on the *bitlines*. A DRAM array consists of multiple *subarrays*, each comprising DRAM cells organized in rows and columns and connected to wordlines and bitlines. Multiple subarrays are connected via sense amplifiers [113]. In addition the sense amplifiers and DRAM cells, the bitlines are also connected to a *row buffer* that stores the contents of the currently activated row. Since reading is a destructive operation, the value of the row buffer has to be written back to the DRAM array before another row can be activated. Physically, the row buffer is typically implemented as a feedback loop of the sense amplifiers [72].

Initially, the row buffer is empty, and the bitlines are *precharged*. This means the bitline voltage is half the operation voltage of the DRAM array. When a row is *activated*, the voltage of the bitlines is either slightly increased in case of a charged capacitor discharging to the bitline or slightly decreased in case of an empty capacitor charging from the bitline. Technically, the sense amplifiers compare the voltage on one bitline connected to a DRAM cell with that of another precharged bitline [72]. Following

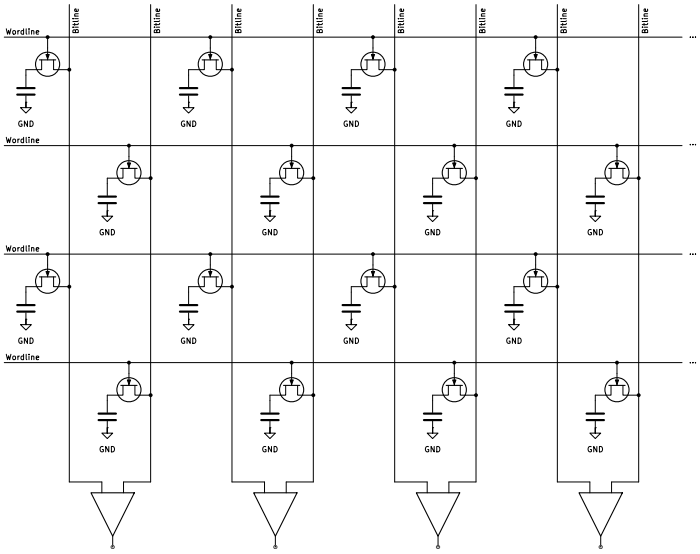


Figure 2.2.: Simplified architecture of a DRAM array: Each sense amplifier is connected to two bitlines, which are connected to cells from different wordlines. Therefore, the voltage level of the bitline connected to the activated cell is compared with that of the other, precharged bitline. Following this, the difference is amplified and fed back into the bitline connected to the activated cell via a feedback loop (not shown in the figure).

a slight increase in the voltage of the bitline connected to the activated DRAM cell due to a charged capacitor, the sense amplifiers amplify the difference, finally resulting in the bitline having the full operation voltage. In the opposite case (the voltage of the bitline connected to the activated DRAM cell is slightly decreased due to an empty capacitor), this difference is amplified as well, resulting in the bitline having a voltage of 0 V. See Figure 2.2 for an overview of the combination of DRAM cells and sense amplifiers.

Data can only be accessed from the row buffer after a row has been activated and its data has been loaded into the row buffer. Typically, data is transferred between DRAM and the CPU in chunks of 8 B, which is equal to the DRAM bus bandwidth of 64 bit. A burst of 8 of these chunks form a *cache line* with a size of 64 B, which is the size of continuous data stored in the last-level cache of the CPU, e.g., only whole cache lines (bursts of eight 8 B chunks) are transferred. Depending on whether a read

or write operation is performed, cache lines can be sent to or received from the CPU. When the operation is complete, the row can be closed, effectively writing the the row buffer content back to the DRAM array and then deactivating the wordline. Next, the DRAM is precharged, e.g., the bitline voltage is set to half the operating voltage. After that, another row can be activated as described above.

When an access is performed to a row that is currently active, the row is already in the row buffer and can be served directly from it. This case is called a *row hit*. In contrast, there are also cases in which a row is accessed that is currently not activated. In these cases, the currently activated row must be closed to restore the content of the row buffer to the DRAM array, and a `PRECHARGE` command must be issued to bring the bitlines into the precharged state. In modern DRAM, data is already written back during the read/write operation, and the bitlines only have to be precharged. After that, the other row must be activated, and subsequently, the data can be served from the row buffer. This case is called a *row conflict*. A row conflict is slower than a row hit due to the additional commands that must be executed before the access can be performed. On a system with an Intel Core i7-12700H, row hits take approximately 400 cycles, and row conflicts take approximately 570 cycles, as measured using the measurement loop used in our tool for reverse-engineering DRAM bank addressing functions on systems with AMD CPUs [32].

Regarding the closing of rows, there are multiple scheduling policies. An *open page policy* [41] keeps DRAM rows open as long as possible, e.g., closes the row only when another row is accessed. In contrast, a *closed page policy* [41] closes the row as soon as possible after the access is completed, e.g., trying to keep the DRAM in the precharged state. There are also policies like First-Ready First-Come-First-Served (FR-FCFS) [87] that combine both approaches. The FR-FCFS scheduler optimizes accesses for row hits by executing accesses that lead to a row hit out of order before other accesses.

On systems with many logical CPU cores and, therefore, many processes running in parallel, the probability of consecutive DRAM accesses targeting the same row is relatively low. Hence, a closed page policy is typically preferred on these systems. In contrast, on systems with fewer logical CPU cores, consecutive accesses may go to the same DRAM row, e.g., for sequential read accesses from the same process. On these systems, an open page policy is often preferred.

The combination of the DRAM array, sense amplifiers, and row buffer (e.g., feedback loop), as shown in Figure 2.2, is called a DRAM *bank*. These banks are stored on DRAM chips, with banks distributed across multiple chips, and each chip holding parts of multiple banks. The banks are grouped in one or multiple *ranks*. Beginning with DDR4, banks are not directly organized in ranks, but in *bank groups*, which are organized in ranks.

A *Dual Inline Memory Module (DIMM)*, typically inserted into the memory slots on the mainboard, contains one or multiple ranks and additional components, such as the serial presence detected (SPD) EEPROM. There are also systems in which the DRAM chips are directly soldered to the mainboard, or which support a combination of soldered DRAM chips and DIMMs that can be inserted. The DIMMs are connected to the CPU with buses called *channels*. Multiple DIMMs can be connected to the same channel; however, only one DIMM per channel can be used simultaneously. It is possible to connect multiple DIMMs to different channels, allowing one DIMM per channel to be used simultaneously. On DDR5, a single DIMM has two independent channels, called *subchannels*, each with a bandwidth of 32 bit.

Since multiple DRAM channels can be used simultaneously, the memory controller tries to distribute sequential accesses to multiple channels. As a result, sequential cache lines can be read in parallel, thereby increasing the effective memory bandwidth for sequential accesses [82]. For arbitrary accesses, the probability of a collision is $\frac{1}{n_{Ch}}$, where n_{Ch} is the number of populated channels on the system.

2.1.2. DRAM Addressing

On current x86 systems, the physical address space is divided into *pages*, which are mapped to the virtual address space of the processes. These pages are continuous memory blocks, each with a typical size of 4 KiB, on x86. Therefore, the same physical page can be mapped to multiple processes, as is the case with shared libraries. To store the mapping of physical pages to virtual addresses for different processes, *page tables* are used. When a process accesses a virtual address, the kernel looks up the virtual address in the *Translation Lookaside Buffer (TLB)*, which caches recently accessed mappings. In the case of a TLB hit, the mapping is stored in the TLB and can be used directly. If the mapping is not found in

the TLB (TLB miss), the page tables are used to get the physical address mapped to the virtual address that was accessed. If the address could be resolved using page tables, this is stored in the TLB. If an access is made to a virtual address to which no physical page is mapped, the page table walk fails, triggering a segmentation fault (the kernel sends `SIGSEGV` to the process). The translation between virtual and physical pages is handled by the CPU, while the operating system maintains the mapping between virtual pages and physical addresses.

Page tables utilize multiple layers of mappings to achieve fast lookups while minimizing memory consumption. Therefore, the kernel keeps a pointer to a *Page Global Directory* (`PGD`) for each process. We consider 4-level paging [39]. The `PGD` contains 512 entries, each with flags and a pointer to a *Page Level 4 Directory* (`P4D`). These entries might also be `NULL`, e.g., there do not have to be 512 `P4Ds`, but only as many as are currently used. Each `P4D` contains 512 entries, each with flags and a pointer to a *Page Upper Directory* (`PUD`). Similar to the `PGD`, not all entries have to be populated. Each `PUD` contains 512 entries, each with flags and a pointer to a *Page Middle Directory* (`PMD`). As for the previous directories, not every entry of the `PMD` has to be populated. Each `PMD` contains 512 entries, each with flags and a pointer to a *Page Table Entry* (`PTE`). Again, not all entries in the `PMD` have to be populated. In contrast to the previous data structures, a `PTE` does not contain multiple entries, but only flags and the address of a single *Page Frame Number* (`PFN`) mapped to a virtual address.

For a 4 layer address translation using page tables, a 64-bit virtual address is split in multiple parts [39]: The first 16 bit of the virtual address are the sign extension and not directly used for the resolution of the mapping using page tables. They are set to the same value as the next bits, so a 64-bit virtual address starts either with 17 bits with a value of one or 17 bits with a value of zero when 4-level paging is used. The following blocks of 9 bit are used as offsets in the `PGD`, `P4D`, `PUD`, and `PMD`. The last 12 bit are used as an offset within the 4 KiB page. Therefore, the last 12 bit of a virtual address and a physical address are identical. Figure 2.3 provides a graphical overview. There is also 5-level paging, which uses 57 bit of the virtual address [38].

There are also hugepages with sizes of 2 MiB and 1 GiB, which is due to the multi-level page table architecture: For 2 MiB hugepages, the entries in the `PUD` do not point to a `PMD` but directly to a `PTE`. This `PTE` contains a `PFN` for a continuous block of memory, addressable with $12\,\text{bit} + 9\,\text{bit} = 21\,\text{bit}$,

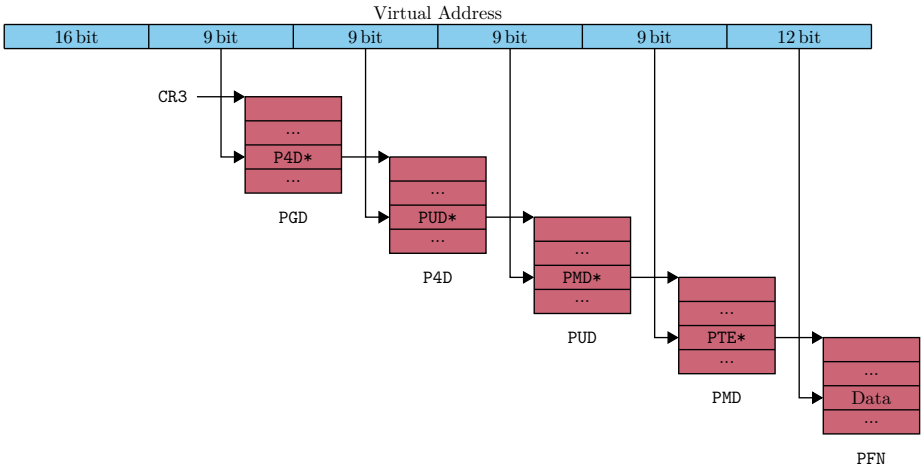


Figure 2.3.: Page tables with 4 layers: `PGD`, `P4D`, `PUD`, and `PMD`. Entries in the `PMD` contain a pointer to a `PTE`, which stores the `PFN` mapped to a virtual address, along with additional data such as flags and permissions. Fields marked with * are pointers to the corresponding data structures, e.g., *P4D** is a pointer to a `P4D` data structure.

which is equal to 2 MiB. Similarly, 1 GiB hugepages skip the `PMD` and `PUD`, so the `P4D` stores pointers to `PTEs`. These `PTEs` contain `PFNs` to blocks of continuous memory with 12 bit + 9 bit + 9 bit = 30 bit, which is equal to 1 GiB. The `PS` flag determines whether the next level of paging or a hugepage is used. It is set to 1 when no additional levels of page tables are used [39]. See Figure 2.4 for a graphical representation of hugepages.

Processes can request mapping of physical memory to virtual addresses via syscalls. Hugepages have to be requested explicitly from the Linux kernel using the hugetlb interface [96]. In contrast, the kernel can also map hugepages transparently, e.g., without the process directly recognizing if a hugepage is mapped or not [97].

The memory controller is typically part of the CPU and maps physical addresses to locations in DRAM. Therefore, *DRAM addressing functions* are used. For current CPUs, these addressing functions are not publicly disclosed and must be reverse-engineered. AMD published the DRAM addressing functions in the *BIOS and Kernel Developer's Guide (BKDG)* up to microarchitecture `16h` [2]. Beginning with microarchitecture `17h`

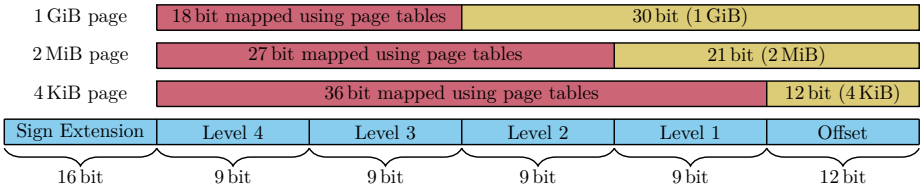


Figure 2.4.: Graphical representation of the different splits between page table-based mapping (depicted in red) and direct mapping (depicted in yellow) in a 64 bit virtual address.

released in 2017, no BKDG was released. Therefore, the DRAM addressing functions on newer AMD CPUs are no longer publicly available.

Rows and columns are typically directly mapped in a way that n bits of the physical address map to n bits of the row or column. In contrast, functions addressing DRAM banks typically use multiple (≥ 2) bits of the physical address that map to 1 result bit. To reduce ≥ 2 input bits to 1 output bit, these functions typically use an XOR operation over all input bits. Effectively, the output bit is 1 when an odd number of input bits is set, and 0 when an even number of input bits is set. Multiple of these bit masks can be combined, e.g., to address 8 banks, 3 bit masks (each returning 1 bit) are required. These functions are called *linear* addressing functions. Figure 2.5 illustrates the addressing functions of an Intel Haswell CPU with one 4 GiB DIMM.

Since the functions that use a combination of multiple bits as input can only address components that are a power of two, they are only used on systems where all elements are a power of two. For example, they can only be used to address banks on systems with 8, 16, or 32 banks, but not on systems with 12 or 24 banks. On such systems, *nonlinear* addressing functions are used, and reverse-engineering them remains an open research question for DRAM today. However, Gerlach et al. [23] reverse-engineered nonlinear addressing functions of caches. This type of nonlinear addressing function should not be confused with the nonlinear addressing functions reverse-engineered by Jattke et al. [46] on systems with AMD CPUs. The latter are essentially linear DRAM bank addressing functions that account for offsets in the physical address space used by AMD.

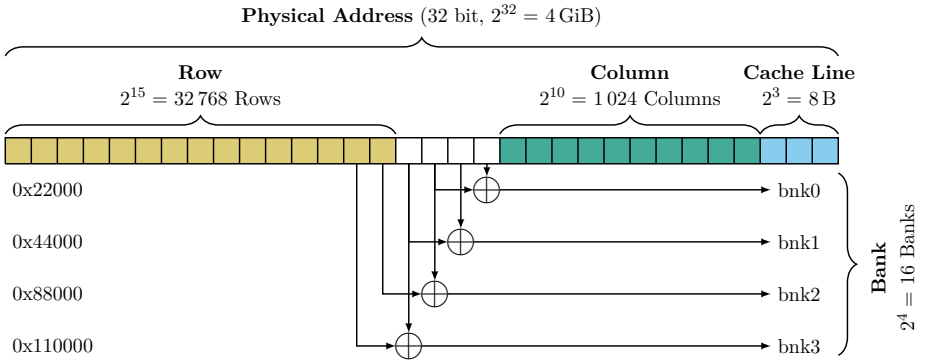


Figure 2.5.: Linear DRAM addressing functions on an Intel Haswell system with one 4 GiB DIMM. The row and column are addressed by directly using bits from the physical address. The banks are addressed by using two bits of the physical address in an XOR operation for each bank bit. The hexadecimal numbers on the left side are bit masks applied to the physical address to mask the bits used in the XOR operation.

2.2. Rowhammer

The capacitors in DRAM cells discharge over time due to charge leakage. Therefore, the cells are refreshed periodically with a refresh window of 64 ms on DDR3 [47] and DDR4 [48], and a refresh window of 32 ms on DDR5 [49]. When a refresh is performed, the current value stored in a DRAM cell is read and afterwards written back to the cell. Therefore, a cell is fully charged or fully discharged after a refresh. Disturbance errors occur only when the leakage is so high that the charge level of a cell changes in a way that it is detected as another charge level at the next refresh [57]. For example, the amount of charge leaking from a charged cell must be so high that the cell is detected as discharged at the next refresh. Thereby, the time an attacker has to trigger a disturbance error is limited to one refresh window.

For DDR4, the memory controller issues `REF` commands every 7.8 µs [29]. Following, ≈8 K `REF` commands are sent over the refresh window of 64 ms. So it is expected that ≈8 K `REF` commands refresh every DRAM row, and every row is refreshed every ≈8 K accesses. However, as shown by Hassan et al. [29], more rows might be refreshed internally, leading to a refresh every 3 758 `REF` commands instead of every ≈8 K `REF` commands.

As hypothesized by the authors, this could be an additional mitigation to protect against Rowhammer. For DDR5, Meyer et al. [75] have shown that the periodic row refresh is larger than ≈8K `REF` commands. The authors have shown factors of 1.3 and 2.0 for different DIMMs, which effectively doubles the refresh window for single cells. This behaviour increases susceptibility, as the aggressor rows can be hammered for a longer time before the victim rows are refreshed.

When two DRAM rows are accessed alternately, the DRAM bank alternates between activated and precharged states. As a result, the capacitors of the cells in these two rows are discharged when the row is activated and (if they were charged before) charged when the row is closed. Due to these charging and discharging cycles, disturbance errors can occur in nearby DRAM cells that were not accessed [57]. The accessed rows are called *aggressors*, and the rows that are likely to contain bit flips afterwards are called *victims*. This type of disturbance error was first mentioned in multiple patent applications by Intel regarding the problem of "row hammer" [26, 5, 6], hence the name *Rowhammer*. According to Kim et al. [57], in the vast majority of cases, DRAM cells lose charge when they are disturbed.

Kim et al. [57] used a setup based on a memory controller they implemented using an FPGA. Thereby, they were able to send DRAM commands directly to the DIMM, bypassing the memory controller. Additionally, they demonstrated that Rowhammer can trigger bit flips when running on an x86 Linux system. The assembly code of their native x86 Linux exploit using the double-sided pattern is shown in Listing 2.2.1, and a graphic representation of the accesses triggered by that code is shown in Figure 2.6.

```
1 hammer:
2   mov (Row 0), %eax
3   mov (Row 2), %ebx
4   clflush (Row 0)
5   clflush (Row 2)
6   jmp hammer
```

Listing 2.2.1: Simplified assembly code used by Kim et al. [57] to exploit Rowhammer on x86 systems.

Instead of using specific access patterns, Seaborn and Dullien [92] randomly selected the addresses they accessed. Thereby, on a system with n banks, they selected two addresses mapping to the same bank with a probability

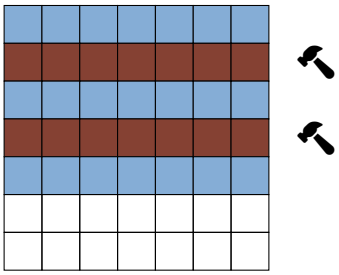
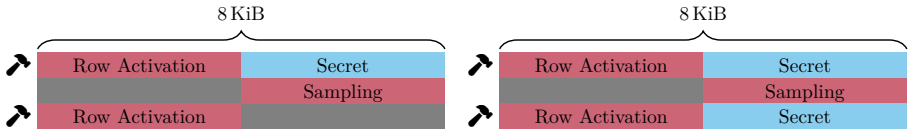


Figure 2.6.: Double-Sided Rowhammer access pattern as used by Kim et al. [57]. The red rows are the ones accessed by the attacker, and the blue rows are the ones likely to contain bit flips afterwards.

of $\frac{1}{n}$. While this has the advantage of not requiring DRAM addressing functions, the likelihood of row hits decreases with an increasing number of banks. Therefore, the number of bit flips identified in a given time using this approach is significantly lower compared to the number of bit flips triggered in a given time when using a more targeted approach.

In contrast to the double-sided Rowhammer access pattern depicted in Figure 2.6, there are also other patterns. For example, Kim et al. [57] also introduced the single-sided pattern, in which two aggressor rows are located in the same bank, but with a distance greater than a single row. There is also the one-location pattern introduced by Gruss et al. [27], which utilizes a single aggressor row and therefore eliminates the need for DRAM addressing functions. Frigo et al. [22] were the first to suggest many-sided access patterns with more than two aggressor rows. Later, Kogler et al. [58] presented the half-double access pattern, in which the second next rows from the victim are accessed. Jattke et al. [45] introduced the entirely novel approach of non-uniform access patterns. In this context, non-uniform means that the aggressors in the pattern are not accessed uniformly, e.g., one aggressor after another, and the pattern is repeated after all aggressors have been accessed. Instead, patterns can be nested, with different access counts for each nested pattern.

Initially, Rowhammer was seen as a solely theoretical problem with no practical impact. This perspective changed when Seaborn and Dullien [92] published two exploits based on Rowhammer: a *Native Client (NaCl)* sandbox escape and a local privilege escalation based on bit flips in `PTEs`. After that, vendors began deploying mitigations that addressed these



(a) Single-Sided RAMBleed (b) Double-Sided RAMBleed

Figure 2.7.: Graphical illustration of RAMBleed based on the illustration of Kwong et al. [60]. The attacker controls the red pages, and the victim controls the blue pages. Grey pages are not used.

attacks. Since then, novel mitigations against known attacks, and novel attacks that bypass known mitigations have been published regularly.

The bit flips triggered by Rowhammer depend on the values stored in the cells of the aggressor rows. In 2020, Kwong et al. [60] introduced a novel approach that does not exploit bit flips to modify the memory of other processes, but rather to read their memory. Since a DRAM row typically has a size of 8 KiB, it can store two whole pages. Depending on the DRAM addressing functions, a single row can also contain parts of more than two pages.

The basic approach of Kwong et al. [60] is to place an attacker-controlled page in the same row as the victim's page to be read. Additionally, another page controlled by the attacker, used to sample disturbance errors, is placed next to the aggressor row controlled by the attacker. When the attacker performs accesses to the aggressor rows, bit flips will occur in the attacker-controlled sampling row. Then, the attacker can read the contents of the sampling row and infer the contents of the victim process's page based on the bit flips that occurred. See Figure 2.7 for a graphical illustration.

To align the pages as described above, the attacker needs to obtain three consecutive 8 KiB rows first. Kwong et al. [60] do this by using the Linux buddy allocator to allocate 2 MiB of consecutive memory; they do not use huge pages. Next, they use the row buffer timing side channel published by Pessl et al. [82] to identify pages within the block that are located in the same DRAM bank. Afterwards, they run a templating phase by using double-sided Rowhammer to identify locations affected by Rowhammer.

The behavior of the buddy allocator is exploited by allocating n dummy pages (where n is the number of pages the victim allocates before the page

that stores the secret). After that, the attacker deallocates the page where the victim should store the secret, followed by deallocating the n pages mapped before. Directly afterwards, the victim is triggered, resulting in the allocation of the n dummy pages, followed by the templated page. Afterwards, the attacker can run double-sided Rowhammer again to read the bits from the victim page. These steps are repeated with multiple sampling pages until a sufficient number of bits is leaked.

The authors show an attack against OpenSSH in which they recovered 68 % (4 200 key bits) at a rate of 0.31 bit/s with an accuracy of 82 %. They then use an optimized key search algorithm to recover the entire RSA private key in about 3 min on a consumer laptop.

3

State of the Art

In this chapter, we describe the state of the art and the impact of our publications on the state of the art in different research directions. First, we consider DRAM addressing function reverse-engineering. Next, we describe the state of the art of Rowhammer attacks and mitigations, as well as the research methodology for Rowhammer. Finally, we discuss the state of the art in the area of other DRAM disturbance errors.

3.1. DRAM Addressing Function Reverse-Engineering

For linear DRAM addressing functions, there are multiple approaches to reverse-engineer them. Initially, high-bandwidth oscilloscopes were used; however, Pessl et al. [82] introduced a novel approach to exploit the side channel of different timings of row hits and row conflicts. In the case of a row hit, the data can be directly read from or written to the row buffer. In contrast, in the case of a row conflict, the current row in the row buffer must be restored to the DRAM array, and the bitlines must be precharged. Afterwards, another row can be loaded into the row buffer before the data can be read from or written to it. Therefore, a row hit is faster than a row conflict, and the timing difference between both cases can be used to estimate whether an access is a row hit or a row conflict.

The basic idea of Pessl et al. [82] is to use addresses and access them alternately in a way that prevents them from being cached by the CPU. This can be achieved, among other methods, by using the `clflush` instruction or utilizing uncached memory or eviction sets. When accesses are performed, the access times are measured and compared to the threshold between row hits and row conflicts. When the access time was faster than

the threshold, the access was a row hit. Otherwise, the access was a row conflict.

Based on this information, it can be determined whether both addresses are on the same bank in different rows (row conflict), or on the same bank in the same row or on different banks (row hit). Therefore, addresses on the same bank can be grouped so that there is one group per bank. Next, addressing functions can be guessed in a way that they follow the general format of linear DRAM bank addressing functions. If a guessed function splits the groups with addresses in half, e.g., has a value of 0 for half of the groups and a value of 1 for the other half, the function is likely correct. However, this also adds functions that are a linear combination of other functions, e.g., when f_1 and f_2 are valid functions, this approach would also identify $f_1 \oplus f_2$ as a valid function. Therefore, the functions are reduced in a way that eliminates functions that are linear combinations of other functions.

Multiple publications use the general approach of time-based DRAM addressing function reverse-engineering with some additional improvements: Frigo et al. [22] published an adjusted version of the implementation by Pessl et al. [82], which, in addition to the bank functions, also reverse-engineers the bits used for row addressing. They identify the bits used for row addressing by selecting addresses within the same bank using their previously reverse-engineered bank addressing functions. When the timing difference between accessing two addresses in the same bank is smaller than the threshold between row hits and row conflicts, and therefore a row hit occurred, both addresses are in the same row within the same bank. Wang et al. [103] improved existing approaches by adding known information about the DRAM layout, which can be retrieved from the system, into the reverse-engineering process. This enabled more targeted reverse-engineering of DRAM addressing functions. Later, we introduced an improved implementation of grouping addresses in our paper, "Reverse-Engineering Bank Addressing Functions on AMD CPUs" [32] (also included as Chapter 5 in this thesis), which led to more stable grouping and also worked on AMD-based systems. Jattke et al. [46] were the first to consider the offsets used by systems with AMD CPUs, resulting in fully-functional addressing function reverse-engineering on AMD systems. In contrast to previous approaches which required exponential time to identify DRAM bank addressing functions, Plin et al. [83] demonstrated a more effective approach for recovering these functions in polynomial time.

Another approach for DRAM addressing function reverse-engineering by Helm et al. [37] relies on performance counters rather than timing differences between row hits and row conflicts. They access specific addresses and use the performance monitoring units (PMUs) of the channels to count transfers on those channels. In detail, they use performance events for each rank with separate umasks for each bank or bank group [40]. Following, each measurement returns the number of transfer events on a specific channel, rank, and bank. By cycling through all possible combinations, they identify one combination with a counter value equal to or higher than the number of accesses they performed. In contrast to the error-prone timing side channel, performance counters are significantly less error-prone, leading to higher reliability. However, this approach requires systems that support these performance counters, which is often only the case on server systems. Therefore, while more stable, the approach is limited to devices supporting the used performance counters.

As shown above, there are multiple approaches for DRAM addressing function reverse-engineering based on timing differences between row hits and row conflicts [82, 22, 103, 32, 46, 83]. There is also an approach based on performance counters [37]. While these approaches identify DRAM addressing functions, the identified functions are not always correct, since at least timing-based approaches are error-prone [37], and the performance-counter-based approach only works on server systems that support the required counters. Therefore, Pessl et al. [82] and Jattke et al. [46] utilize high-bandwidth oscilloscopes to monitor DRAM accesses on the bus physically. While this approach works reliably, it has the disadvantage of requiring specific hardware and physical access, rendering it unrealistic for real-world attacks.

Another approach for verifying reverse-engineered DRAM addressing functions is to use Rowhammer. When the functions are used in a Rowhammer attack, and bit flips occur, the functions are assumed to be correct. However, since Rowhammer also triggers bit flips with randomly selected addresses [92] or when only accessing a single bank [27], even incorrect DRAM addressing functions might trigger bit flips. Therefore, the occurrence of bit flips induced by Rowhammer is not a reliable indicator of correct DRAM addressing functions.

We introduced a novel approach for verifying previously reverse-engineered DRAM addressing functions in our paper “Verifying DRAM Addressing in Software” [33] (also included as Chapter 7 in this thesis). Our approach is based on the timing difference between row hits and row conflicts. It can be

used to verify single DRAM bank addressing functions, e.g., to determine whether a single function in a list of functions to be verified is correct. Since our approach only analyzes differences between a base measurement and a measurement with a single function removed, a constant level of noise does not pose a problem compared to approaches that use the timing side channel for reverse-engineering. In contrast to other approaches, our approach does not require specific hardware or physical access. It is less error-prone than using the existence of bit flips as an indicator for correct addressing functions.

3.2. Comprehensive Overview of Rowhammer Research

Rowhammer is a disturbance effect in DRAM. Frequent access to aggressor rows leads to bit flips in spatially nearby victim rows. The section describes the state of the art in Rowhammer research, grouped by Rowhammer attacks, Rowhammer mitigations, and Rowhammer research methodology.

3.2.1. Rowhammer Attacks

In general, there are different directions of Rowhammer attack research. One direction focuses on the existence of bit flips across different DRAM generations and various scenarios. The second direction demonstrates how Rowhammer can be used in real exploits.

Existence of Bit Flips. Rowhammer attacks have been demonstrated on different DRAM generations, e.g., on DDR3 [57, 10, 85, 107, 28, 27, 94, 95, 50, 60, 116, 81, 81, 98, 55, 30], DDR4 [28, 43, 1, 27, 13, 66, 22, 81, 15, 45, 58, 98, 80, 108, 24, 69, 53, 71, 55, 46, 16, 30, 44, 4, 86], DDR5 [46, 75], LPDDR2 [100, 66], LPDDR3 [100, 115], LPDDR4 [100, 58], LPDDR4X [22, 45, 58], HBM2 [79] and GDDR6 [65].

DDR3 typically requires more activations per refresh window to flip bits. Kim et al. [57] showed that at least 139 K activations are required on the DDR3 DIMMs they tested to trigger bit flips. Since the integration density increased on DDR4, fewer activations are required within a single refresh window. According to Frigo et al. [22], 45 K activations are sufficient to trigger bit flips on DDR4. Later, Olgun et al. [78] showed that as few as

7 K activations per refresh window can be sufficient to induce bit flips on DDR4. According to Meyer et al. [75], 29.3 K activations are sufficient to trigger bit flips on DDR5. Therefore, there is a trend of requiring fewer activations within one refresh window to trigger bit flips in newer DRAM generations. However, since DDR5 uses on-Die ECC, the number of activations required to trigger a sufficient amount of bit flips to not be corrected on DDR5 is not necessarily lower as on DDR4.

Rowhammer disturbance errors have been analyzed using FPGA-based memory controllers [57, 80, 69], which provide researchers with fine-grained control over the DRAM commands sent. Attacks also have been demonstrated locally on x86 systems [27, 94, 50, 60, 98, 55, 43, 1, 22, 116, 45, 58, 24, 69, 53, 46], RISC-V systems [71], ARM systems [115], and mobile devices [100, 22, 58] by executing native code. Additionally, (partly) remote attacks have been demonstrated over the network [95, 66] and via JavaScript in the browser [10, 28, 21, 15, 16]. Other attacks exploit Rowhammer in settings where an attacker and a victim VM run on the same host [85, 107]. In contrast to these scenarios, Rowhammer can also be used as a source of randomness for physically uncloneable functions [20, 91]. Mechelinck et al. [74] suggest an approach to utilize Rowhammer to bind software to a specific device.

We have shown that the number of bit flips occurring in a given time can be amplified by a factor of 830 in our publication "Flipper: Rowhammer on Steroids" [31], which is also included in this thesis in Chapter 6. We use two basic primitives: parallel Rowhammer execution across multiple threads and additional DRAM accesses to increase memory pressure. We verify our approach using six DDR3 DIMMs and demonstrate that it can be used to bypass the mitigation of a double refresh rate.

End-to-End exploits. Initially, Rowhammer was considered a purely theoretical problem with no real-world implications. In 2015, Seaborn and Dullien [92] published the first Rowhammer-based exploits, a NaCl sandbox escape, and a local privilege escalation. The privilege escalation is based on bit flips in `PTEs`. Other attacks exploit bit flips in domains [85] from which the Linux package managers download their packages. Some attacks target the implementation of cryptographic algorithms [85, 8, 18, 76, 3]. There are also attacks based on bit flips in the instructions stored in the page cache [27]. Other attacks target machine learning models [99]. As shown by Kwong et al. [60], Rowhammer can not only affect the integrity

of systems but also compromise confidentiality, e.g., by reading bits in memory without accessing them.

3.2.2. Rowhammer Mitigations

There are four basic directions of Rowhammer mitigations. One approach is to reduce the number of accesses an attacker can perform within a refresh window, thereby keeping the number of activations below the threshold for bit flips to occur. The second direction is to isolate rows so that an attacker can still trigger bit flips in adjacent rows, but these rows are not used for (critical) data. The third direction focuses on remapping rows at runtime, spatially separating aggressor and victim rows (e.g., by mapping the aggressor row to another physical row). Finally, the fourth direction leverages error correction, assuming bit flips will occur, and attempts to correct them.

Reduce accesses per refresh window. The first Rowhammer mitigation on DDR3 was to double the refresh rate, as suggested by Kim et al. [57]. However, this approach would require a refresh window of 8.2 ms to effectively mitigate Rowhammer, resulting in a time overhead of 11 % to 35 % compared to 1.4 % to 4.5 % without using the mitigation. Consequently, reducing the refresh window from 64 ms to 32 ms was ineffective as a mitigation. Another approach was to use pseudo Target Row Refresh (pTRR), in which the DIMM reports a maximum activation count (MAC), a number of activations at which bit flips could happen. The memory controller tracked accesses and sent refresh commands to rows adjacent to those accessed as often as the threshold.

In DDR4, target row refresh (TRR) is implemented directly on the DIMM. Similar to pTRR, TRR counts row activations and performs early refreshes on potential victim rows. However, this approach is based on the detection of Rowhammer patterns and only works for patterns it detects. There were multiple approaches [27, 22, 29, 45, 58] that bypassed TRR using patterns that were not detected at the time. Since TRR is implemented on the DIMM, it can not be updated once deployed. Therefore, novel attacks that bypass TRR affect all DIMMs built before the vendors adjust their TRR implementations to detect these patterns.

Bennett et al. [7] propose to adjust the DRAM design to implement counters. Using these counters, they detect Rowhammer attacks and

notify the memory controller to stop issuing DRAM commands when time for refreshing potential victim rows is required. Marazzi et al. [70] propose ProTRR, a in-DRAM Rowhammer mitigation performing Target Row Refresh with formal security guarantees. ProTRR has a negligible impact on power, performance, and area, and is compatible with DDR4 and the DDR5 Refresh Management (RFM) extension. Jaleel et al. [42] propose probabilistically detecting potential victim rows, e.g., adding each accessed row to a queue with a specific probability. Frequently accessed rows are therefore added with a higher probability. Marazzi et al. [73] propose to add low-overhead buffering sense amplifiers to enable refreshes in parallel to DRAM accesses. Thereby, refreshes can be performed more frequently while only minimally impacting the performance.

To address the issues with DDR4 TRR, DDR5 introduced multiple new features, including per-row activation counters (PRAC), which enable more effective pattern tracing [49]. Woo et al. [106] describe a PRAC-based Rowhammer mitigation that addresses some problems with other PRAC implementations. Additionally, on-die ECC is deployed to detect and correct bit flips on DDR5 [49]. However, as shown by Meyer et al. [75], on SK Hynix DIMMs, some refresh intervals are not counted. By performing accesses only during the not-counted refresh intervals, the authors bypassed these mitigations on DDR5.

In contrast to approaches that focus on adding refreshes, Yağlıkçı et al. [109] suggested throttling accesses to aggressor rows. Similar to TRR or pTRR, this approach requires identifying (potential) aggressor rows. Canpolat et al. [12] propose a similar approach with better performance. In our work, "Memory Band-Aid: A Principled Rowhammer Defense-in-Depth" [19], we leverage dynamic memory bandwidth limits. When a process is detected to perform a Rowhammer attack, the bandwidth of this process can be reduced. Therefore, without increasing the refresh rate or detecting single victim rows, the number of accesses within the refresh window can be limited only for that single process, without affecting other processes. We suggest enabling the limits only on the banks that are hammered, which would reduce the overhead for processes being detected to perform Rowhammer attacks. While this would lead to a low overhead of 0 % to 9.4 % for processes which are detected to run Rowhammer, per-bank limits are not possible with current systems and would require minor hardware changes. In contrast to other approaches, it is sufficient to detect that the process performs a Rowhammer attack and detect the

bank to which the accesses are going. It is not required to identify single aggressor rows.

Spatial isolation of rows. Assuming that bit flips occur in rows adjacent to those activated by the attacker, an approach is to separate the rows spatially. Thereby, bit flips would still happen, but not in rows used for ("critical") data, as described below. Brasser et al. [11] introduced a mitigation that separates rows used by the kernel from those used by user-space processes, thereby preventing exploits targeting kernel memory. Van der Veen et al. [101] adjust this approach by utilizing guard rows to isolate DMA buffers, thereby preventing DMA-based attacks on ARM-based devices. Bock et al. [9] used the same approach, but extended it by isolating rows of different processes against one another. Konoth et al. [59] further extended the approach by isolating each DRAM row containing data with guard rows, which can be used as integrity-checked swap space.

However, as shown by Kogler et al. [58], Rowhammer can also induce bit flips in the second next row. Lang et al. [61] found that the effect extends not only to the second next row but also up to four rows away. These attacks can effectively defeat mitigations based on single guard rows. Loughlin et al. [68] introduce an approach to isolate different VMs by separating them on a subarray basis, e.g., not using rows in the same subarray for different VMs. However, Yuksel et al. [113] showed later that another DRAM disturbance effect, ColumnDisturb, can also affect cells in adjacent subarrays.

Row Remapping. According to Lang et al. [61], Rowhammer can induce bit flips in rows up to four rows away from the aggressor row. Consequently, Saxena et al. [90] propose a mitigation that detects Rowhammer attacks and dynamically relocates the aggressor rows to a dedicated memory location. Therefore, the spatial correlation between the aggressor and victim rows is broken. While bit flips might still occur, they can only happen in the quarantine region (1 % of the DRAM size), which no longer affects the victim row.

Another mitigation suggested by Saileshwar et al. [88] does not require a dedicated quarantine zone. Instead, identified aggressor rows are swapped with randomly selected rows if a threshold of 4.8 K activations is reached. Therefore, assuming the detection mechanism works and the relocation is sufficiently fast, bit flips should no longer occur. Wi et al. [105] propose

a similar mitigation approach that dynamically randomizies DRAM row mappings.

Di Dio et al. [17] introduce a relocation approach based on ECC. The authors propose that when ECC corrects a bit flip, the OS is notified of the correction. Then, the page with the corrected bit flip is relocated to another location in memory and the location with the bit flip is taken offline. Therefore, a location that is detected to be susceptible to Rowhammer is no longer used. However, this works only when the number of bit flips is not sufficiently high to bypass ECC, as shown on DDR3 [14] and DDR4 [54].

Error Correction. Instead of avoiding bit flips by reducing accesses to aggressor rows per refresh window, or remapping rows, or preventing exploitability by spatially isolating rows, another approach is to assume bit flips will occur and attempt to correct them. The most intuitive approach is to use DRAM that already supports error correction codes (ECC), which effectively corrects bit flips that occur. However, it has been demonstrated that ECC can be bypassed on DDR3 [14] and DDR4 [54]. It was also shown that on-die ECC on DDR5 can be bypassed [75].

Another approach that utilizes cryptographic message authentication codes (MACs) for integrity protection. In contrast to ECC, MAC cannot correct errors but only detect them. One approach by Saxena et al. [89] leverages unused bits in `PTEs` to store the cryptographic MAC. When an error is detected, a best-effort correction can be performed by flipping one bit at a time until the MAC matches again. Another approach by Juffinger et al. [52] also uses cryptographic MACs and brute-force error correction. However, this approach works on the entire DRAM, and the authors demonstrated that an arbitrary number of bit flips can be detected with high probability, assuming fewer than one silent data corruption per 10^9 years. Errors of up to 8 bit flips within 256 bits (DDR5) or 512 bits (DDR4) can be corrected in practical time constraints.

3.2.3. Rowhammer Research Methodology

As shown above, Rowhammer attacks have been demonstrated on various DRAM generations, ranging from DDR3 to DDR5, from LPDDR2 to LPDDR4x, and even to GDDR6. There are also multiple end-to-end exploits in different scenarios, from local privilege escalation to cross-VM attacks. According to academic publications, e.g., [57, 100, 45, 58, 69,

24, 46], Rowhammer is a significant problem that affects the majority of systems. However, there is no known case of Rowhammer being used as part of a real-world attack chain.

We analyze this stark discrepancy in our publication "Epistemology of Rowhammer Attacks: Threats to Rowhammer Research Validity" [36], which is included in this thesis at Chapter 8. We identify six threats to Rowhammer research validity and provide seven suggestions for future research to consider, aiming to make more realistic claims regarding the severity of novel attacks. Thereby, the discrepancy between academic attention and real-world impact can be reduced in the future.

One of our suggestions is to conduct large-scale studies on Rowhammer prevalence. In our publication "FlippyRAM: A Large-Scale Study of Rowhammer Prevalence" [35] (included in this thesis at Chapter 9), we perform a large-scale study of Rowhammer prevalence. We show that 12.5 % of the 1 006 datasets we collected from 822 unique systems are susceptible to Rowhammer. Additionally, we show that the main reason Rowhammer attacks do not work in our automated setup is the lack of fully automated DRAM addressing function reverse-engineering tools (since DRAM addressing function reverse-engineering failed on approximately 50 % of the systems). Additionally, the lack of tools that yield good results when running Rowhammer on AMD systems and on systems with DDR5 reduced the number of systems identified as susceptible. Therefore, we hypothesize that the number of systems susceptible to Rowhammer is actually higher, and the number reported in the publication is a lower bound. Future research in automated DRAM addressing function reverse-engineering and improved support for various system types may significantly increase the fraction of susceptible systems.

3.3. Other DRAM Disturbance Errors

Besides Rowhammer, other DRAM disturbance errors can also trigger bit flips in DRAM cells. We describe two of them in this section.

3.3.1. Rowpress

Another DRAM disturbance error is *Rowpress*, first described by Luo et al. [69]. In contrast to Rowhammer, which maximizes the number of

open and close operations on the accessed rows, Rowpress keeps a row open as long as possible. Thereby, the show that the number of row activations required to trigger bit flips can be reduced by one to two orders of magnitude compared to Rowhammer. Similar to Rowhammer patterns, the authors proposed single-sided and double-sided Rowpress, attempting to alternately keep one of the aggressor rows open. Thus, they find bit flips in rows adjacent to the aggressor rows they are accessing.

They demonstrate, using an FPGA-based setup, that the Rowpress effect can trigger bit flips that differ from the bit flips triggered when Rowhammer is used on the same rows. In addition to the FPGA-based setup, they perform an experimental evaluation of Rowpress on DDR4-based systems. They show that Rowpress can trigger bit flips on systems on which Rowhammer can not induce any bit flips due to Rowhammer mitigations. Therefore, they conclude that Rowpress has to be a different disturbance effect.

In our work "Presshammer: Rowhammer and Rowpress without Physical Address Information" [53], we show that one-location Rowhammer (e.g., accessing a single row) induces nearly the same amount of bit flips as double-sided Rowhammer, but only 61.8 % of the locations overlap. Therefore, we conclude that our one-location Rowhammer attack induces some bit flips due to the Rowhammer effect (the ones which overlap with the double-sided Rowhammer attack) and some bit flips due to the Rowpress effect (the ones which to not overlap). We present the first end-to-end, one-location Rowpress attack that escalates to kernel privileges within less than 10 min.

3.3.2. ColumnDisturb

Rowhammer and Rowpress typically affect the rows directly adjacent to the aggressor rows. As shown by Kogler et al. [58], it is also possible to trigger bit flips in the second-next row of the aggressor. Lang et al. [61] demonstrate that Rowhammer not only affects the second-next row but also rows over a distance of up to 4. So, Rowhammer can trigger bit flips in spatially nearby rows, where *nearby* can be up to 4 rows away. Consequently, refreshing rows directly adjacent to the aggressor row does not fully mitigate Rowhammer and, as shown by Kogler et al. [58], might even support Rowhammer attacks.

As shown by Yuksel et al. [113], another DRAM disturbance effect, called *ColumnDisturb*, also occurs. In contrast to Rowhammer, ColumnDisturb induces disturbance errors in rows more than 4 rows apart. It triggers disturbance errors in rows that are not even in the same subarrays as the aggressor row. They show that disturbance errors are triggered across three subarrays, *i.e.*, 3 072 DRAM rows.

4

Conclusion

In this thesis, we have discussed the ongoing problem of Rowhammer attacks. For more than a decade, vendors have attempted to mitigate Rowhammer, and researchers have published novel attacks that bypass these mitigations. At the same time, novel approaches to mitigate Rowhammer are published. We can draw conclusions in four different aspects from this thesis:

DRAM Addressing Function Reverse-Engineering. We presented a novel approach for reverse-engineering the DRAM bank addressing function on AMD CPUs [32], which outperformed existing tools on systems with AMD CPUs at the time of publication. Later, the approach is superseded by an offset-aware approach by Jattke et al. [46]. Since timing-based DRAM addressing function reverse-engineering is prone to errors, verification of the addressing functions should be performed. High-bandwidth oscilloscopes can be used to verify identified addressing functions. We presented a novel approach for software-only DRAM bank addressing function verification that leverages the timing side channel to verify a given set of DRAM bank addressing functions [33]. Our approach can not only verify entire sets, but also single addressing functions within the set, and thereby detect whether individual addressing functions are correct or incorrect. These approaches enable a more reliable reverse-engineering of addressing functions in general. However, more research is needed to further improve stability.

DRAM Disturbance Attacks. To exploit Rowhammer, bit flips have to fulfil specific requirements regarding their location and their direction. For example, for opcode flipping presented by Gruss et al. [27], the bit must flip in the correct direction and at the correct offset within the page. Within the byte specified by the offset, the correct bit must flip and additionally, no

relevant other bits must flip in the page (e.g., modifying other instructions that are executed). For the latter problem, an approach was suggested by Ji et al. [50]. Regarding the first problem, we have presented an approach which increases the number of bit flips on a DDR3 system by a factor of up to 830 [31]. Thereby, more bit flips occur, and the probability of exploitable bit flips increases. We have also demonstrated that our approach can bypass the double refresh rate Rowhammer mitigation often used on DDR3 systems. With Presshammer [53], we analyzed the differences between the Rowpress [69] and Rowhammer [57] effects. We have shown that one-location Rowhammer and one-location Rowpress cannot be separated from each other, since the implementation triggers both effects. These approaches enable stronger Rowhammer attacks. However, they should be analyzed in combination with newer DRAM technologies, e.g., DDR5.

Rowhammer Mitigations. In general, there are multiple directions of Rowhammer mitigations: There are Rowhammer mitigations based on the reduction of activations per refresh window [57, 109], based on spatial isolation of rows [11, 101, 9, 59, 68], based on row remapping [90, 88, 105, 17], or based on error correction [89, 52]. Proprietary mitigations can also be grouped by these mitigation types. Doubling the refresh rate, pTRR, and TRR reduce the activations per refresh window, and ECC is based on error correction. With Memory Band-Aid [19], we proposed a mitigation approach based on reducing activations per refresh window. Instead of detecting single aggressor rows, our mitigation detects if a Rowhammer attack is running. In that case, the memory bandwidth of the process is limited to reduce the effective number of accesses the process can do within a refresh window. In contrast to other mitigations, this only limits the memory bandwidth of that process, without affecting other processes. While this approach can mitigate Rowhammer in general, hardware support would be required to use it more efficiently.

Rowhammer Research Methodology. According to academic publications, Rowhammer is a huge problem, and the majority of systems are affected. However, there is a stark discrepancy between academic publications and real-world exploitation, since, to the best of our knowledge, Rowhammer has not yet been used to attack real systems outside of research. We analyzed Rowhammer-related publications with a focus on the experimental evaluations performed [36]. We identify six threats

to Rowhammer research validity and propose seven suggestions to improve Rowhammer research validity in the future. To address the lack of information regarding general Rowhammer susceptibility, we conducted a large-scale Rowhammer study, collecting 1 006 datasets from 822 unique systems [35]. We show that the primary reason systems are not susceptible to Rowhammer in our experiments is the failure to reverse-engineer DRAM addressing functions. Additionally, we did not add tools that yield good results on systems with AMD CPUs and systems with DDR5, which have become available in the meantime. We have shown that Rowhammer is a real-world problem affecting at least 12.5 % of the datasets we collected. However, more research is needed in this area with additional tools like ZenHammer [46] or Phoenix [75] to gain insights of the Rowhammer susceptibility of systems with AMD CPUs and DDR5 DRAM. Additionally, our study is limited to x86 devices, so other studies could evaluate the prevalence of Rowhammer on different architectures like ARM.

References

[1] Misiker Tadesse Aga, Zelalem Birhanu Aweke, and Todd Austin. When good protections go bad: Exploiting anti-DoS measures to accelerate Rowhammer attacks. In: HOST. 2017 (pp. 4, 26, 27).

[2] AMD. BIOS and Kernel Developer's Guide (BKDG) for AMD Family 16h Models 00h-0Fh Processors. 2015. URL: `https://www.amd.com/content/dam/amd/en/documents/archived-tech-docs/programmer-references/48751_16h_bkdg.pdf` (p. 15).

[3] Samy Amer, Yingchen Wang, Hunter Kippen, Thinh Dang, Daniel Genkin, Andrew Kwong, Alexander Nelson, and Arkady Yerukhimovich. PQ-Hammer: End-to-end Key Recovery Attacks on Post-Quantum Cryptography Using Rowhammer. In: S&P. 2025 (p. 27).

[4] Seungmin Baek, Minbok Wi, Seonyong Park, Hwayong Nam, Michael Jaemin Kim, Nam Sung Kim, and Jung Ho Ahn. Marionette: A RowHammer Attack via Row Coupling. In: ASPLOS. 2025 (pp. 4, 26).

[5] K.S. Bains and J.B. Halbert. Row hammer monitoring based on stored row hammer threshold value. US Patent App. 13/690,523. 2014 (p. 18).

[6] K.S. Bains, J.B. Halbert, C.P. Mozak, T.Z. Schoenborn, and Z. Greenfield. Row hammer refresh command. US Patent App. 13/539,415. 2014 (p. 18).

[7] Tanj Bennett, Stefan Saroiu, Alec Wolman, and Lucian Cojocar. Panopticon: A Complete In-DRAM Rowhammer Mitigation. DRAMSec. 2021 (p. 28).

[8] Sarani Bhattacharya and Debdeep Mukhopadhyay. Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. In: CHES. 2016 (p. 27).

[9] Carsten Bock, Ferdinand Brasser, David Gens, Christopher Liebchen, and Ahamd-Reza Sadeghi. RIP-RH: Preventing Rowhammer-based Inter-Process Attacks. In: AsiaCCS. 2019 (pp. 30, 36).

[10] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In: S&P. 2016 (pp. 4, 26, 27).

[11] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In: USENIX Security. 2017 (pp. 30, 36).

[12] Oğuzhan Canpolat, A Giray Yağlıkçı, Ataberk Olgun, Ismail Emir Yuksel, Yahya Can Tuğrul, Konstantinos Kanellopoulos, Oğuz Ergin, and Onur Mutlu. BreakHammer: Enhancing RowHammer Mitigations by Carefully Throttling Suspect Threads. In: MICRO. 2024 (p. 29).

[13] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In: S&P. 2020 (pp. 4, 26).

[14] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In: S&P. 2019 (p. 31).

[15] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript. In: USENIX Security. 2021 (pp. 4, 26, 27).

[16] Finn de Ridder, Patrick Jattke, and Kaveh Razavi. Posthammer: Pervasive Browser-based Rowhammer Attacks with Postponed Refresh Commands. In: USENIX Security. 2025 (pp. 4, 26, 27).

[17] Andrea Di Dio, Koen Koning, Herbert Bos, and Cristiano Giuffrida. Copy-on-Flip: Hardening ECC Memory Against Rowhammer Attacks. In: NDSS. 2023 (pp. 31, 36).

[18] Michael Fahr Jr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray Perlner, Arkady Yerukhimovich, et al. When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer. In: CCS. 2022 (p. 27).

[19] Carina Fiedler, Jonas Juffinger, Sudheendra Raghav Neela, Martin Heckel, Hannes Weissteiner, Abdullah Giray Yağlıkçı, Florian Adamsky, and Daniel Gruss. Memory Band-Aid: A Principled Rowhammer Defense-in-Depth. In: NDSS. 2026 (pp. 8, 29, 36).

[20] Bernhard Fischer, Daniel Dorfmeister, Harald Lampesberger, and Eckehard Hermann. Leveraging Rowhammer for Physically Unique and Non-tamperable Device Identification. In: Procedia Computer Science (2025) (p. 27).

[21] Pietro Frigo, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU. In: S&P. 2018 (p. 27).

[22] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In: S&P. 2020 (pp. 3, 4, 19, 24–28).

[23] Lukas Gerlach, Simon Schwarz, Nicolas Faroß, and Michael Schwarz. Efficient and generic microarchitectural hash-function recovery. In: S&P. 2024 (p. 16).

[24] Lukas Gerlach, Fabian Thomas, Robert Pietsch, and Michael Schwarz. A Rowhammer Reproduction Study Using the Blacksmith Fuzzer. In: ESORICS. 2023 (pp. 4–6, 26, 27, 32).

[25] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks. In: USENIX Security. 2018 (p. 3).

[26] Zvika Greenfield and Tomer Levy. Throttling support for row-hammer counters. US Patent 9251885. 2014 (p. 18).

[27] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In: S&P. 2018 (pp. 4, 19, 25–28, 35).

[28] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016 (pp. 4, 26, 27).

[29] Hasan Hassan, Yahya Can Tugrul, Jeremie S. Kim, Victor van der Veen, Kaveh Razavi, and Onur Mutlu. Uncovering In-DRAM RowHammer Protection Mechanisms:A New Methodology, Custom RowHammer Patterns, and Implications. In: MICRO. 2021 (pp. 17, 28).

[30] Wei He, Zhi Zhang, Yueqiang Cheng, Wenhao Wang, Wei Song, Yansong Gao, Qifei Zhang, Kang Li, Dongxi Liu, and Surya Nepal. WhistleBlower: A System-Level Empirical Study on RowHammer. In: IEEE Transactions on Computers (2023) (pp. 4, 26).

[31] Martin Heckel and Florian Adamsky. Flipper: Rowhammer on Steroids. In: uASC. 2025 (pp. 4, 5, 27, 36).

[32] Martin Heckel and Florian Adamsky. Reverse-Engineering Bank Addressing Functions on AMD CPUs. In: DRAMSec. 2023 (pp. 3–5, 12, 24, 25, 35).

[33] Martin Heckel, Florian Adamsky, Jonas Juffinger, Fabian Rauscher, and Daniel Gruss. Verifying DRAM Addressing in Software. In: ESORICS. 2025 (pp. 4, 6, 25, 35).

[34] Martin Heckel, Daniel Gruss, and Florian Adamsky. Ten Years of Rowhammer: A Retrospect (and Path to the Future). 2026. URL: https://media.ccc.de/v/38c3-ten-years-of-rowhammer-a-retrospect-and-path-to-the-future (p. 7).

[35] Martin Heckel, Nima Sayadi, Jonas Juffinger, Carina Fiedler, Daniel Gruss, and Florian Adamsky. FlippyR.AM: A Large-Scale Study of Rowhammer Prevalence. In: NDSS. 2026 (pp. 5, 7, 32, 37).

[36] Martin Heckel, Hannes Weissteiner, Florian Adamsky, and Daniel Gruss. Epistemology of Rowhammer Attacks: Threats to Rowhammer Research Validity. In: ESORICS. 2025 (pp. 5–7, 32, 36).

[37] Christian Helm, Soramichi Akiyama, and Kenjiro Taura. Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE. 2020 (p. 25).

[38] Intel. 5-Level Paging and 5-Level EPT. 2017. URL: https://www.intel.com/content/www/us/en/content-details/671442/ (p. 14).

[39] Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual. 2025. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html (pp. 14, 15).

[40] Intel. Intel® Xeon® Processor Scalable Memory Family Uncore Performance Monitoring Reference Manual. 2017. URL: https://www.intel.com/content/www/us/en/content-details/671389/ (p. 25).

[41] Bruce Jacob, David Wang, and Spencer Ng. Memory Systems: cache, DRAM, Disk. Morgan Kaufmann Publishers, 2008 (p. 12).

[42] Aamer Jaleel, Gururaj Saileshwar, Stephen W. Keckler, and Moinuddin Qureshi. PrIDE: Achieving Secure Rowhammer Mitigation with Low-Cost In-DRAM Trackers. In: ISCA. 2024 (p. 29).

[43] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In: SysTEX. 2017 (pp. 4, 26, 27).

[44] Patrick Jattke, Michele Marazzi, Flavien Solt, Max Wipfli, Stefan Gloor, and Kaveh Razavi. MCSEE: Evaluating Advanced Rowhammer Attacks and Defenses via Automated DRAM Traffic Analysis. In: Usenix Security. 2025 (pp. 4, 26).

[45] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: Rowhammering in the Frequency Domain. In: S&P. 2021 (pp. 4–6, 19, 26–28, 31).

[46] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms. In: USENIX Security. 2024 (pp. 3–7, 16, 24–27, 32, 35, 37).

[47] JEDEC Solid State Technology Association. DDR3 SDRAM STANDARD. 2012. URL: https://www.jedec.org/standards-documents/docs/jesd-79-3d (pp. 9, 17).

[48] JEDEC Solid State Technology Association. DDR4 SDRAM Standard. 2021. URL: https://www.jedec.org/standards-documents/docs/jesd79-4a (pp. 9, 17).

[49] JEDEC Solid State Technology Association. DDR5 SDRAM Standard. 2024. URL: https://www.jedec.org/standards-documents/docs/jesd79-5c01 (pp. 9, 17, 29).

[50] Sangwoo Ji, Youngjoo Ko, Saeyoung Oh, and Jong Kim. Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks. In: AsiaCCS. 2019 (pp. 4, 26, 27, 36).

[51] Zhen Hang Jiang, Yunsi Fei, and David Kaeli. Exploiting bank conflict-based side-channel timing leakage of gpus. In: ACM TACO (2019) (p. 3).

[52] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. CSI: Rowhammer - Cryptographic Security and Integrity against Rowhammer. In: S&P. 2023 (pp. 31, 36).

[53] Jonas Juffinger, Sudheendra Raghav Neela, Martin Heckel, Lukas Schwarz, Florian Adamsky, and Daniel Gruss. Presshammer: Rowhammer and Rowpress without Physical Address Information. In: DIMVA. 2024 (pp. 4, 7, 8, 26, 27, 33, 36).

[54] Nureddin Kamadan, Walter Wang, Stephan van Schaik, Christina Garman, Daniel Genkin, and Yuval Yarom. ECC. fail: Mounting Rowhammer Attacks on DDR4 Servers with ECC Memory. In: USENIX Security. 2025 (p. 31).

[55] Ingab Kang, Walter Wang, Jason Kim, Stephan van Schaik, Youssef Tobah, Daniel Genkin, Andrew Kwong, and Yuval Yarom. SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism. In: USENIX Security. 2024 (pp. 4, 26, 27).

[56] B. Keeth, R. Baker, B. Johnson, and F. Lin. DRAM Circuit Design : Fundamental and High-Speed Topics. 2008. ISBN: 0470184752 (p. 9).

[57] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: ISCA. 2014 (pp. 3–7, 10, 17–19, 26–28, 31, 36).

[58] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In: USENIX Security. 2022 (pp. 4–6, 19, 26–28, 30, 31, 33).

[59] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks. In: USENIX OSDI. 2018 (pp. 30, 36).

[60] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In: S&P. 2020 (pp. 4, 20, 26, 27).

[61] Zhenrong Lang, Patrick Jattke, Michele Marazzi, and Kaveh Razavi. Blaster: Characterizing the blast radius of rowhammer. In: DRAMSec. 2023 (pp. 30, 33).

[62] Yoochan Lee, Jinhan Kwak, Junesoo Kang, Yuseok Jeon, and Byoungyoung Lee. PSPRAY: Timing Side-Channel based Linux Kernel Heap Exploitation Technique. In: USENIX Security. 2023 (p. 3).

[63] Lenovo. BIOS Update Utility README (T540p). 2015. URL: https://download.lenovo.com/pccbbs/mobiles/gmuj16us.txt (p. 4).

[64] Lenovo. BIOS Update Utility README (X230T). 2015. URL: https://download.lenovo.com/pccbbs/mobiles/gcuj22us.txt (p. 4).

[65] Chris S. Lin, Joyce Qu, and Gururaj Saileshwar. GPUHammer: Rowhammer Attacks on GPU Memories are Practical. In: USENIX Security. 2025 (pp. 4, 26).

[66] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. Nethammer: Inducing Rowhammer Faults through Network Requests. In: SILM Workshop. 2020 (pp. 4, 26, 27).

[67] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In: S&P. 2021 (p. 3).

[68] Kevin Loughlin, Jonah Rosenblum, Stefan Saroiu, Alec Wolman, Dimitrios Skarlatos, and Baris Kasikci. Siloz: Leveraging DRAM isolation domains to prevent inter-VM rowhammer. In: SOSP. 2023 (pp. 30, 36).

[69] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In: ISCA. 2023 (pp. 4–7, 26, 27, 31, 32, 36).

[70] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. PROTRR: Principled yet Optimal In-DRAM Target Row Refresh. In: S&P. 2022 (p. 29).

[71] Michele Marazzi and Kaveh Razavi. RISC-H: Rowhammer Attacks on RISC-V. In: DRAMSec Workshop. 2024 (pp. 4, 26, 27).

[72] Michele Marazzi, Tristan Sachsenweger, Flavien Solt, Peng Zeng, Kubo Takashi, Maksym Yarema, and Kaveh Razavi. HiFi-DRAM: Enabling High-fidelity DRAM Research by Uncovering Sense Amplifiers with IC Imaging. In: ISCA. 2024 (p. 10).

[73] Michele Marazzi, Flavien Solt, Patrick Jattke, Kubo Takashi, and Kaveh Razavi. REGA: Scalable Rowhammer Mitigation with Refresh-Generating Activations. In: S&P. 2023 (p. 29).

[74] Ruben Mechelinck, Daniel Dorfmeister, Bernhard Fischer, Stijn Volckaert, and Stefan Brunthaler. GlueZilla: Efficient and Scalable Software to Hardware Binding using Rowhammer. In: DIMVA. 2024 (p. 27).

[75] Diego Meyer, Patrick Jattke, Michele Marazzi, Salman Qazi, Daniel Moghimi, and Kaveh Razavi. Phoenix: Rowhammer Attacks on DDR5 with Self-Correcting Synchronization. In: S&P. 2026 (pp. 4, 18, 26, 27, 29, 31, 37).

[76] Koksal Mus, Yarkın Doröz, M Caner Tol, Kristi Rahman, and Berk Sunar. Jolt: Recovering TLS Signing Keys via Rowhammer Faults. In: S&P. 2023 (p. 27).

[77] Mathias Oberhuber, Martin Unterguggenberger, Lukas Maar, Andreas Kogler, and Stefan Mangard. Power-Related Side-Channel Attacks using the Android Sensor Framework. In: NDSS. 2025 (p. 3).

[78] Ataberk Olgun, F. Nisa Bostanci, Ismail Emir Yuksel, Oguzhan Canpolat, Haocong Luo, Geraldo F. Oliveira, A. Giray Yağlıkçı, Minesh Patel, and Onur Mutlu. Variable Read Disturbance: An Experimental Analysis of Temporal Variation in DRAM Read Disturbance. In: HPCA. 2025 (p. 26).

[79] Ataberk Olgun, Majd Osseiran, A Giray Yağlıkçı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. An Experimental Analysis of RowHammer in HBM2 DRAM Chips. In: DSN. 2023 (p. 26).

[80] Lois Orosa, Ulrich Rührmair, A Giray Yağlıkçı, Haocong Luo, Ataberk Olgun, Patrick Jattke, Minesh Patel, Jeremie Kim, Kaveh Razavi, and Onur Mutlu. SpyHammer: Using RowHammer to Remotely Spy on Temperature. In: arXiv:2210.04084 (2022) (pp. 4, 26, 27).

[81] Lois Orosa, Abdullah Giray Yağlıkçı, Haocong Luo, Ataberk Olgun, Jisung Park, Hasan Hassan, Minesh Patel, Jeremie S. Kim, and Onur 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. 4, 26).

[82] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security. 2016 (pp. 3–6, 13, 20, 23–25).

[83] Antoine Plin, Lorenzo Casalino, Thomas Rokicki, and Ruben Salvador. Knock-Knock: Black-Box, Platform-Agnostic DRAM Address-Mapping Reverse Engineering. In: uASC. 2025 (pp. 24, 25).

[84] Mark Randolph and William Diehl. Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman. In: Cryptography 4.2 (2020) (p. 3).

[85] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In: USENIX Security. 2016 (pp. 4, 26, 27).

[86] Finn de Ridder, Patrick Jattke, and Kaveh Razavi. Softhammer: Exploiting Rowhammer Bit Flips without Crashing. In: DRAMSec. 2025 (pp. 4, 26).

[87] Scott Rixner, William J Dally, Ujval J Kapasi, Peter Mattson, and John D Owens. Memory access scheduling. In: ACM SIGARCH Computer Architecture News (2000) (p. 12).

[88] Gururaj Saileshwar, Bolin Wang, Moinuddin Qureshi, and Prashant J Nair. Randomized Row-Swap: Mitigating Row Hammer by Breaking Spatial Correlation between Aggressor and Victim Rows. In: ASPLOS. 2022 (pp. 30, 36).

[89] Anish Saxena, Gururaj Saileshwar, Jonas Juffinger, Andreas Kogler, Daniel Gruss, and Moinuddin Qureshi. PT-Guard: Integrity-Protected Page Tables to Defend Against Breakthrough Rowhammer Attacks. In: DSN. 2023 (pp. 31, 36).

[90] Anish Saxena, Gururaj Saileshwar, Prashant J Nair, and Moinuddin Qureshi. AQUA: Scalable Rowhammer Mitigation by Quarantining Aggressor Rows at Runtime. In: MICRO. 2022 (pp. 30, 36).

[91] Andre Schaller, Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Stefan Katzenbeisser, and Jakub Szefer. Intrinsic Rowhammer PUFs: Leveraging the Rowhammer effect for improved security. In: Hardware Oriented Security and Trust (HOST. 2017 (p. 27).

[92] Mark Seaborn and Thomas Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. 2015. URL: http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html (pp. 3, 4, 18, 19, 25, 27).

[93] Chaoqun Shen, Congcong Chen, and Jiliang Zhang. Micro-architectural cache side-channel attacks and countermeasures. In: DAC. 2021 (p. 3).

[94] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In: RAID. 2018 (pp. 4, 26, 27).

[95] Andrei Tatar, Radhesh Krishnan, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer Attacks over the Network and Defenses. In: USENIX ATC. 2018 (pp. 4, 26, 27).

[96] The Linux Kernel. HugeTLB Pages. 2025. URL: https://docs.kernel.org/admin-guide/mm/hugetlbpage.html (p. 15).

[97] The Linux Kernel. Transparent Hugepage Support. 2025. URL: https://docs.kernel.org/admin-guide/mm/transhuge.html (p. 15).

[98] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G Shin. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In: S&P. 2022 (pp. 4, 26, 27).

[99] M Caner Tol, Saad Islam, Andrew J Adiletta, Berk Sunar, and Ziming Zhang. Don't Knock! Rowhammer at the Backdoor of DNN Models. In: DSN. 2023 (p. 27).

[100] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In: CCS. 2016 (pp. 4–6, 26, 27, 31).

[101] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM. In: DIMVA. 2018 (pp. 30, 36).

[102] Daimeng Wang, Ajaya Neupane, Zhiyun Qian, Nael Abu-Ghazaleh, Srikanth V Krishnamurthy, Edward JM Colbert, and Paul Yu. Unveiling your keystrokes: A Cache-based Side-channel Attack on Graphics Libraries. In: NDSS. 2019 (p. 3).

[103] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. DRAMDig: A Knowledge-assisted Tool to Uncover DRAM Address Mapping. In: Design Automation Conference (DAC). 2020 (pp. 3, 4, 24, 25).

[104] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. PwrLeak: Exploiting Power Reporting Interface for Side-Channel Attacks on AMD SEV. In: DIMVA. 2023 (p. 3).

[105] Minbok Wi, Jaehyun Park, Seoyoung Ko, Michael Jaemin Kim, Nam Sung Kim, Eojin Lee, and Jung Ho Ahn. SHADOW: Preventing Row Hammer in DRAM with Intra-Subarray Row Shuffling. In: HPCA. 2023 (pp. 30, 36).

[106] Jeonghyun Woo, Shaopeng Chris Lin, Prashant J Nair, Aamer Jaleel, and Gururaj Saileshwar. QPRAC: Towards Secure and Practical PRAC-based Rowhammer Mitigation using Priority Queues. In: HPCA. 2025 (p. 29).

[107] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: USENIX Security. 2016 (pp. 4, 26, 27).

[108] A Giray Yağlıkçı, Haocong Luo, Geraldo F De Oliviera, Ataberk Olgun, Minesh Patel, Jisung Park, Hasan Hassan, Jeremie S Kim, Lois Orosa, and Onur Mutlu. Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. In: DSN. 2022 (pp. 4, 26).

[109] A. Giray Yağlıkçı, Minesh Patel, Jeremie S. Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, Saugata Ghose, and Onur Mutlu. BlockHammer: Preventing RowHammer at Low Cost

by Blacklisting Rapidly-Accessed DRAM Rows. In: HPCA. 2021 (pp. 29, 36).

[110] Qing Yang, Paolo Gasti, Gang Zhou, Aydin Farajidavar, and Kiran S Balagani. On inferring browsing activity on smartphones via USB power analysis side-channel. In: IEEE Transactions on Information Forensics and Security (2016) (p. 3).

[111] Yuval Yarom and Naomi Benger. Recovering OpenSSL ECDSA Nonces Using the FLUSH+ RELOAD Cache Side-channel Attack. In: Cryptology ePrint Archive, Report 2014/140 (2014) (p. 3).

[112] Yuval Yarom and Katrina Falkner. Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. In: USENIX Security. 2014 (p. 3).

[113] Ismail Emir Yuksel, Ataberk Olgun, Nisa Bostanci, Haocong Luo, Abdullah Giray Yağlıkçı, and Onur Mutlu. ColumnDisturb: Understanding Column-based Read Disturbance in Real DRAM Chips and Implications for Future Systems. In: MICRO. 2025 (pp. 10, 30, 34).

[114] Jiliang Zhang, Congcong Chen, Jinhua Cui, and Keqin Li. Timing Side-channel Attacks and Countermeasures in CPU Microarchitectures. In: ACM Computing Surveys 56.7 (2024) (p. 3).

[115] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. Triggering Rowhammer Hardware Faults on ARM: A Revisit. In: ASHES Workshop. 2018 (pp. 4, 26, 27).

[116] Zhi Zhang, Yueqiang Cheng, Dongxi Liu, Surya Nepal, Zhi Wang, and Yuval Yarom. PThammer: Cross-User-Kernel-Boundary Rowhammer through Implicit Accesses. In: MICRO. 2020 (pp. 26, 27).

[117] Mark Zhao and G Edward Suh. FPGA-based Remote Power Side-Channel Attacks. In: S&P. 2018 (p. 3).

Part II.

Publications

List of Publications

During my thesis, I contributed to 7 publications in conference proceedings, 5 of which are included in this thesis as shown below.

Publications in this Thesis

1. **Martin Heckel** and Florian Adamsky. Flipper: Rowhammer on Steroids. In: uASC. 2025.

2. **Martin Heckel** and Florian Adamsky. Reverse-Engineering Bank Addressing Functions on AMD CPUs. In: DRAMSec. 2023.

3. **Martin Heckel**, Florian Adamsky, Jonas Juffinger, Fabian Rauscher, and Daniel Gruss. Verifying DRAM Addressing in Software. In: ESORICS. 2025.

4. **Martin Heckel**, Nima Sayadi, Jonas Juffinger, Carina Fiedler, Daniel Gruss, and Florian Adamsky. FlippyR.AM: A Large-Scale Study of Rowhammer Prevalence. In: NDSS. 2026.

5. **Martin Heckel**, Hannes Weissteiner, Florian Adamsky, and Daniel Gruss. Epistemology of Rowhammer Attacks: Threats to Rowhammer Research Validity. In: ESORICS. 2025.

Other Contributions

1. Carina Fiedler, Jonas Juffinger, Sudheendra Raghav Neela, **Martin Heckel**, Hannes Weissteiner, Abdullah Giray Yağlıkçı, Florian Adamsky, and Daniel Gruss. Memory Band-Aid: A Principled Rowhammer Defense-in-Depth. In: NDSS. 2026.

2. Jonas Juffinger, Sudheendra Raghav Neela, **Martin Heckel**, Lukas Schwarz, Florian Adamsky, and Daniel Gruss. Presshammer: Rowhammer and Rowpress without Physical Address Information. In: DIMVA. 2024.

5

Reverse-Engineering Bank Addressing Functions on AMD CPUs

Publication Data

Martin Heckel and Florian Adamsky. Reverse-Engineering Bank Addressing Functions on AMD CPUs. In: DRAMSec. 2023

Contributions

Main author.

Reverse-Engineering Bank Addressing Functions on AMD CPUs

Martin Heckel[1,2], Florian Adamsky[1]
[1]firstname.lastname@hof-university.de
[1]*Institute of Information Systems (iisys) Hof University of Applied Sciences* Hof, Germany
[2]*Institute of Information Security (ISEC) Graz University of Technology* Graz, Austria

Abstract

The memory controller of the CPU uses bank addressing functions to determine physical locations within DRAM DIMMs. There are many fields of application for these addressing functions, particularly in security. For example, many Rowhammer proof-of-concepts use bank addressing functions to select addresses located on the same bank but in different rows to produce row conflicts. AMD provides these addressing functions for older CPU models. Hence, research on reverse-engineering addressing functions mainly targeted Intel CPUs since Intel did not publish these functions. However, AMD stopped to publish the DRAM addressing functions several years ago. AMD manufactures roughly a third of the sold CPUs in today's CPU market. We analyze the reverse-engineering tool for addressing functions published by Pessl et al. and find that it does not work with AMD CPUs, hindering reverse-engineering attempts and Rowhammer attacks on systems with AMD CPUs. In this paper, we introduce an approach to reverse-engineer the addressing functions of AMD CPUs, which facilitates future Rowhammer experiments on AMD CPUs.

1. Introduction

Memory is a substantial component in every information processing device, and due to the drastic increase in the requirements for performance and capacity, memory chips have become very dense. The OS, with the help of the memory management unit (MMU), translates virtual to physical

memory addresses. The memory controller then translates the physical memory addresses into a DRAM location, i. e., channels, DIMMs, ranks, and banks, using DRAM addressing functions.

Addressing functions have many fields of application. For example, researchers in IT security are interested in these functions to understand DRAM attacks better. One such attack is *Rowhammer*, in which an attacker can flip bits by rapidly accessing the content of nearby memory rows. An attacker can conduct better targeted Rowhammer attacks if the addressing functions are known. Pessl et al. [8] presented an approach to measure DRAM addressing functions entirely in software removing the requirement to perform physical probing. However, these addressing functions can also be used for performance optimization, such as application-aware memory channel partitioning [7] or variable page sizes [9] for more efficient row-buffer usage.

In contrast to AMD, Intel has not published the addressing functions of their CPUs. Therefore, the scientific community focused on reverse-engineering the DRAM addressing functions of Intel CPUs [8, 10, 5, 3]. AMD had a market share of 35.2 % (Intel had 62.8 %) in the market of x86 CPUs in Q3 2022 [4]. So, roughly one-third of x86 CPUs are manufactured by AMD. AMD published the DRAM banks' addressing functions in the Bios and Kernel Developer's Guide (BKDG) up to microarchitecture `16h` [1]. Beginning with microarchitecture `17h` (released in 2017 [2]), no BKDG was released, so the addressing functions of the DRAM banks are not publicly available anymore.

Our paper makes the following contributions:

- We present an adapted approach based on the one introduced by Pessl et al. [8] to reverse-engineer the addressing functions on AMD CPUs.
- We evaluate our approach on four CPUs with two different DRAM settings.
- We provide the addressing functions that we have found with our approach.
- We publish our AMD DRAM addressing function reverse-engineering tool[1].

[1] `https://github.com/iisys-sns/amdre` (link changed, since the camera-ready version had still the link to the anonymous git)

However, to the best of our knowledge, this paper is the first to focus on reverse engineering the addressing functions of AMD CPUs.

2. Backgrounds

This section briefly overviews how DRAM memory is organized and how the memory controller uses the addressing functions.

DRAM memory, in general, is organized in channels, DIMMs, ranks, and banks. The channel is a bus that connects the DIMMs with the CPU. In the case of a multi-channel memory architecture, DIMMs can be on different channels or share the same channel. A DIMM contains the actual DRAM chips, containing *banks*, which can be organized in groups called *ranks*. DRAM banks store data in cells consisting of capacitors and transistors organized in arrays of rows and columns. Additionally, a bank contains a *row buffer* that stores the entire row accessed last.

Since reading destroys the data saved in a row, the data has to be written back before the next row can be read. Writing the content of the row buffer back to the DRAM array before the next row is fetched (called *row conflict*) is significantly slower than if the correct row is already in the row buffer (called *row hit*).

The memory controller selects the bank by applying an XOR operation to the bits of the physical address masked with each of the addressing functions. The result of this operation is used as one addressing bit of the DRAM bank [8] for each DRAM addressing function. These functions split the available physical addresses equally to the physically available DRAM banks. If there are n_{banks} DRAM banks and n_{banks} is a power of two, there should be $\log_2(n_{\text{banks}})$ addressing functions. If n_{banks} is not a power of two, non-linear functions are used.

Since the memory controller is part of the CPU, different addressing functions exist on different CPUs. However, Pessl et al. [8] showed that the addressing functions also depend on the system's DIMM configuration. That means that the addressing functions can differ on systems with the same CPU.

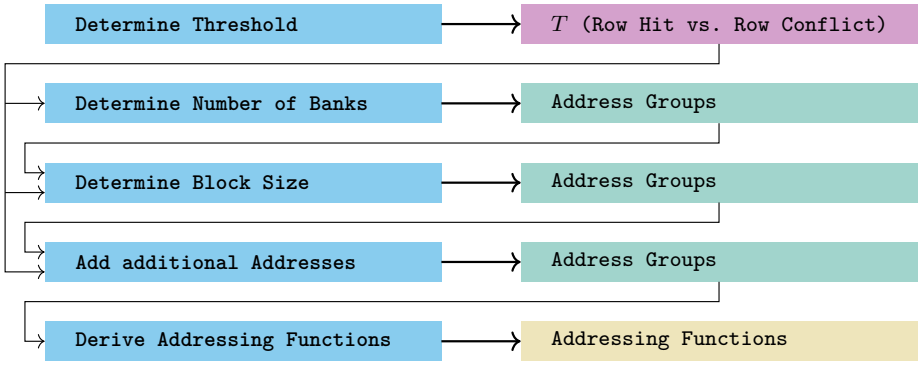


Figure 5.1.: Overview of the reverse-engineering process. Actions (shown on the left side) lead to results (shown on the right side) and require some of the results of previous actions. Different results are depicted in different colors. Actions that require and return address groups add more addresses to the existing groups.

3. Reverse-Engineering of DRAM Addressing Functions on AMD CPUs

This paper introduces an approach to reverse-engineer AMD CPUs' addressing functions that only depends on access to physical addresses and measurements of access timings. Figure 5.1 gives an overview of the procedure.

3.1. Determine the Threshold between Row Hit and Conflict

The first step is determining the threshold T between a row hit and a row conflict, which is required to measure whether addresses belong to the same bank. For that, we need to measure the access time of memory addresses. If the access is slow, we assume a row conflict was triggered. Thus, the memory addresses are in the same bank but in different rows. If the access is fast, we assume a row hit was triggered, and the memory addresses are located in different banks or the same row in the same bank. We need to determine a threshold T to distinguish between the two cases.

Our proof-of-concept allocates a 2 MiB transparent hugepage and measures the access time of the first and the second page (e. g., the first and the $4\,096^{\text{th}}$ byte) n (by default, 200) times to rule out noise. We use the `rdtscp`

instruction to measure the access time. To avoid that the CPU caches the results, our proof-of-concept uses the `clflush` instruction between measurements to remove the data from the cache. Then, the proof-of-concept computes the average by dividing the measured access time by n. Next, we repeat the measurement for all other pages within the transparent hugepage, group all measurements by access time and search for a gap in the access times, which should be between row hits and conflicts. If the gap is found, the middle of the gap is used as the threshold value.

This measurement is repeated for multiple transparent hugepages to increase the accuracy. The number of transparent hugepages can be configured and is set to 21 by default. The median of all measurements is used as the threshold.

3.2. Measuring the Number of Banks

In this step, we measure the number of banks by grouping addresses with row conflicts. Addresses that end up in one group belong to the same bank. Initially, there are no groups. We take every $4\,096^{\text{th}}$ address, i. e. one per 4 KiB page, and compare it with x randomly selected addresses of each group. If the address does not fit in a group or there are no groups, this address goes into a new group. If a group has fewer than x addresses, the address is compared to all addresses in this group. The comparison works as follows: When the access time (measured the same way as described in Section 3.1) is higher than the threshold determined before, there is a row conflict, and the address is at the same bank as the other addresses in the group; otherwise, there is a row hit. When multiple groups have access times longer than the threshold, which might occur due to measurement errors, the group with the longest access time is used. We set $x = 9$ in our experiments which worked best for us, but this parameter can be changed via a command line option.

At least two 4 KiB pages fit into one row on x86_64 systems, assuming a size of 8 KiB for a row. As a 4 KiB page can span multiple banks and rows, it is possible to fit parts of more than two 4 KiB pages into one row on one bank [8]. Thus, comparing two addresses that are located in the same row will produce a row hit even if they belong to the same bank. To avoid this problem, we regroup the initial groups. We iterate through the groups, removing each one after another, and try to add the addresses that belong to the removed group to the remaining groups. If one of the remaining

groups matches (e. g., a row conflict occurs), our proof-of-concept adds the address to that group. Otherwise, our proof-of-concept adds the address to a new group. This regrouping step is repeated until the number of groups is plausible, meaning it is a power of two. Afterwards, the number of groups is fixed, e. g., no new groups are created by the following steps.

3.3. Detecting the Block Size

The next step is to detect how many consecutive memory addresses are located in the same bank. This *block size* helps us determining which bits we can ignore when inferring the addressing functions. Additionally, it decreases the number of addresses we need to group, speeding up the process.

First, we guess the block size, starting with 4 096 B. Then, we count the number of consecutive memory addresses for the guessed block size. When we start with 4 096 B, we need to find memory addresses at offsets of `0x1000` ($\widehat{=}$ 4096). If we do not find consecutive memory addresses, we divide the block size by two and try again. The next block size would be 2 048 B. We already found the number of banks, but since we halved the block size, we have more addresses, which we need to sort again into the groups. That is the case since only every $4\,096^{\text{th}}$ address was grouped before. Now, every $2\,048^{\text{th}}$ address has to be grouped. In contrast to the grouping described before, no new groups are created at this stage. When an address does not fit into any existing group, it is discarded.

If the number of consecutive memory addresses is one, the block size is smaller than or equal to the currently guessed block size. If it is two, the system's block size is twice the currently guessed block size, so the block size is known in this case. Therefore, we must halve the block size until the number of consecutive memory addresses is two. We will repeat this process until we find the block size or reach 64 B, which is the size of one cache line.

3.4. Add Additional Addresses to the Groups

When we found the block size, we can allocate more transparent hugepages and add more memory addresses to the groups. This procedure is similar to the one described in Section 3.2, except that we will not create new groups and use the measured block size. If an address cannot be added to any

group, e. g., due to measurement errors or noise in the system, we discard the address. This optional step was not necessary for the tested systems in Section 4. However, there might be systems where the addressing function uses a bit before the 21^{st} bit. In that case, one transparent hugepage is insufficient to derive the full addressing functions. In that case, additional transparent hugepages are required.

3.5. Deriving the Addressing Functions

The final step is to derive the linear addressing functions. We implemented this step with multi-threading to reduce the time to derive the addressing functions. Each thread computes a *mask candidate* similar to Drama [8]. Initially, we try one bit (e. g., `0x01`) followed by the next larger combinations with the same number of bits. If this is no longer possible, two bits (e. g., `0x11`) followed by the next larger combinations with the same number of bits are tried. This is repeated until max_mask_bits bits are set. Thus, all possible masks with $1 \leq n \leq$ max_mask_bits are computed in total. We set the max_mask_bits $= 7$ by default to limit the number of candidates and, thereby, the time required for mask detection. Each mask is then checked to see whether it is valid based on the following criteria:

- The function yields the same results for all addresses within the same group for all groups.
- The function splits the groups equally, e. g.,, it provides the results 0 and 1 for 50 % of the the groups.
- The mask is not equivalent to a simpler mask. This is analyzed by toggling the ones in the mask to zeroes in all possible combinations. If one of the modified masks yields the same results, the current mask is equivalent to a mask with fewer bits and, therefore, does not have to be added.

After the threads are done, the detected masks are unified since the algorithm described above does also detect linear combinations of other masks. Finally, it is verified that the number of detected masks equals $\log_2 n_{banks}$. If that is the case, the detected masks are returned.

3.6. Runtime Estimation

The runtime of the approach described before depends on multiple parameters. We divide the reverse engineering process into three stages: In the first initialization stage (S1), our proof-of-concept measures the threshold between row hit and conflict. In the grouping stage (S2), our proof-of-concept groups memory blocks based on their timings when accessed alternatingly. In the last stage (S3), our proof-of-concept derives the addressing functions with multiple threads.

First, we look at the number of banks n_b depending on the number of addresses n_a. In stage S2, when $n_a \leq n_b \cdot x$, the runtime does not depend on n_b. When $n_a \geq n_b \cdot x$, the runtime has a linear dependency to n_b. Each address is compared to at most x (9 by default) randomly selected other addresses from the same bank. Therefore, if there are more than x addresses in the bank, the number of comparisons for each new address is limited to x per bank.

Next, we look at the number of transparent hugepages n_t. For both stages, S1 and S2, the runtime depends on n_t linearly. The block size n_s influence stages S1 and S2 inverse proportionally. If our proof-of-concept is run with multiple threads n_c, the last stage S3 depends inverse proportional on n_c.

Regarding the number of bits set at most in the mask candidates n_m: For S3, we need to sum up the binomial coefficient since all combinations with $1 \leq x \leq n_m$ bits are selected.

See Section 4 for runtime measurements during the performed experiments.

4. Experimental Evaluation

Table 5.1.: Results of the reverse-engineering tool introduced in this paper.

CPU	n_D	n_B	Blocksize	Addressing Functions (Bit Mask)	Time
AMD Ryzen 9 5950X	2	64	64 B	0x003fc0, 0x004100, 0x008000, 0x070000, 0x090000, 0x120000	2 673 s
	1	32	64 B	0x004000, 0x048000, 0x090000, 0x103fc0, 0x138000	8 048 s
AMD Ryzen 9 3900X	2	64	64 B	0x003fc0, 0x004100, 0x008000, 0x070000, 0x090000, 0x120000	2 675 s
	1	32	64 B	0x004000, 0x048000, 0x090000, 0x103fc0, 0x138000	16 201 s
Intel Core i9-10900K	2	32	64 B	0x004080, 0x01b300, 0x048000, 0x090000, 0x120000	742 s
	1	16	8 192 B	0x002000, 0x024000, 0x048000, 0x090000	18 s
Intel Core i7-4800MQ	2	32	64 B	0x00f380, 0x020000, 0x044000, 0x087380, 0x130000	3 644 s
	1	16	8 192 B	0x022000, 0x044000, 0x088000, 0x110000	58 s

Table 5.2.: Results of the reverse-engineering tool introduced by Pessl et al. [8].

CPU	n_D	n_B	Blocksize	Addressing Functions (Bit Mask)	Time
AMD Ryzen 9 5950X	2	64	64 B	(none)	131 s
	1	32	64 B	0x10290000, 0x80, 0x20010200, 0xc100200, 0x22204000, ... (14)	555 s
AMD Ryzen 9 3900X	2	64	64 B	0x40, 0x100, 0x400, 0x800, 0x800000, 0x21000, ... (12)	562 s
	1	32	64 B	0x8000, 0x4000000, 0x41000, 0x401000, 0x8001000, ... (20)	42 s
Intel Core i9-10900K	2	32	64 B	0x004080, 0x01b300, 0x048000, 0x090000, 0x120000, ... (6)	371 s
	1	16	8 192 B	0x002000, 0x024000, 0x048000, 0x090000, ... (5)	135 s
Intel Core i7-4800MQ	2	32	64 B	0x00f380, 0x00040, 0x044000, 0x088000, 0x110000	111 s
	1	16	8 192 B	0x022000, 0x044000, 0x088000, 0x110000, ... (5)	96 s

We reverse-engineered the addressing functions on several systems with different CPUs and multiple DIMM configurations. In addition, we performed these measurements for different proof-of-concepts. In order to ensure that our approach works on Intel CPUs as well, we run our proof-of-concept on several Intel CPUs in addition to the AMD CPUs. First, we analyze the existing Drama proof-of-concept [8]. Next, we evaluate the proof-of-concept introduced in this paper. Table 5.3 provides an overview of the systems used for experimental evaluation.

Table 5.3.: Systems used for experimental evaluation

CPU	DRAM type	Number of DIMMs
AMD Ryzen 9 5950X	DDR4	1, 2
AMD Ryzen 9 3900X	DDR4	1, 2
Intel Core i9-10900K	DDR4	1, 2
Intel Core i7-4800MQ	DDR3	1, 2

As the approach shown by Pessl et al. [8] is similar to our approach, it is used for evaluation. Their proof-of-concept returns all reverse-engineered addressing functions (might be more than addressing functions on the system) and their probability. Afterwards, the user has to guess which of the returned addressing functions are the correct ones. Therefore, the $\log_2(n_B)$ most probable addressing functions are listed for systems with n_B banks in the evaluation. Additionally, we provide the total number of identified addressing functions. The proof-of-concept was executed on several systems with AMD and Intel CPUs. Table 5.2 shows the results of our measurements.

The number of DIMMs in Table 5.2 is depicted as n_D and the number of DRAM banks is depicted as n_B. Not all identified addressing functions are listed due to space limits, but only the most probable ones. We provide the total number of addressing functions if more addressing functions were identified.

The proof-of-concept introduced in this paper was executed on the same systems with AMD and Intel CPUs. The configuration of the systems significantly affects the runtime (see Section 3.6 for detailed estimation of runtime impacts). The results of the measurements are shown in Table 5.1.

It should be noted that both the proof-of-concept by Pessl et al. [8] and our approach identified addressing functions on systems with AMD CPUs. The addressing functions reverse-engineered on the Intel systems are the same for both tools with the exception of the i7-4800MQ where Drama did not find all required sets. However, the ones on AMD systems are different. The addressing functions derived by Drama were not stable on the AMD systems, e. g., different when the tool was executed multiple times. Our tool took longer to execute but was more stable than Drama. Therefore, we conclude that Drama does not run stable on systems with AMD CPUs. However, we depict the addressing functions identified by Drama for completeness.

5. Limitations

Like prior approaches [8], our reverse-engineering approach introduced in this paper only works on systems with linear DRAM addressing functions, i. e., the total number of DRAM banks is a power of two. Moreover, since it is exclusively based on timings, it is impossible to get further semantic information related to the detected addressing functions, e. g., determining whether a detected function addresses a channel, rank, DIMM, bank group, or bank. Our proof-of-concept needs root privileges to get the physical addresses mapped to virtual addresses using `/proc/self/pagemap`. Since the addressing functions can only be applied to physical addresses (or parts of virtual addresses that are directly mapped to physical addresses), the restriction with the root privileges should not be a problem in typical application scenarios.

6. Related Work

In 2014, Kim et al. [6] analyzed the effect that bits in DRAM flipped (e. g., a 1 changed to a 0 or vice versa) when spatially nearby memory locations were repeatedly accessed. In addition, the authors reverse-engineered addressing functions on the Intel CPUs they used in a way. They found that the two selected addresses should have a distance of 8 MiB to be on the same bank.

Later, Pessl et al. [8] introduced a more generic approach to reverse-engineer addressing functions without requiring physical access, by using

timing instead. The primary approach is to measure the time a specific number of alternating accesses to two addresses takes when the accesses are not cached. If that access time is considered *high*, there is a row conflict, and both addresses are on the same bank in different rows. Otherwise, there is a row hit, and the addresses are either on different banks or on the same bank in the same row. Based on the conflicts, addresses are grouped, mapping to banks. Afterwards, the addressing functions can be derived using the physical addresses in the groups. The authors evaluated their approach on several systems with Intel CPUs by physically probing the accessed components and comparing the results of the reverse-engineered addressing functions to the probed values. They showed that different addressing functions are used depending on the configuration of DIMMs in the systems. In contrast to their tool, our tool measures the number of DRAM banks, so it does not have to be specified manually and can be used as additional sanity check. Additionally, we do not select addresses randomly out of an address pool but group entire 2 MiB transparent hugepages. We modified the measurements of the timings to get more precise results and reduce the impact of noise. However, this approach slows down the measurements, so our tool brings more stable results at the cost of a higher runtime.

In 2018, Barenghi et al. [3] showed that the addressing functions used by the memory controller depend on the configuration of DIMMs in the system. Their approach is similar to the one showed by Pessl et al. [8].

Two years later, Wang et al. [10] combined the approach introduced by Pessl et al. [8] with additional knowledge about the DIMM configuration and addressing functions in general. Thereby, they were able to reverse-engineer DRAM bank addresses significantly faster. Additionally, their approach gave information about the parts of addresses used to address rows and columns within the same bank. The authors claimed to open-source their tool, which seems not have happened until now. Upon a request we send to the authors we did not get any response.

In the same year, Helm et al. [5] introduced an approach that uses CPU performance counters instead of timings to group addresses by DRAM bank. Due to the usage of the counters, the results are more precise than the ones received via timing. Since the timers the authors are using are not supported on the AMD systems we used for our experiments, we can not evaluate their approach on AMD CPUs.

However, all of these publications focus on Intel CPUs, and none of them verified their approach on AMD. Our work closes this gap and extends over prior techniques.

7. Conclusion and Future Work

In this paper, we introduced a new approach extending the one by Pessl et al. [8]. We evaluated the proof-of-concept from prior work [8] for reverse-engineering the addressing functions on systems with AMD and Intel CPUs. Additionally, we evaluated our new approach on these systems and showed that prior approaches did not yield good results on AMD-based systems.

In the future, the scientific community should evaluate techniques to reverse-engineer DRAM addressing functions further on AMD CPUs to better understand both, what DRAM addressing functions on AMD in general look like and which patterns they follow, and which corner cases on different microarchitectures limit the applicability of reverse-engineering techniques.

Acknowledgment

We thank the Deutsche Forschungsgemeinschaft (DFG) for their support and Daniel Gruss for his review and fruitful discussions, which influenced the work of this paper. This work was funded by the DFG under grant number 503876675.

References

[1] AMD Developer Guides, Manuals and ISA Documents. 2020. URL: https://developer.amd.com/resources/developer-guides-manuals/ (p. 57).

[2] AMD's Zen CPU is now called Ryzen, and it might actually challenge Intel. arstechnica, 2016. URL: https://arstechnica.com/gadgets/2016/12/amd-zen-performance-details-release-date/ (p. 57).

[3] Alessandro Barenghi, Luca Breveglieri, Niccolò Izzo, and Gerardo Pelosi. Software-only Reverse Engineering of Physical DRAM Mappings for Rowhammer Attacks. In: IVSW. 2018 (pp. 57, 67).

[4] Distribution of Intel and AMD x86 computer central processing units (CPUs) worldwide from 2012 to 2022, by quarter. statista, 2023. URL: `https://www.statista.com/statistics/735904/worldwide-x86-intel-amd-market-share/` (p. 57).

[5] Christian Helm, Soramichi Akiyama, and Kenjiro Taura. Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters. In: MASCOTS. 2020 (pp. 57, 67).

[6] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: SIGARCH (2014) (p. 66).

[7] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In: MICRO. 2011 (p. 57).

[8] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security Symposium. 2016 (pp. 56–58, 60, 62, 64–68).

[9] Kshitij Sudan, Niladrish Chatterjee, David Nellans, Manu Awasthi, Rajeev Balasubramonian, and Al Davis. Micro-pages: increasing DRAM efficiency with locality-aware data placement. In: SIGARCH. 2010 (p. 57).

[10] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. DRAMDig: A Knowledge-assisted Tool to Uncover DRAM Address Mapping. In: DAC. 2020 (pp. 57, 67).

6

Flipper: Rowhammer on Steroids

Publication Data

Martin Heckel and Florian Adamsky. Flipper: Rowhammer on Steroids. In: uASC. 2025

Contributions

Main author.

Flipper: Rowhammer on Steroids

Martin Heckel[1, 2], Florian Adamsky[1]
[1]firstname.lastname@hof-university.de
[1]*Institute of Information Systems (iisys) Hof University of Applied Sciences* Hof, Germany [2]*Institute of Information Security (ISEC) Graz University of Technology* Graz, Austria

Abstract

The density of memory cells in modern DRAM is so high that frequently accessing a memory row can flip bits in nearby rows. That effect is called Rowhammer, and an attacker can exploit this phenomenon to flip bits by rapidly accessing the contents of nearby memory rows. In recent years, researchers have developed sophisticated exploits based on this vulnerability, which enable privilege escalation on desktop computers, mobile devices, and even cloud systems without requiring any software vulnerability. However, rows are not equally vulnerable to Rowhammer. Therefore, an attacker has to massage the memory, for instance, with page table entry (PTE) spraying, to increase the chance of successful exploitation. More bit flips mean the attacks become easier and faster to conduct.

In this paper, we present Flipper, a Rowhammer amplification attack against DDR3, consisting of two components: CMP_{IST} exploits the `cmpsb` and `repe` x86 instructions to get DRAM access with higher frequency. CMP_{PAR} exploits the effect of hammering in multiple threads, which increases the number of bit flips found in a given time, as shown in previous work. As a result, we can increase the number of bit flips by a factor of 830 on the measured devices, even on systems featuring mitigation techniques, without using administrative privileges. We evaluate our technique on six DDR3 DIMMs. Although DDR3 memory has been superseded by DDR4 and DDR5 memory technologies, it is still widely used in devices that do not require frequent replacement, such as projectors, smart displays, servers, embedded devices, routers, and printers.

1. Introduction

DRAM stores data in memory cells consisting of capacitors and transistors. These DRAM cells are organized in arrays of rows and columns. A high density of memory cells is required to meet the market demands for storage capacity. Due to the high density, rapidly accessing a memory cell can affect spatially nearby memory cells. That means rapidly reading the content of memory rows can cause bit flips in adjacent memory rows. This vulnerability is known as Rowhammer [20].

Initially, the problem behind Rowhammer was known as a side-effect with little to no security implications [33]. In recent years, however, researchers developed sophisticated exploits based on Rowhammer. These exploits achieve, for instance, privilege escalation on desktop computers [34, 6, 5, 4, 14, 31, 25, 19, 15], mobile devices [38, 41, 4, 24, 22], and even on cloud systems [30, 3, 40, 37], all without a software vulnerability. All these exploits have something in common: the more bit flips are found, the easier and faster the exploits can be conducted.

One technique to amplify Rowhammer attacks is *multi-thread hammering*, in which multiple threads hammer different memory locations. Previous works used multi-thread hammering with mixed results. Some works [23, 13, 7] show that multi-thread hammering increases the number of bit flips. In contrast, Qiao and Seaborn [28] show that multi-thread hammering was much less effective than to single-threaded hammering, at least for DDR4 DIMMs with target row refresh (TRR).

Another technique that Kang et al. [19] introduced is called *bank-level parallelism*. When a program accesses memory locations in different banks, the memory controller parallelizes these accesses, even when accessed from a single-threaded program. They show that bank-level parallelism is quite effective for Rowhammer on DDR4 and increases the number of bit flips. However, their measurements show that it is not effective against DDR3.

Even though DDR3 memory is superseded by newer DDR4 and DDR5 memory technologies, it is still a relevant attack vector. Globally, a share of cloud systems still use DDR3, and a considerable number of consumer and office systems still use DDR3 [11]. An analysis at our institution confirmed this on a local scale as well, as we found many devices in use that still had DDR3 memory, e. g., projectors, smart screens, embedded devices, routers, and printers.

In this work, we show an effective amplification attack against DDR3. We combine multi-threaded hammering with bank-level parallelism and show x86 instructions that can be used to increase the memory pressure. Thereby, we get an increase in the number of bit flips found in a given time by a factor of 830. Overall our paper makes the following contributions:

1. We introduce CMP$_{IST}$, a Rowhammer amplification attack on DDR3 DRAM that exploits the x86 `cmpsb` and `repe` instructions.
2. With CMP$_{PAR}$, we reproduce the results of previous work [23, 13, 7] and verify that the hammering process can be parallelized.
3. We perform a systematic evaluation of both primitives, separately and in combination, and show that the number of bit flips on DDR3 systems can be increased by a factor of 830 on our systems, even on systems featuring the double refresh rate mitigation.
4. We publish FLIPPER, the tool used for evaluation[1].

Outline. Section 2 provides background information. In Section 3, we introduce two novel amplification attacks. Section 4 provides an experimental evaluation of both attacks. We discuss the security impact in Section 5. Section 6 lists possible countermeasures to mitigate the amplification effects. Related work is discussed in Section 7. We conclude in Section 8.

2. Background

This section briefly overviews the memory architecture and describes how Rowhammer works.

2.1. Memory Architecture

DRAM cells consist of capacitors and transistors organized in columns and rows, referred to as a DRAM array. The DRAM array, sense amplifiers, and the *row buffer*, containing the last row that was accessed, is called a *bank*. Multiple banks are placed across multiple chips, and multiple chips are organized in ranks. A DIMM consists of one or multiple of these ranks.

[1] `https://github.com/iisys-sns/Flipper`

Memory systems can have multiple buses that connect the CPU and the DIMMs. These buses are called *channels*.

When data from memory is accessed, the entire row is loaded into the row buffer. However, this operation destroys the original row's content in the DRAM array. Thus, the data has to be restored from the row buffer back into the row located in the DRAM array.

Additionally, the capacitors lose charge over time. Therefore, they must be refreshed regularly. The standards for DDR3 [16] and DDR4 [17] specify a refresh interval of 64 ms, while the standard for DDR5 [18] specifies a refresh interval of 32 ms for each DRAM cell.

2.2. Rowhammer

When two different rows in the same bank are frequently accessed, this results in multiple accesses to the actual DRAM array, involving the loading and restoring of the rows. Due to these accesses, charge leakage can occur in other physically adjacent rows—a phenomenon known as *Rowhammer*. An attacker can exploit this side effect to induce bit flips in memory. The accessed rows are referred to as *aggressor rows*, while the rows prone to bit flips are called *victim rows*.

Bit flips can only occur between two refresh operations of the rows because the cells' capacitors are restored during the refresh process. One mitigation technique for Rowhammer is to double the refresh rate, ensuring that a cell is refreshed every 32 ms instead of 64 ms [20]. Not all memory regions exhibit the same vulnerability to Rowhammer; consequently, the number of bit flips observed within a given period varies depending on the scanned memory region. Furthermore, DIMMs are not uniformly vulnerable to Rowhammer, even when they are the same model.

To exploit Rowhammer effectively, an attacker has to reverse-engineer the mapping process from virtual memory addresses to spatial memory locations, which is typically handled by the memory management unit (MMU) and the memory controller [27]. For modern CPUs, the mapping functions between physical addresses and spatial memory locations remain unpublished. Because the memory controller translates physical addresses to memory locations, an attacker must ascertain the physical addresses mapped to the virtual addresses used by processes. Alternatively, mechanisms that ensure proper alignment of virtual memory addresses, such as

transparent hugepage (THP), can be utilized. When the physical address (or a portion of it) is known, DRAM addressing functions can be utilized to determine the spatial location of the physical address. Although these functions are not publicly documented, several methodologies exist to reverse-engineer them [27, 39, 9, 4, 8, 15].

3. Rowhammer Amplification Attacks

In this Section, we introduce a novel Rowhammer amplification attack which we call Flipper, that increases the number of found bit flips on mitigated systems. Our amplification attack consists of two parts: CMP_{IST} and CMP_{PAR}.

3.1. Terminology

The Rowhammer amplification primitive based on specific x86 instruction as described in Section 3.2 is called CMP_{IST}. Our implementation of this approach, a standalone binary, is called MemPressureGen.

The Rowhammer amplification primitive based on parallel execution of Rowhammer on multiple banks as described in Section 3.3 is called CMP_{PAR}. Our implementation of this approach, a Rowhammer tool that includes features for addressing function reverse-engineering and supports multi-threaded execution of Rowhammer, is called HammerTool.

The combination of both, e. g., running HammerTool in one process and MemPressureGen in another process at the same time, is called Flipper. For experimental evaluation, we use RowhammerJS, a Rowhammer tool from Gruss et al. [6] in addition to our own implementation HammerTool.

3.2. cmp_{IST}: Exploiting the `cmpsb` and `repe` Instructions

The fundamental discovery for CMP_{IST} is, that the function `memcmp(...)`, shown in Listing 6.1 is using assembly instructions to optimize the comparison. The implementation uses the x86 `cmpsb` (compare byte strings) instruction in combination with the `repe` instruction, which repeats the `cmpsb` instruction as long as the `ECX` register is not zero and the `ZF` flag is set [12].

```
32 int memcmp(const void *s1, const void *s2, size_t len)
33 {
34   bool diff;
35   asm("repe; cmpsb" CC_SET(nz)
36       : CC_OUT(nz) (diff), "+D" (s1), "+S" (s2), "+c" (len));
37   return diff;
38 }
```

Listing 6.1: Source code of the function `memcmp` from the Linux Kernel v6.13-rc5 located in arch/x86/boot/string.c.

Both instructions are not new and were already available on the 8086 CPU [10]. In comparison, the ARM implementation of `memcmp` increases two pointers as long as the bytes to be compared are equal. As stated by Shirriff [35], these x86 instructions are faster than the implementation in assembly code. A quick benchmark[2] between the two implementations reveals that the x86 implementation is around 2.65 times as fast as the ARM implementation.

Our implementation of CMP$_{IST}$, MemPressureGen, allocates 1 024 MiB of memory and initializes all pages in the allocated memory area with the same random data. Thereby, identical data is compared and the `cmpsb` instruction keeps running. MemPressureGen compares every page with the first one and repeats this procedure in a loop until the process is terminated.

To amplify the Rowhammer attack, we run the native double-sided RowhammerJS exploit published by Gruss et al. [6] and start MemPressureGen as new process in parallel. With this approach, the number of bit flips found in a given time can be increased significantly, as we show in Section 4.

3.3. cmp$_{PAR}$: Parallel Hammering

Accessing data in DRAM can be parallelised, provided that the data is stored in different DRAM banks [21]. Previous work shows that parallel hammering can have positive [23, 13, 7, 19] and negative [28] impacts on the number of bit flips occurring.

[2]We measured the time to compare 1 000 000 pages of memory with identical content.

This is why we implemented a multithreading mode in our Rowhammer PoC called HammerTool. HammerTool is implemented in a way that each thread hammers a single DRAM bank. Therefore, each thread gets a list of all *items* that should be hammered for a single bank, where each item contains a list of aggressors that should be hammered and a list of victims that should be checked for bit flips after the aggressors were accessed. After the entire list is processed, the thread gets another list with all *items* that should be hammered for another bank. It is possible to manually specify the number of threads. By default, it uses n_threads = min(n_logical_cpus, n_memory_banks) threads on the victim's system. Thus, we are leveraging bank-level parallelism from Kang et al. [19].

The *items* described before are generated in a initialization phase that preceeds the multi-threaded hammering. In that phase, memory is allocated and grouped by banks either by using addressing functions (which requires elevated privileges) or by utilizing access time measurements. Afterwards, addresses from the same group with a specified distance searched depending of the submitted pattern. Thereby, it is possible to generate arbitrary statical patterns. The double-sided pattern with the aggressor mask `101` (aggressor row, free row, aggressor row) yielded good results and was used for the evaluation. During the search, all rows that are next to an aggressor row and not an aggressor row themselves are handled as victim rows. In the end, a list of *items* that match the submitted aggressor mask is returned for the bank. When the *items* were generated for all banks, multi-threaded Rowhammer as described above is started.

4. Experimental Evaluation

4.1. Experimental Setup

Our experimental setup consists of two laptops, a ThinkPad T540p and a ThinkPad X230T. On both, the latest BIOS version[3] is installed. According to the BIOS changelog, both systems contain a mitigation for the "*risk of security vulnerability related to DRAM Row Hammering*" [2, 1]. The changelog, however, did not specify which mitigation was applied. Therefore, we measured the refresh interval on both systems before and after the BIOS update and confirmed that they applied the double refresh

[3] At the time of writing, the Thinkpad X230T had version 2.75 (2019-10-04) and the Thinkpad T540p had version 2.38 (2020-05-19) installed.

rate mitigation. Details of this measurement can be found in Section 4.2. All systems run Arch Linux. Table 6.1 shows the hardware specification of our setup.

Table 6.1.: Hardware specification of our setup.

System	CPU	DIMM	Capacity	Kernel
ThinkPad X230T	i5-3320M	`M1`	4 GiB	`5.11.16`
ThinkPad T540p	i7-4800MQ	`M4`	4 GiB	`5.16.16`

We use several DIMMs for the experimental evaluation which are named `M1`–`M6`. If not stated otherwise, the experiments are done with the DIMM listed in Table 6.1 for the according systems.

4.2. Measuring the Refresh Rate

By default, DDR3 DRAM is refreshed at a rate of 64 ms [16]. The refresh is done in batches containing multiple rows [26]. There are 8 192 of these batches. So, one batch is refreshed every $\frac{64\,\text{ms}}{8\,192} \approx 7.8\,\mu\text{s}$. With a double refresh rate, one batch is refreshed every $\frac{32\,\text{ms}}{8\,192} \approx 3.9\,\mu\text{s}$. During a refresh, there are no accesses to the DRAM array. For this reason, accesses during a refresh are slower because they are only executed after the refresh is done.

The refresh rate can be measured by accessing an address multiple times from DRAM by using the `clflush` instruction to flush it from the CPU cache before accessing it. When measuring the time of the access, there are fast and slow accesses. When there is a row hit and no refresh is running, the access is fast. When data from another row is requested between two accesses, there is a row conflict and the access is slow. When any rows are refreshed, the access is slow as well.

Next, the duration of the accesses can be analyzed. Slow accesses are expected at a regular basis: When they are every 7.8 µs, the system has a refresh rate of 64 ms. When the are every 3.9 µs, the system has a refresh rate of 32 ms.

4.3. Experimental Methods

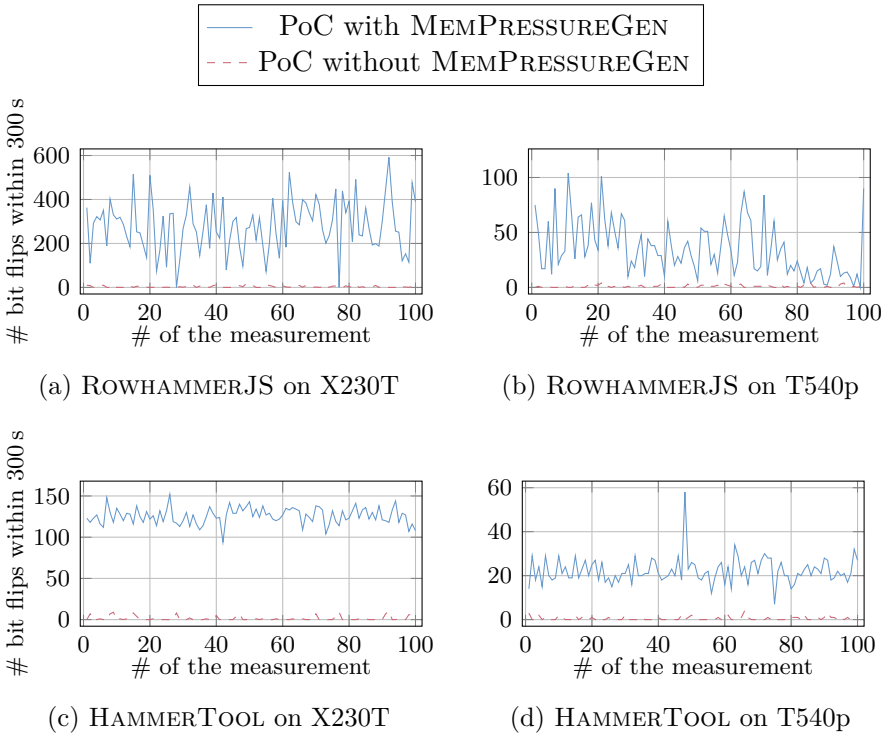
We use the native implementation of RowhammerJS [6] and HammerTool for our evaluation. The first one is an existing PoC for Rowhammer and the latter one is the PoC we implemented. Both tools use dynamic memory allocation and, therefore, scan different memory areas every time they are started. Therefore, we repeat the measurements multiple times and compute the average value and confidence intervals, since the occurrence of the bit flips is a statistical process as shown in the following subsections. Most of the measurements were repeated 100 times and we show the average and confidence intervals of 99 %. Most single measurements run for 300 s. If the number of measurements, confidence interval, or runtime differs for an experiment, it is stated there.

We also tried to use the implementation of hammertime introduced by Tatar et al. [36] as the authors state that their implementation is able to find orders of magnitude more bit flips on their test system. However, we were unable to identify any bit flips using hammertime on our test systems. Since this would require a detailled analysis of the root cause, we skipped hammertime and only use RowhammerJS and HammerTool.

4.4. Evaluation of $\mathbf{cmp}_{IST}$

In this experiment, we run Rowhammer attacks with HammerTool in single-thread mode and RowhammerJS. In parallel, we execute MemPressureGen as described in Section 3.2. Figure 6.1 depicts the amount of bit flips found by RowhammerJS and HammerTool within 300 s with and without MemPressureGen running.

Without MemPressureGen, HammerTool finds approximately 1.32 bit flips in 300 s on average on the X230T. With MemPressureGen, it finds approximately 125 bit flips in the same time. So, the usage of MemPressureGen increases the amount of bit flips by a factor of 94.70. RowhammerJS finds approximately 3.45 bit flips without MemPressureGen running in parallel. When MemPressureGen is running in parallel, approximately 281 bit flips are found on the X230T. MemPressureGen increases the amount of bit flips by a factor of 81.45. So, the usage of MemPressureGen lead to an increase in the amount of bit flips by factor 88.08 on the X230T.



(a) RowhammerJS on X230T

(b) RowhammerJS on T540p

(c) HammerTool on X230T

(d) HammerTool on T540p

Figure 6.1.: Number of bit flips measured within 300 s with and without MemPressureGen running in parallel.

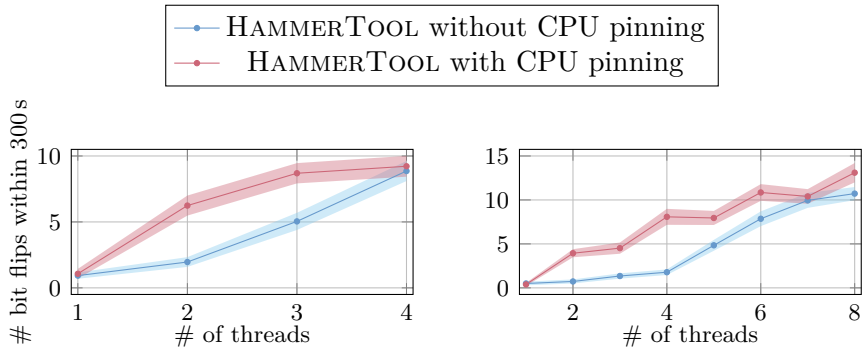
HammerTool finds approximately 0.5 bit flips in 300 s on average on the T540p when MemPressureGen is not running in parallel. With MemPressureGen, it finds approximately 22.28 bit flips in the same time. The usage of MemPressureGen increases the amount of bit flips by a factor of 44.56. Without MemPressureGen running in parallel, RowhammerJS finds approximately 0.91 bit flips. In the same time, there are approximately 35.09 bit flips found when MemPressureGen is running. So, the usage of MemPressureGen increases the amount of bit flips by a factor of 38.56. The usage of MemPressureGen leads to an increase in the amount of bit flips by factor 36.83 on the T540p.

Note: We identified a bug in the flush mechanism after performing all experiments. Due to that bug, HammerTool only flushed the first cache line of a victim before comparing the entire victim. Therefore, depending on other processes running on the same system, parts of the victim

might be read from the CPU cache instead of DRAM. A short evaluation showed that this yields approximately two times the number of bit flips when used without MemPressureGen running in parallel. This explains the difference in the number of bit flips found by HammerTool and RowhammerJS and shows that our measurements are rather conservative.

4.5. Evaluation of cmp_{PAR}

Because Rowhammer requires accesses to the DIMMs, the speed of the hammering process is limited by the speed of the DIMMs. We show that it is possible to hammer multiple memory locations in parallel as described in Section 3.3, if they are located at different banks. The number of bit flips found in a given time increases with the number of threads used for hammering. Figure 6.2 depicts the amount of bit flips found by HammerTool in relation to the number of threads used for hammering with and without pinning the threads to logical CPU cores.



(a) X230T with 4 logical cores

(b) T540p with 8 logical cores

Figure 6.2.: Number of bit flips found within 300 s depending on the number of threads used by HammerTool without MemPressureGen running. The average values of 100 measurements are depicted for each number of threads with confidence intervals of 99 %. In general, the number of bit flips increases with the number of threads.

On both systems, the graphs without CPU pinning can be divided into two parts: The first one in the range $1 \leq x \leq \frac{n}{2}$, where x is the number of threads and n is the number of logical cores. The second one in the range $\frac{n}{2} \leq x \leq n$. The second part of both graphs has a stronger slope than the

first one. This is due to the fact that the scheduler tries to balance load among logical cores. When there are $\leq \frac{n}{2}$ threads, the scheduler can shift all threads at the same time. This leads to the threads "jumping" across the cores[4]. When there are $> \frac{n}{2}$ threads, at least one thread has to keep running because not all of them can be shifted around the cores at the same time. We assume that this effect leads to fewer shifting in general and, thereby to a more efficient usage of processing time.

In contrast to that, the graphs with CPU pinning have a stronger slope in the beginning. This can be explained with the fact that the hammering threads do not "jump" across the cores and that they are not interrupted by other threads so often because these are put to the "free" cores if possible. With an increase in the number of hammering threads, there are less possibilities to schedule other threads to other cores (in the case where $x = n$, there is no possibility to do so), so the slope decreases with the number of threads.

On the T540p, it can be seen that the number of bit flips increases strongly at every second number of threads when CPU pinning is enabled (see Figure 6.2b). This effect does not occur on the X230T (see Figure 6.2a). The T540p has a CPU with Haswell microarchitecture and the X230T one with Ivy Bridge Microarchitecture.

The memory subsystem in Ivy Bridge can handle two 16 byte loads and one 16 byte store per cycle. For Haswell, two 32 byte loads and one 32 byte store can be performed per cycle [32]. Ivy Bridge has a L2 $\rightarrow$ L1 bandwidth of 32 bytes per cycle while Haswell has a L2 $\rightarrow$ L1 bandwidth of 64 bytes per cycle [32]. However, for both microarchitectures, the L2$\rightarrow$L3 bandwidth is 32 bytes per cycle. One physical core and the L1 and L2 caches are shared between two hyperthreads on both systems.

On Ivy Bridge, two hyperthreads can perform one 16 byte load each in one cycle resulting in 32 bytes being loaded, which is also the bandwidth of L2 $\rightarrow$ L1 and L3 $\rightarrow$ L2. So, running two hyperthreads saturates the bandwidth of the sytem.

In contrast to that, on Haswell, two hyperthreads can perform one 32 byte load each in one cycle resulting in 64 bytes being loaded. The L2 $\rightarrow$ L1 bandwidth is 64 bytes per cycle as well, so this saturates the L2 $\rightarrow$ L1 bandwidth as well. However, the L3 $\rightarrow$ L2 bandwidth is only 32 bytes

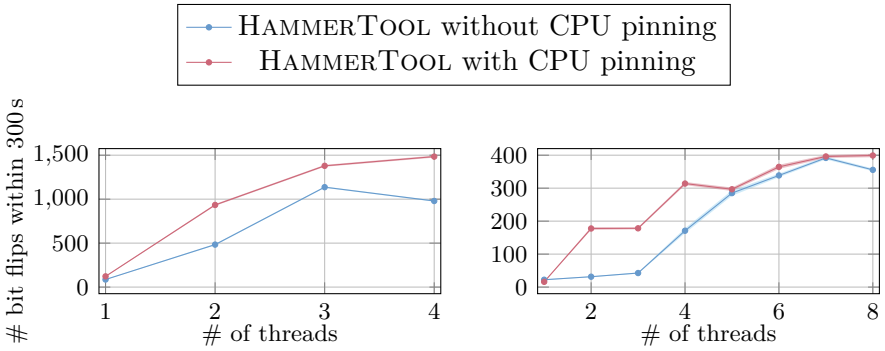
[4]This behaviour can be observed with an interactive process-viewer during the experiments

per cycle and thereby the bottleneck for two hyperthreads running on the same core.

On Haswell, one hyperthread per core already saturates the L3 → L2 bandwidth. When a second hyperthread is added, it does not increase the memory access speed. Therefore, an increase happens only when a new hyperthread is created on a new physical core (not already used for hammering), which happens only for every second hyperthread. However, this is only a hypothesis.

4.6. Evaluation of Flipper — Combining cmp_{IST} and cmp_{PAR}

Both Rowhammer amplification attacks introduced in this paper can be used at the same time leading to a substantial increase in the amount of bit flips found in a given time. To evaluate this, we repeat the experiment from Section 4.5 with MemPressureGen running in parallel. Figure 6.3 shows the amount of bit flips found with cmp_{IST} running at the same time.



(a) X230T with 4 logical cores (b) T540p with 8 logical cores

Figure 6.3.: Number of bit flips found by HammerTool in 300 s with MemPressureGen running in parallel. An average value from 100 measurements is depicted for each number of threads with confidence intervals of 99 % which are very small.

The result of the experiment from Figure 6.3 shows that cmp_{IST} can be combined with cmp_{PAR} and, thereby, find more bit flips in 300 s on the X230T. The number of bit flips peaks in the graph without CPU pinning

at 3 threads on the X230T, because MemPressureGen is running in a separate process which leads to one logical core to run at full load. Because the X230T has 4 logical cores, there are 3 logical cores idling when MemPressureGen is running. So, they can be used to run 3 threads of hammering. Each of those threads brings one logical core to full load. When there are more CPU-intensive threads than logical CPUs on the system, the operating system has to reschedule them. This leads to a decrease in the amount of bit flips found because the hammer threads are interrupted by the scheduler frequently. The same applies for the T540p which has 8 logical cores and, therefore, a maximum at 7 threads without CPU pinning.

With CPU pinning, the effect described in Section 4.5 can be seen again. In contrast to the case without CPU pinning, the number of bit flips found within 300 s increases slightly from $x = n - 1$ to $x = n$ threads. This is the case because the threads are pinned to the CPU cores and, therefore, not scheduled to other cores (there is no "jumping"). However, there is only one thread at a time interrupted by MemPressureGen. As the other threads are not interrupted at that time, it can be assumed that the number of bit flips is at least at the level of the measurement with $x = n - 1$ threads. The slight increase is because the threads get some computation time even if they are interrupted.

4.7. Comparison of Attacks based on Unique Bit Flips

The previous experiments have shown that CMP_{IST} leads to a significant increase in the number of bit flips found in a given time. But what about unique bit flips; can the usage of MemPressureGen lead to bit flips being discovered that would otherwise not be found? When scanning a memory area (e. g., one THP) for the first time, there are more bit flips found when using MemPressureGen. To clarify if using MemPressureGen leads to more unique bit flips, the same memory area is scanned multiple times. We define unique bit flips based on the following parameters:

Aggressor rows that were hammered when the bit flip was triggered;

Victim row that contained the actual bit flip;

Offset in the victim row, which specifies the byte that contained the bit flip;

Flip mask which specifies the bit/bits that flipped within the byte; and

Flip direction which specifies if a bit flipped from 1 to 0 or vice versa.

If at least one of the parameters is different, two bit flips are considered to be distinct. For this experiment, bit flips are only considered if they differ from all other bit flips found within the same measurement.

In this experiment, one THP is allocated and scanned x times without MemPressureGen. Afterwards, the amount of absolute and unique bit flips is counted and reset. Next, MemPressureGen is started and the measurement is repeated x times on the same THP. Then, the number of absolute and unique flips is counted again. For each value of x, HammerTool was started 10 times and the average value is used as result. Figure 6.4 depicts the amount of unique bit flips.

The measurements on both systems depicted in Figures 6.4a and 6.4b show that there is a limited growth within the scope of the measurements. This behaviour applies to the amount of bit flips found with and without MemPressureGen. However, within 60 measurements of the same memory area, the amount of bit flips measured without MemPressureGen did not reach the amount of bit flips with MemPressureGen. This means that the usage of MemPressureGen leads to more unique bit flips even when measuring multiple times.

Afterwards, the ratio between the number of unique bit flips found with and without MemPressureGen is calculated. This ratio is shown in Figure 6.4c and 6.4d. It can be seen that the ratio decreases with the amount of repeating measurements. However, even when the measurements are repeated 60 times, the ratio is still approximately 2 on the X230T and approximately 10 on the T540p. So, CMP_{IST} leads to more found bit flips even when the same memory area was scanned 60 times.

On the X230T, the factor is about 13 for $x = 1$. The difference between the factor of 13 in this measurement and the factor of 90 in the measurement shown in Section 4.4, is due the parallelized mode of HammerTool and warmup measurement errors. We assume that the sudden increase in Figure 6.4b values at $x = 20$ is a background process that was started. However, it did not have a significant effect to the ratio, meaning that it affected both measurements (with and without MemPressureGen) the same way.

4.8. Relation between Measurement Time and Bit Flips Found

After the measurement of repeatedly scanning the same memory areas as described in Section 4.7, we measure the amount of bit flips found with and without the usage of MemPressureGen within a given time. Figure 6.5 depicts the results of the measurement.

When MemPressureGen is used (see Figure 6.5a and Figure 6.5b), the number of bit flips found in a given time is linearly correlated that time. So, doubeling the measurement time results in the double number of bit flips within the range of this experiment. This behavior can be seen on both systems, the X230T (see Figure 6.5a) and the T540p (see Figure 6.5b).

In contrast to that, when MemPressureGen is not used (see Figure 6.5c and Figure 6.5d), the number of bit flips found in a given time does not seem to be linearly correlated to that time. However, when looking at the error bars, it is likely that they are linear as well and the difference from the linear function are just fluctuations within the depicted error range.

4.9. Amplification Factor on Systems with Double Refresh Rate

To estimate the effect of Flipper in real-world scenarios, we measure the amplification factor when combining CMP_{IST} and CMP_{PAR} in contrast to single-thread hammering without CMP_{IST} on both systems with multiple DIMMs. The amount of bit flips found in 300 s with and without Flipper are shown in Figure 6.6.

Based on the results shown in Figure 6.6, the highest amplification factor was 4 771 for `M3` on the T540p. The lowest amplification factor was 231.64 for `M1` on the X230T. In average, the attacks introduced in this paper lead to an amplification factor of approximately 1 675.

Consequently, despite the fact that the base implementation of HammerTool finds about half the amount of bit flips in a given time as RowhammerJS, as shown in Section 4.4, Flipper effectively yields 837.5 times more bit flips in the same time frame. This is an improvement over the state of the art by almost 3 orders of magnitude.

It should be also noted that no bit flips were found on `M6` without Flipper on both systems. When Flipper was used, both systems were affected.

5. Discussion on the Security Impact

In this Section, we would like to discuss what an increased number of flips means regarding security.

First, it reduces the time an attacker needs to find the appropriate bits, e. g., bits belonging to a PTE. A bit flip in a PTE's physical page number can give an attacker access to different memory pages [34]. The longer an attacker needs to find the appropriate bits, the greater the chance of getting caught. The same applies to other exploitation approaches, e. g., flipping bits in opcodes of binaries within the page cache, as Gruss et al. [5] showed. However, that attack overall took 95.5 h. From that time, 26.2 h were used to find bit flips. With our attack that time could have been significantly reduced. The same is true for finding the correct flip to attack a page cross-VM when Kernel Samepage Merging is enabled, as shown by Razavi et al. [30].

Second, since our attack also finds bit flips that would not have been found without our attack, the chance of finding bit flips at the right places is higher. Our experiments show that our attack finds bit flips that other tools, e. g., RowhammerJS, did not find. Therefore, the chances are higher to find bit flips that are security-relevant.

Third, the BIOS update enabled the double refresh rate mitigation on both test systems. So, all our experiments were run with this mitigation enabled. Thus, our presented attacks help to bypass this mitigation technique.

6. Countermeasures

Defending Rowhammer attacks is a difficult task, particularly since it is a hardware side-effect. Since the `cmsp` and `repe` x86 instructions are available in user space, there is no easy countermeasure. The microcode implementation is not public, this is why we can only speculate about countermeasures.

One possible countermeasure would be to adjust the microcode implementation to behave like a normal memory comparison internally. However, that would introduce a performance decrease of around factor 2.65 as described in Section 3.2. Increasing the refresh rate further, such as 2–4 times [29], would be possible, but that comes with disadvantages. A higher

refresh rate would mean more power consumption and less performance without a guarantee to mitigate Rowhammer completely.

7. Related Work

Kim et al. [20] were the first that described the technical details behind Rowhammer. One year later, Seaborn [34] published an exploit based on Rowhammer, which can be used for local privilege escalation. They published their tool for automated Rowhammer testing. This tool uses a probabilistic approach which means that it is not required to know the mapping between memory addresses and their locations.

Gruss et al. [6] described the approach of executing a Rowhammer attack without the usage of the `clflush` instruction by evicting cache lines from the CPU cache instead. The source code published in the scope of that paper included a native component with `clflush` as well. That tool makes it possible to find bit flips when the addressing functions of the memory controller are known. We use RowhammerJS to evaluate our tool.

In 2016, Pessl et al. [27] described an approach to automatically reverse-engineer the addressing functions of the memory controller by using time-based measurements. We have implemented their time-based approach in HammerTool. Additionally, we compare the addressing functions reverse-engineered by HammerTool to the ones reverse-engineered by Drama. Both tools identify the same addressing function on our test systems.

In the same year, Razavi et al. [30] described an approach to exploit rowhammer in a virtualized environment. To get the correct address mappings, they use THPs, which we also do. This approach is used to get contiguous physical memory and, thereby, remove the need to know the physical addresses.

There are several publications that use parallelized hammering. Some of them stated that it has a positive impact on the number of bit flips found in a given time [23, 13, 7, 19]. Others state that is has a negative impact [28]. We implemented and evaluated the approach and showed that it has a positive impact.

Ridder et al. [31] introduced rowhammer on DDR4 DRAM in JavaScript from the browser. They described an novel approach that exploits DDR4

memory with implemented mitigation against Rowhammer anyway. They increased the hammering pattern by using a double pointer chase; in contrast to our work, we use multithreading and special instructions to increase the hammering frequency.

In 2022, Jattke et al. [14] also explored fuzzing of hammering patterns to find more effective patterns. These patterns are exploitable on specific DRAM modules, bypassing defenses that target only specific hammering patterns.

In 2024, Kang et al. [19] showed that it is possible to perform Rowhammer attacks parallelized at DRAM bank level. They used the effect that the memory controller parallelizes serial memory access instructions. With DDR4, they inspected a significant increase in the number of bit flips. However, they were unable to reproduce this behaviour on DDR3 where the number of bit flips even decreased with their parallelization approach. Kim et al. [20] reported threshold number of activations of at least 139 k within one t_{REFI} for DDR3. In contrast, Frigo et al. [4] showed that 45 k activations within one t_{REFI} are sufficient for DDR4. So, the number of activation within one t_{REFI} required to trigger bit flips is significantly higher on DDR3 than on DDR4. We hypothesize that bank-level parallelization leads to more overall accesses, e. g., to all banks, but reduces the number of accesses to a single bank. While the reduced number of activations for the single banks is still sufficient to trigger bit flips on DDR4 systems, it is not on DDR3 systems. For DDR4, the bank-level parallelization increases the overall number of bit flips, because multiple banks are accessed at the same time effectively. In contrast, when multiple treads are used, the overall number of accesses is higher, so the number of activations is sufficient for DDR4, but also still sufficient for DDR3. Therefore, multiple banks on DDR3 can be hammered in parallel when using multi-threading, while bank-level parallelism on the memory controller itself is not sufficient to increase the number of bit flips when a single thread is used.

8. Conclusion and Future Work

In this paper, we showed that the number of bit flips can be increased by using two primitives: A combination of the x86 instructions `cmpsb` and `repe` leads to an increase in the number of bit flips found in a given time. Also, the parallel execution of Rowhammer attacks as previously described [23, 13, 7] yielded a significant increase. We evaluated Flipper,

our implementation of both primitives, and showed that it brings amplification factors of 830 compared to RowhammerJS [6] on DDR3 DRAM. We showed that Flipper leads to bit flips that did not occur without using it. We ran our experiments on systems that have the double refresh rate mitigation activated. We showed that Flipper can help to bypass this mitigation technique.

As described before, DDR4 and DDR5 are gradually replacing DDR3 on the market. Therefore, the experiments we described in Section 4 should be repeated on DDR4 and DDR5 DIMMs in future work. Additionally, the amount of DIMMs should be increased to understand this vulnerability better and analyze other instructions to see whether they have similar effects.

Acknowledgements

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) under grant number 503876675, and partly funded by the European Union under grant number ROF-SG20-3066-3-2-2.

References

[1] BIOS Update Utility README (T540p). 2015. URL: `https://download.lenovo.com/pccbbs/mobiles/gmuj16us.txt` (p. 78).

[2] BIOS Update Utility README (X230T). 2015. URL: `https://download.lenovo.com/pccbbs/mobiles/gcuj22us.txt` (p. 78).

[3] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In: S&P. 2016 (p. 73).

[4] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In: S&P. 2020 (pp. 73, 76, 90).

[5] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In: S&P. 2018 (pp. 73, 88).

[6] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016 (pp. 73, 76, 77, 80, 89, 91).

[7] Wei He, Zhi Zhang, Yueqiang Cheng, Wenhao Wang, Wei Song, Yansong Gao, Qifei Zhang, Kang Li, Dongxi Liu, and Surya Nepal. WhistleBlower: A System-level Empirical Study on RowHammer. In: IEEE Transactions on Computers (2023) (pp. 73, 74, 77, 89, 90).

[8] Martin Heckel and Florian Adamsky. Reverse-Engineering Bank Addressing Functions on AMD CPUs. In: Workshop on DRAM Security (DRAMSec). 2023 (p. 76).

[9] Christian Helm, Soramichi Akiyama, and Kenjiro Taura. Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE. 2020 (p. 76).

[10] iAPX 286 Programmer's Reference Manual. 1983. URL: http://bitsavers.org/components/intel/80286/210498-001_iAPX_286_Programmers_Reference_1983.pdf (p. 77).

[11] IC Insights. Distribution of DRAM Market Revenue Worldwide from 2010 to 2021. Statista. Jan. 1, 2021. URL: https://www.statista.com/statistics/553383/worldwide-dram-market-share-by-architecture/ (p. 73).

[12] Intel® 64 and IA-32 ArchitecturesSoftware Developer's Manual - Volume 1: Basic Architecture. Apr. 2021. URL: https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html (p. 76).

[13] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In: SysTEX. 2017. DOI: 10.1145/3152701.3152709 (pp. 73, 74, 77, 89, 90).

[14] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: Rowhammering in the Frequency Domain. In: S&P. Nov. 2021 (pp. 73, 90).

[15] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms. In: USENIX Security. 2024 (pp. 73, 76).

[16] JEDEC Solid State Technology Association. DDR3 SDRAM STANDARD. 2012. URL: https://www.jedec.org/standards-documents/docs/jesd-79-3d (pp. 75, 79).

[17] JEDEC Solid State Technology Association. DDR4 SDRAM STANDARD. 2021. URL: https://www.jedec.org/standards-documents/docs/jesd79-4a (p. 75).

[18] JEDEC Solid State Technology Association. DDR5 SDRAM STANDARD. 2024. URL: https://www.jedec.org/standards-documents/docs/jesd79-5c01 (p. 75).

[19] Ingab Kang, Walter Wang, Jason Kim, Stephan van Schaik, Youssef Tobah, Daniel Genkin, Andrew Kwong, and Yuval Yarom. SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism. In: USENIX Security. 2024 (pp. 73, 77, 78, 89, 90).

[20] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: ISCA. 2014. DOI: 10.1109/ISCA.2014.6853210 (pp. 73, 75, 89, 90).

[21] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In: ISCA. 2012. DOI: 10.1109/ISCA.2012.6237032 (p. 77).

[22] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In: USENIX Security. 2022 (p. 73).

[23] Mark Lanteigne. How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware. 2016. URL: http://www.thirdio.com/rowhammer.pdf (pp. 73, 74, 77, 89, 90).

[24] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. Nethammer: Inducing Rowhammer Faults through Network Requests. In: SILM Workshop. 2020 (p. 73).

[25] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlikçi, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In: ISCA. 2023 (p. 73).

[26] Measuring the DRAM refresh rate by timing memory accesses. 2015. URL: https://github.com/google/rowhammer-test/blob/master/refresh_timing/README.md (p. 79).

[27] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security. 2016 (pp. 75, 76, 89).

[28] Rui Qiao and Mark Seaborn. A New Approach for Rowhammer Attacks. In: HOST. 2016 (pp. 73, 77, 89).

[29] Moinuddin Qureshi. Rethinking ECC in the Era of Row-Hammer. In: DRAMSec. 2021. URL: https://dramsec.ethz.ch/papers/rethinking-ecc.pdf (p. 88).

[30] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In: USENIX Security. 2016 (pp. 73, 88, 89).

[31] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript. In: USENIX Security. 2021 (pp. 73, 89).

[32] Subhash Saini, Robert Hood, Johnny Chang, and John Baron. Performance Evaluation of an Intel Haswell-and Ivy Bridge-Based Supercomputer Using Scientific and Engineering Applications. In: HPCC-SmartCity-DSS. 2016, pp. 1196–1203. DOI: 10.1109/HPCC-SmartCity-DSS.2016.0167 (p. 83).

[33] Jerome H. Saltzer and M. Frans Kaashoek. Principles of Computer System Design: An Introduction. 2009 (p. 73).

[34] Mark Seaborn. Exploiting the DRAM rowhammer bug to gain kernel privileges. Mar. 2015. URL: http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html (pp. 73, 88, 89).

[35] Ken Shirriff. The Microcode and Hardware in the 8086 Processor that Perform String Operations. 2023. URL: https://www.righto.com/2023/04/8086-microcode-string-operations.html (p. 77).

[36] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating software mitigations against rowhammer: a surgical precision hammer. In: RAID. 2018 (p. 80).

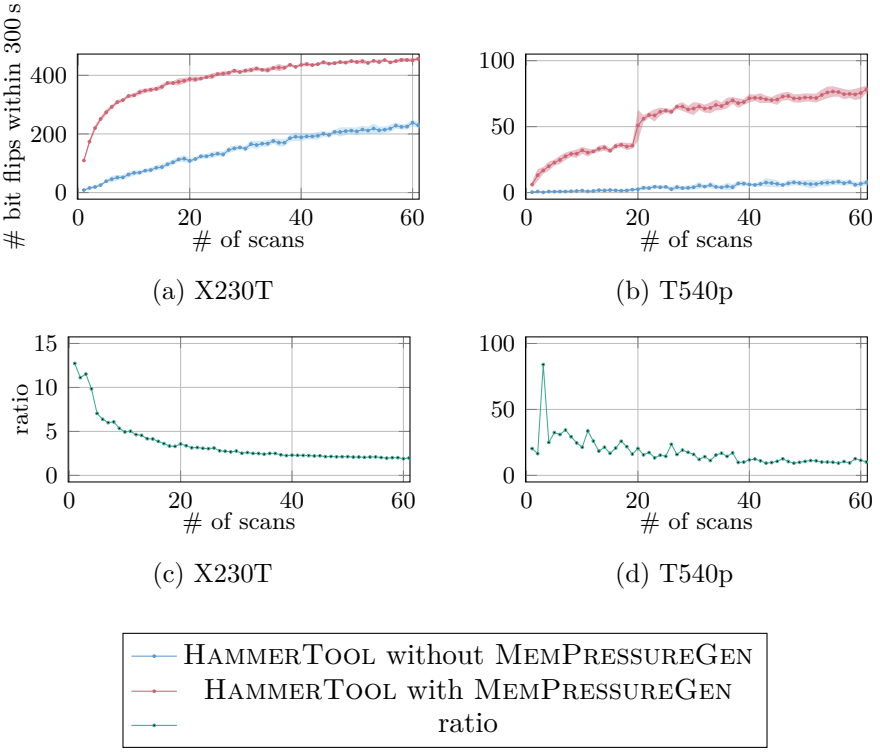
[37] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer Attacks over the Network and Defenses. In: USENIX Security. July 2018 (p. 73).

[38] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clementine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In: ACM CCS. 2016 (p. 73).

[39] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. Dramdig: A Knowledge-assisted Tool to UncoverDRAM Address Mapping. In: Design Automation Conference (DAC). 2020 (p. 76).

[40] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: USENIX Security. 2016 (p. 73).

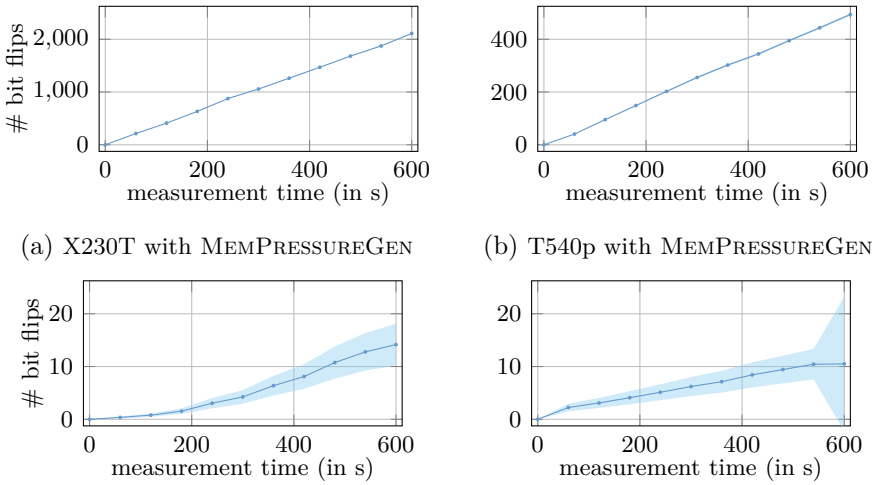
[41] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. Triggering Rowhammer Hardware Faults on ARM: A Revisit. In: Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security. 2018 (p. 73).



(a) X230T (b) T540p

(c) X230T (d) T540p

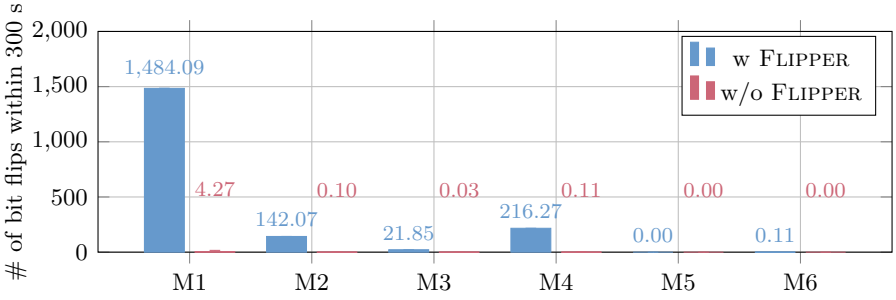
Figure 6.4.: Subfigure (a)–(b) show the number of unique bit flips found by HammerTool with and without MemPressureGen running at the same time when scanning the same THP x times. We show the average over 10 measurements and confidence intervals of 99 %. Subfigure (c)–(d) shows the ratio of bit flips found with and without MemPressureGen in relation to the time of the measurement. Mind the different scales of the y axes.



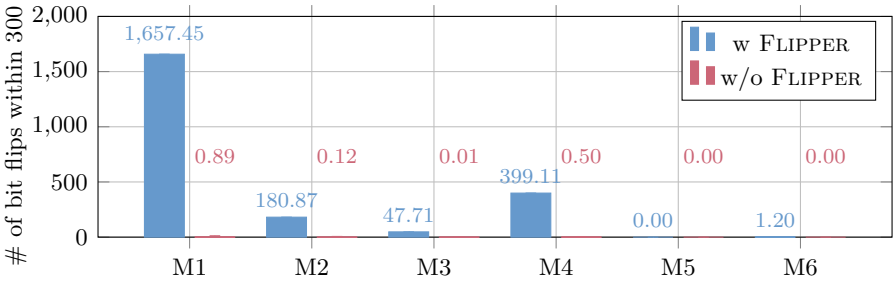
(a) X230T with MemPressureGen (b) T540p with MemPressureGen

(c) X230T without MemPressureGen (d) T540p without MemPressureGen

Figure 6.5.: Ratio of bit flips found with and without MemPressureGen in relation to the time of the measurement. Each measurement runs for 600 s with and without MemPressureGen on the same memory areas. Afterwards, the numbers were summed till the time depicted on the x axis. This means, the value at 60 s contains all bit flips found in the first 60 s, the value at 120 s contains all flips found within the first 120 s, etc. We did 100 measurements with confidence intervals of 99 %. Mind the different scales of the y axis.



(a) X230T



(b) T540p

Figure 6.6.: Comparison of the number of bit flips with and without FLIPPER for six different DDR3 DRAM modules. The error bars show 99 % confidence intervals and are so small that they are nearly not visible.

7

Verifying DRAM Addressing in Software

Publication Data

Martin Heckel, Florian Adamsky, Jonas Juffinger, Fabian Rauscher, and Daniel Gruss. Verifying DRAM Addressing in Software. In: ESORICS. 2025

Contributions

Main author.

Verifying DRAM Addressing in Software

Martin Heckel[1,2], Florian Adamsky[1], Jonas Juffinger[2], Fabian Rauscher[2], Daniel Gruss[2]
[1]firstname.lastname@hof-university.de
[2]firstname.lastname@tugraz.at
[1]*Institute of Information Systems (iisys) Hof University of Applied Sciences* Hof, Germany
[2]*Institute of Information Security (ISEC) Graz University of Technology* Graz, Austria

Abstract

In this paper, we introduce a novel approach to reliably verifying DRAM addressing functions and function components from software. We perform the first systematic analysis of 5 DRAM function reverse-engineering tools on 2 different DDR3, 4 DDR4, and 4 DDR5 system configurations, revealing a significant variance in the success rate of these tools, from 0 % to 92.9 %. We discover the previously unknown rank selection side channel and reverse engineer its function on two DDR4 and two DDR5 systems. These results enable novel DDR5 row-conflict side-channel attacks, which we demonstrate in two scenarios: First, we evaluate the DDR5 row-conflict side channel in a covert channel with 1.39 Mbit/s. Second, we evaluate the channel in a website fingerprinting attack with an F_1 score of 84 % on DDR4 and 74 % on DDR5.

1. Introduction

Software-level security often assumes that hardware is functioning correctly and without side effects. However, physical effects can undermine system security with side-channel and fault attacks. One attack target is DRAM, the main memory in modern computers. There are side channels [27, 41], fault attacks [14, 5], and slowdown attacks [24] on DRAM, undermining a system's confidentiality, integrity, and availability. DRAM addressing functions can be used in performance optimization, such as application-aware memory channel partitioning [25] or variable page sizes [38] for more efficient row-buffer usage. However, DRAM side-channel and fault attacks

also often use these functions: Pessl et al. [27] presented the first DRAM side-channel attacks exploiting reverse-engineered addressing functions. Seaborn [34, 33] used DRAM addressing functions for the first practical *Rowhammer* exploit. Rambleed [17] exploits bit flips in attacker memory, depending on inaccessible victim data bits and addressing functions.

DRAM addressing functions are defined by the CPU's memory controller, i. e., specific to the CPU model and the memory configuration. Consequently, it is necessary to reverse-engineer the functions **anew** for every system. Prior works utilize DRAM access timings [27, 40, 4, 8]. Helm et al. [9] reverse-engineered the DRAM functions using performance counters. Most works did not verify the functions systematically but only tested whether an attack succeeded. However, Rowhammer bit flips can also occur with incorrect or even without addressing functions, e. g., One-Location Rowhammer [5]. Only Pessl et al. [27] and Jattke et al. [12] verified addressing functions using high-bandwidth oscilloscopes.

In this paper, we present a novel methodology to verify the correctness of DRAM addressing functions purely in software. Our approach is based on the fact that the theoretical success rate of DRAM side channel tests changes with each function component. Consequently, we can verify the correctness of even single output bit of the DRAM addressing functions purely from software. Thus, we can identify incorrect function components and combine the outputs of multiple reverse-engineering tools to a complete and correct a set of functions. We evaluate our approach on 10 systems and show that the maximum deviation from theoretical values for correct functions is 0.76 % on DDR3, 0.52 % on DDR4, and 0.49 % on DDR5, indicating the high precision of our approach.

Based on our novel verification methodology, we present the first systematic analysis of DRAM addressing function reverse-engineering approaches. We observe success rates of 92.9 % on DDR3, 85.6 % on DDR4, and 87.3 % on DDR5 for the full DRAM addressing functions with the best respective reverse-engineering tool [27, 40, 4, 8, 12]. We show that 3 tools yield good results (i. e., ≈ 90 % correct) on our DDR3 systems (Intel), 4 tools yield good results (i. e., ≈ 80 % correct) on our DDR4 systems (Intel), and only 1 tool yields moderate results (i. e., ≈ 60 % correct) on DDR4 (AMD) and good results (i. e., ≈ 85 % correct) on DDR5 (Intel). No tool yields good results on DDR5 (AMD).

Using our approach, we found an additional layer in DRAM addressing: There is a measurable timing difference between addresses of the same

rank and those of different ranks. We strongly suspect this is due to the rank select commands sent between accesses to different ranks. We are the first to reverse-engineer *rank addressing functions* and show that using such functions increases the success rate by 18.36 % to 36.11 % on different systems.

We built the first row-conflict covert channel on DDR5 with a true capacity of up to 2.23 Mbit/s on DDR3, 0.66 Mbit/s on DDR4, and 1.39 Mbit/s on DDR5. Additionally, we evaluate our approach in a row-conflict side-channel website-fingerprinting attack, identifying a single website out of 100 with an F_1 score of 84 % on DDR4 and 74 % on DDR5.

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

1. We present a new approach for DRAM address function verification in software[1] based on computing deviations from the theoretical behavior.
2. We systematically evaluate 5 reverse-engineering tools on 2 DDR3, 4 DDR4, and 4 DDR5 systems and show that none produce good results on all systems.
3. We discover a novel rank selection timing channel and reverse-engineer the corresponding rank function on two DDR4 and two DDR5 systems.
4. We present the first row-conflict covert channel on DDR5[2]. with 1.39 Mbit/s.
5. We demonstrate a novel row-conflict-based side-channel attack on DDR5, allowing to distinguish 100 websites with an F_1 score of 74 %.

Outline. Section 2 provides background. Section 3 describes our setup. Section 4 presents our new function verification approach. Section 5 presents our DRAM rank addressing insights. Section 6 presents our covert channel on DDR5, and Section 7 our website-fingerprinting attack. Section 8 concludes.

2. Background and Related Work

This section discusses DRAM and DRAM addressing, covert channels, website fingerprinting, and related work.

DRAM. DRAM cells consist of capacitors and transistors organized in rows and columns, forming DRAM banks grouped into ranks on a

[1] `https://github.com/iisys-sns/DramaVerify`
[2] `https://github.com/iisys-sns/DramaNg`

DIMM [27]. DIMMs are connected to the CPU's memory controller via channels. Activating cells (reading the data into a row buffer) is destructive, so content must be written back before activating another row. Since capacitors lose charge over time, DRAM needs periodic recharging, e. g., every 64 ms.

DRAM Addressing Functions. Addressing functions map consecutive memory into different banks, ranks, and channels to minimize bank conflicts. The memory controller translates physical addresses via addressing functions into DRAM components, i. e., channels, DIMMs, ranks, banks, rows, and columns.

Linear Addressing functions are used on systems where the number of all DRAM components is a power of two. They are represented as a hexadecimal bitmask, indicating which bits need to be XORed. For example, the addressing function $0x88000$ corresponds to 1000100000000000000_2, indicating that the 19th and the 15th bit of the physical address needs to be XORed. Each function distinguishes two states, meaning there are $log_2(n_{\text{banks}})$ addressing functions on a system with n_{banks} DRAM banks. Reverse-engineering non-linear addressing functions (which use operations other than XOR) remains an open problem.

Covert Channels. Various microarchitectural elements have been used in covert communication channels [18], e. g., via CPU load [26], CPU caches [42, 45, 46, 22, 21, 23, 28, 7, 31], and the memory bus [44, 43]. Covert channels have become a best practice to evaluate the capacity of side channels. Semal et al. [35] present a DRAMA covert channel on DDR3 and DDR4 systems with up to 729 bit/s. Wang et al. [41] show that this channel also affects Intel SGX. Van der Veen et al. [39] amplify the DRAMA channel by making all memory uncacheable. In a simulation, Kushwaha et al. [16] showed certain *secure* cache designs can also amplify the DRAMA channel from a simulated range of 2.73 Mbit/s to 4.61 Mbit/s to 4.53 Mbit/s to 6.82 Mbit/s. The fastest DRAM covert channel to date [27] achieves a capacity of up to 2 Mbit/s on DDR4.

Website-Fingerprinting Side Channels. A common scenario for side-channel evaluation with low spatial resolution is website fingerprinting. Fingerprinting attacks exploited Android data usage [37], browser memory usage [10], the power side channel [29], cache occupancy [36], interrupt timing [3, 47, 30], SSD contention [13], and timing side channels in the operating system [19], often achieving high F1 scores in open- and closed-world scenarios on the top 100 websites.

Related Work. Several Rowhammer exploits do not use DRAM bank addressing functions [14, 32, 5]. However, newer and sophisticated Rowhammer exploits [32, 6, 4, 15, 11] use DRAM addressing functions to increase the number of bit flips with more targeted hammering.

Pessl et al. [27] reverse-engineered DRAM bank addressing functions from software and used physical memory-bus probing for verification. Barenghi et al. [2] and Marazzi et al. [20] also use the timing-based approach to reverse-engineer DRAM addressing functions. Prior work often determines the correctness through success rates of resulting attacks, e. g., the overall success rate of the full set of addressing functions. In contrast, we demonstrate that the *theoretical influence of correct functions can be leveraged as ground truth to verify the correctness of single reverse-engineered functions*. While we also discover the rank selection side channel that was previously unknown, we focus our verification on the known row-conflict side channel with addressing functions from prior work (which did not include rank selection functions).

DRAMDig [40] is a knowledge-assisted tool to determine DRAM address mappings and then run a double-sided rowhammer test for verification. Zhang et al. [48] extended DRAMDig to a Rowhammer testing tool using reverse-engineered addressing functions. Frigo et al. [4] trigger bit flips in TRR-enabled DIMMs using a many-sided pattern with many aggressor rows, relying on reverse-engineering DRAM addressing functions. Helm et al. [9] use performance-counter-based reverse-engineering, focusing on Intel Haswell, Broadwell, and Skylake. Since we also cover AMD, we cannot use their approach in our measurements.

Fewer works focused on AMD. AMD published functions on older CPUs [1] but not for newer ones, where Heckel et al. [8] and Jattke et al. [12] recently reverse-engineered DRAM addressing functions. They also reverse-engineered DRAM addressing functions on DDR5 utilizing the row-conflict side-channel we utilize for the attacks presented in this paper. Jattke et al. [12] verify the function correctness with an oscilloscope, like Pessl et al. [27].

3. Experimental Setup

Our experimental setup consists of 2 systems with DDR3, 4 with DDR4, and 4 with DDR5 DRAM. Each system has a unique ID that follows the

Table 7.1.: Systems used for experimental evaluation. The number of banks (n_{bnk}) is the number of all banks in the system, e. g., there are $\frac{n_{\text{banks}}}{n_{\text{rnk}}}$ banks per rank.

	System	CPU	Memory	n_{bnk}	n_{rnk}
DDR3	*S301*	*i5-3320M*	4 GiB	8	1
	S302	*i7-4800MQ*	4 GiB	16	2
DDR4	*S401*	*i9-10900K*	8 GiB	16	1
	S402	*i9-10900K*	8 GiB	32	2
	S403	*5950X*	8 GiB	16	1
	S404	*5950X*	8 GiB	32	2
DDR5	*S501*	*i7-13700*	16 GiB	64	1
	S502	*i7-13700*	16 GiB	64	1
	S503	*7700X*	16 GiB	64	1
	S504	*7700X*	16 GiB	64	1

format: `S⟨DDR Version⟩⟨Counter⟩`. All systems run a current version of Arch Linux (6.8.7-arch1-1). Since rank addressing functions are a significant part of the experimental evaluation, we use DIMMs with one or two ranks. If we have multiple similar systems, we use a DIMM with one rank in one system and a DIMM with two ranks in the other (see Table 7.1 for details). We use `/proc/self/pagemap` to get physical to virtual addresses mappings.

4. Verification of DRAM Addressing Functions

This section shows how we verify DRAM bank addressing functions based on Drama [27]. Rather than exploiting the varying timings between row hits and conflicts to create a covert channel, we utilize this side channel to confirm the accuracy of the bank addressing functions. Although we use the same side channel from Drama, we utilize it *differently*, as discussed in Section 4.1 We verify our approach using DRAM bank addressing functions reverse-engineered by five existing tools [27, 40, 4, 8, 12]. Our criterion for selecting the tools was that they were published in the last ten years and do not require features available only on one architecture. We omitted Helm et al. [9] since their approach uses performance counters exclusive to Intel CPUs from Haswell to Skylake.

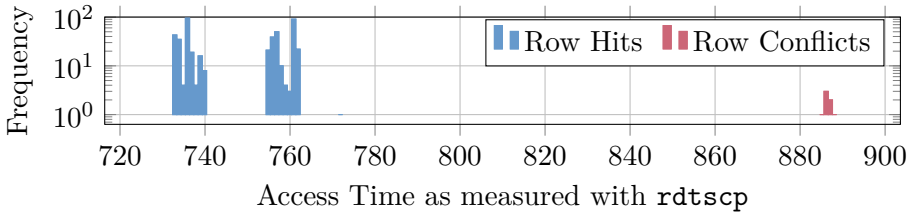


Figure 7.1.: Histogram of access times when accessing randomly selected pairs of addresses in a flush and reload loop. The gap around 825 TSC cycles hints at the threshold between row hits and misses on *S404*.

4.1. Verification Steps

This section describes the steps to verify reverse-engineered DRAM bank addressing functions. First, we measure the threshold between a row hit and a conflict before allocating 1 GiB of memory. We resolve the physical addresses with elevated privileges, group the allocated memory based on the given addressing functions, and run the DRAMA side channel. We compute a success rate for the addressing function from the number of row hits and conflicts.

Measure the Threshold between Row Hit and Row Conflict. We allocate one 2 MiB transparent hugepage (THP) (512 single 4 KiB pages) and measure the access time using the `rdtsc` instruction of the first (offset 0) and second (offset 4 KiB) page within the THP. We repeat this for all other pages within the THP, e. g., comparing the first and the third, and so on. Between measurements, we clear the CPU cache with `clflush`.

A histogram from the measured access times is shown in Figure 7.1. While we expect the histogram to have two peaks, one for fast row hits and one for slow row conflicts, there are three, two with row hit timings. This can be explained by the fact that *S404* has two ranks. We also performed this experiment on *S401*, which has only one rank and only one row-hit peak. Accesses to addresses on the same rank (*rank hits*) do not require the memory controller to issue a rank select command between them. Access to addresses on different ranks (*rank conflicts*) required the memory controller to issue rank select commands. For this reason, those accesses are slightly slower. There are two ranks on *S404*, and the addresses are equally distributed over both ranks, so there are two peaks of similar size. The histogram is analyzed to identify the threshold t_T

between the *row hit* peak and the *row conflict* peak (around 825 for the histogram shown in the plot).

Grouping Addresses. We allocate 1 GiB and get the physical addresses from `/proc/self/pagemap`. With 2^{30} single addresses in 1 GiB, only a fraction (1 % by default) is used. We apply all reverse-engineered DRAM addressing functions under test to the physical addresses. The bits are XORed, and the resulting bit is considered a bit of the bank number. When n addressing functions are applied to a physical address, there is an n-bit bank number. Finally, the virtual address is added to the group to match the bank number computed before.

Verification. After address grouping, the row-conflict side channel [27] is used for group verification. Only correct addressing functions lead to correct groups, so it is sufficient to verify that the groups are correct to derive that the addressing functions are correct. There are 2^n groups for n functions. Several address pairs (a_1, a_2) (we use 5 000 by default, since this number yielded good results in a brief comparison) are randomly selected for each group. Additionally, an address (b_1) from another randomly selected group is selected.

Then, we measure the timing of alternatingly accessing (a_1, a_2), expecting a row conflict ($t_c > t_T$), and (a_1, b_1), expecting a row hit ($t_h < t_T$). Each of these timing measurements is performed a few hundred times, and the median of the measurements is used as the resulting value. If both timings are correct, $t_c > t_T$ and $t_h < t_T$, the address pair is considered to be grouped correctly.

Evaluation. We compute the share of correctly grouped pairs from all address pairs, resulting in a correctness measure for reverse-engineered DRAM bank addressing functions. In Section 4.2, we experimentally evaluate our approach.

4.2. Experimental Evaluation

We evaluate our approach with the following reverse engineering tools: Dare [12], Drama [27], DRAMDig [40], TRRespass RE Tool [4], and AMDRE [8]. After **identifying the addressing functions with these tools**, we ran our verification method on these functions 10 times. We report the average percentage of cases where the assumed banks are correct. We show a graphical representation of all bits identified to belong to at

Table 7.2.: Experimentation results of multiple PoCs [27, 40, 4, 8, 12] on our DDR3 systems. 10 measurements were performed. AFn Mask shows a graphical representation of all bits belonging to at least one DRAM bank addressing function. The average percentage $\%_{\text{avg}}$ is lower than $\%_{\text{max}}$ when some runs failed and some succeeded.

	PoC	AFn Mask	$\%_{\text{avg}}$	σ	$\%_{\text{min}}$	$\%_{\text{max}}$		PoC	AFn Mask	$\%_{\text{avg}}$	σ	$\%_{\text{min}}$	$\%_{\text{max}}$
S301	AMDRE		83.5 %	27.4	1.4 %	92.9 %	*S302*	AMDRE		90.1 %	0.1	90.0 %	90.3 %
	DRAMDig		92.7 %	0.1	92.5 %	92.8 %		DRAMDig		89.8 %	0.8	87.3 %	90.2 %
	Drama		43.9 %	34.1	8.3 %	92.7 %		Drama		42.3 %	35.4	0.0 %	90.1 %
	Dare		0.0 %	0.0	0.0 %	0.0 %		Dare		0.0 %	0.0	0.0 %	0.0 %
	TRRespass		0.0 %	0.0	0.0 %	0.0 %		TRRespass		0.0 %	0.0	0.0 %	0.0 %

Table 7.3.: Experimentation results ([27, 40, 4, 8, 12]) on our DDR4 systems (left) and DDR5 systems (right), analogue to Table 7.3.

	PoC	AFn Mask	$\%_{\text{avg}}$	σ	$\%_{\text{min}}$	$\%_{\text{max}}$		PoC	AFn Mask	$\%_{\text{avg}}$	σ	$\%_{\text{min}}$	$\%_{\text{max}}$
S401	AMDRE		85.4 %	0.1	85.2 %	85.5 %	*S501*	AMDRE		42.9 %	42.9	0.0 %	86.0 %
	DRAMDig		84.8 %	0.2	84.4 %	85.1 %		DRAMDig		0.0 %	0.0	0.0 %	0.0 %
	Drama		15.3 %	15.6	0.0 %	44.9 %		Drama		5.1 %	1.8	1.6 %	6.3 %
	Dare		76.8 %	25.6	0.0 %	85.5 %		Dare		6.3 %	0.0	6.3 %	6.4 %
	TRRespass		84.1 %	2.1	79.6 %	85.6 %		TRRespass		5.4 %	1.2	2.9 %	6.1 %
S402	AMDRE		77.2 %	0.2	76.9 %	77.4 %	*S502*	AMDRE		87.0 %	0.2	86.7 %	87.3 %
	DRAMDig		42.3 %	0.1	42.0 %	42.5 %		DRAMDig		0.0 %	0.0	0.0 %	0.0 %
	Drama		6.7 %	6.5	0.0 %	21.3 %		Drama		4.0 %	2.2	0.0 %	5.5 %
	Dare		29.6 %	19.4	0.0 %	42.4 %		Dare		4.6 %	1.7	0.0 %	5.4 %
	TRRespass		42.2 %	0.1	42.0 %	42.4 %		TRRespass		5.4 %	0.1	5.2 %	5.5 %
S403	AMDRE		23.2 %	13.0	0.0 %	38.8 %	*S503*	AMDRE		0.0 %	0.0	0.0 %	0.0 %
	DRAMDig		0.0 %	0.0	0.0 %	0.0 %		DRAMDig		0.0 %	0.0	0.0 %	0.0 %
	Drama		13.9 %	9.7	0.0 %	23.1 %		Drama		22.7 %	0.6	21.8 %	23.7 %
	Dare		14.4 %	6.0	0.0 %	21.6 %		Dare		1.2 %	1.2	0.0 %	2.6 %
	TRRespass		0.0 %	0.0	0.0 %	0.0 %		TRRespass		0.0 %	0.0	0.0 %	0.0 %
S404	AMDRE		62.3 %	0.5	61.8 %	63.4 %	*S504*	AMDRE		0.0 %	0.0	0.0 %	0.0 %
	DRAMDig		0.0 %	0.0	0.0 %	0.0 %		DRAMDig		0.0 %	0.0	0.0 %	0.0 %
	Drama		16.0 %	5.8	7.3 %	21.8 %		Drama		20.6 %	6.9	0.0 %	23.6 %
	Dare		18.5 %	1.6	16.1 %	20.5 %		Dare		18.9 %	9.5	0.0 %	23.7 %
	TRRespass		0.0 %	0.0	0.0 %	0.0 %		TRRespass		0.0 %	0.0	0.0 %	0.0 %

least one DRAM bank addressing function. Therefore, it is possible to see how stable the functions were over multiple measurements. Additionally, to function correctness, we evaluate the stability of the tools as follows: **Stable** tools yield the same result upon every execution. **Mostly Stable** tools yield the same result in $\geq 70\,\%$ of runs and only 1 or 2 bits difference in the other cases, or no result in at most 3 runs. **Unstable** tools yield the same result in $\geq 70\,\%$ of runs, but other runs vary. **Completely Unstable** tools have varying results for all runs. **Failed** tools crashed or returned nothing.

DDR3. Table 7.2 summarizes our results on two DDR3 systems. DRAMDig consistently identified *stable* addressing functions with success rates around 92 %. AMDRE produced *mostly stable* results but with high variance (1.4 % to 92.9 %). In contrast, Drama was *completely unstable*, ranging from 0 % to 92.7 %. Dare and TRRespass *failed* on both systems.

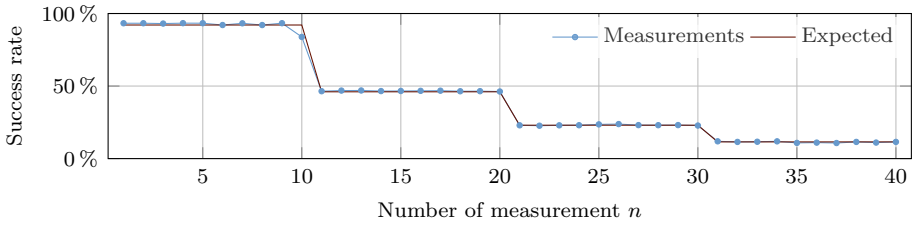


Figure 7.2.: Success rate reported depending on the number of addressing functions submitted. The measurement was done on *S401* with 4 addressing functions. For each number of addressing functions, 10 measurements were performed.

DDR4. As shown in Table 7.3, on *S401* and *S402*, AMDRE, DRAMDig, and TRRespass returned *stable* functions, Dare reported *mostly stable* functions and Drama reported *completely unstable* functions. Drama has a success rate of 0 % to 44.9 %. The maximum success rate for AMDRE, DRAMDig and Dare is around 77 % to 85 % and around 42 % to 85 % for TRRespass. The minimum success rate is approximately 85 % for AMDRE and DRAMDig, it is 79.6 % for TRRespass and 0 % to 42 % for Dare.

On *S403* and *S404*, DRAMDig and TRRespass *failed*. The masks of Dare are *stable* and *mostly stable*. AMDRE returns *completely unstable* and *stable* results. The success rates are generally lower than for the other two systems.

DDR5. Table 7.3 shows the results for our four DDR5 systems. No tool has stable results across all machines. DRAMDig *failed* on all machines and AMDRE and TRRespass on *S503* and *S504*. AMDRE is only *stable* on one machine, TRRespass is *unstable* on the two machines where it works. Drama was *unstable* or *completely unstable* across all machines. The success rates are generally lower than for DDR4, with AMDRE reaching the highest success rates of approximately 87 % on two systems. Dare has *stable* and *mostly stable* results with maximum success rates of 2.6 % to 23.7 %.

4.3. Verification of Single Addressing Functions

We verify the entire set of addressing functions in Section 4.2. However, with that, we cannot make any statement about single functions. This

section extends the approach from Section 4.2 to verify single addressing functions within a set.

If a system has n_{fn} correct DRAM bank addressing functions, the number of banks is $n_{\text{banks}} = 2^{n_{\text{fn}}}$. If we remove one of these functions, the number of banks addressable halves: $2^{n_{\text{fn}}-1} = \frac{2^{n_{\text{fn}}}}{2} = \frac{n_{\text{banks}}}{2}$. No addresses can be added to half of the banks because they can not be addressed with the reduced number of functions. This results in 50 % of the DRAM banks no longer being accessible via addressing functions and reduces the success rate by 50 %, as shown in Figure 7.2.

In the range $0 < n \leq 10$ we use all DRAM addressing functions `0x2040`, `0x24000`, `0x48000`, and `0x90000`. The average success rate is 92.08 %. For $10 < n \leq 20$, we remove `0x90000` and the average success rate halves to 46.61 %. In the $20 < n \leq 30$ range, we remove `0x48000` resulting in an average success rate of 23.09 %. Finally, for $30 < n \leq 40$, the function `0x2040` is used, and the average success rate is 11.35 %, with an expected success rate of 11.51 %.

We conclude that all the DRAM addressing functions used above are correct. However, this approach has the disadvantage that the differences between using or not using a function get lower the more functions were removed before. For example, the expected difference for removing the first function is 46.04 %, which decreases to 11.51 % for removing the third function. However, as DRAM bank addressing functions are not ordered, we remove every function from the initial set of all functions individually, comparing the new success rate to the initial one. If a correct function is removed, the success rate is expected to halve.

We evaluate our verification on DDR3, DDR4, and DDR5, each with the set of addressing functions reverse-engineered in Section 4.2 that yields the highest percentage on the respective system. Then, we manually modify some of the DRAM addressing functions. Afterward, we repeat the experiment with the modified function to verify that our approach can detect the modified, now wrong, addressing functions. The results of this evaluation are shown in Table 7.4.

For *S301*, *S401*, and *S501*, the submitted addressing functions were identified to be correct, with a maximum difference of 0.76 % between the expected and the measured values when no modifications were performed. Without any modified functions, all systems' base success rate ($2 \times \%_{\text{exp}}$) is approximately 90 %. With two modified functions, the success rate drops to 21.34 % on *S301*. Both incorrect functions were identified, with

Table 7.4.: Evaluation of single addressing functions grouped by system. The value of $\%_{\text{exp}}$ is always 50 % of the measured initial success rate for the entire function set. Manual manipulation to obtain wrong functions (see the ✓ under *Mod*).

	Function	Mod.	$\%_{\text{meas}}$	$\%_{\text{exp}}$	$\%_{\text{diff}}$	Cor.
S302	0x22000	✗	47.6 %	46.1 %	1.5 %	✓
	0x44000	✗	47.6 %	46.1 %	1.5 %	✓
	0x88000	✗	47.8 %	46.1 %	1.7 %	✓
	0x110000	✗	47.7 %	46.1 %	1.6 %	✓
S401	0x2040	✗	46.1 %	46.6 %	0.5 %	✓
	0x24000	✗	46.4 %	46.6 %	0.2 %	✓
	0x48000	✗	46.4 %	46.6 %	0.2 %	✓
	0x90000	✗	46.3 %	46.6 %	0.3 %	✓
S501	0x6300	✗	43.6 %	43.3 %	0.3 %	✓
	0x10000	✗	43.6 %	43.3 %	0.3 %	✓
	0x20000	✗	43.8 %	43.3 %	0.5 %	✓
	0x42300	✗	43.4 %	43.3 %	0.1 %	✓
	0x81100	✗	43.5 %	43.3 %	0.2 %	✓
	0x108000	✗	43.3 %	43.3 %	0.0 %	✓

	Function	Mod.	$\%_{\text{meas}}$	$\%_{\text{exp}}$	$\%_{\text{diff}}$	Cor.
S302	0x23000	✓	23.1 %	10.7 %	12.4 %	✗
	0x44000	✗	10.9 %	10.7 %	0.2 %	✓
	0x89000	✓	22.6 %	10.7 %	11.9 %	✗
	0x110000	✗	10.9 %	10.7 %	0.2 %	✓
S401	0x2040	✗	10.4 %	10.4 %	0.0 %	✓
	0x25000	✓	20.8 %	10.4 %	10.4 %	✗
	0x48000	✗	10.5 %	10.4 %	0.1 %	✓
	0x110000	✓	22.6 %	10.4 %	12.2 %	✗
S501	0x6100	✓	9.43 %	3.11 %	6.32 %	✗
	0x10000	✗	3.07 %	3.11 %	0.04 %	✓
	0x24000	✓	9.19 %	3.11 %	6.08 %	✗
	0x42300	✗	3.08 %	3.11 %	0.03 %	✓
	0x82100	✓	9.36 %	3.11 %	6.25 %	✗
	0x108000	✗	3.08 %	3.11 %	0.03 %	✓

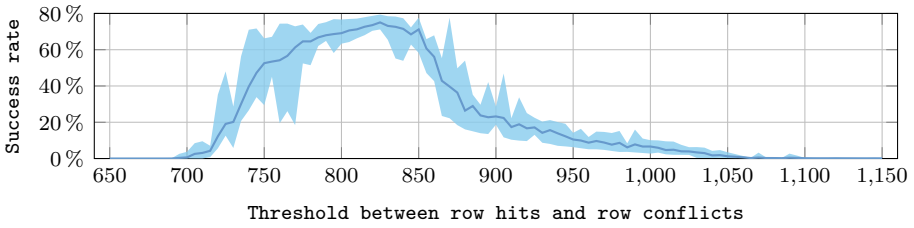
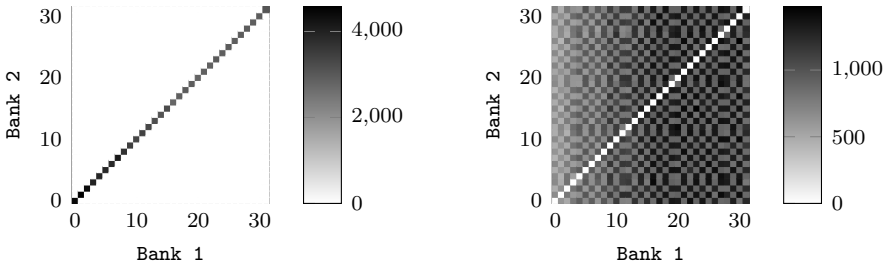


Figure 7.3.: Success rate of our verification approach depending on the threshold. The graph shows average, minimum, and maximum values over 10 measurements.

differences of 11.43 % and 11.80 %. The correct function was identified with a difference of 0.71 %. On *S401*, two modified addressing functions resulted in an overall success rate of 20.88 %. The incorrect functions were identified, with differences of 10.39 % and 12.18 %. Both correct functions were identified, with differences of 0.01 % and 0.08 %. The three modified addressing functions on *S501* resulted in a drop in the overall success rate of 6.22 %. The incorrect functions have differences of 6.32 %, 6.08 %, and 6.25 %. In contrast, the three correct functions have differences of 0.04 %, 0.03 %, and 0.03 %. Our approach identifies correct and incorrect DRAM bank addressing functions in all cases within this experiment.



(a) Number of too fast row conflicts. (b) Number of too slow row hits.

Figure 7.4.: Heat maps of both errors: too fast row conflicts and too slow row hits. The x-axis shows the group of the first address, the y-axis the second selected address. Measurements on *S404* (with threshold 495), averaged over 10 measurements.

5. Rank Timing Analysis

To analyze the lower-than-expected success rate (see Section 4.2), we perform multiple measurements and finally show that a second layer of addressing functions is used to determine the rank of a physical address.

5.1. Rank Addressing Functions

We use the same timing notation introduced in Section 4.1 and find four timing cases, **C1**, with $t_c > t_T$ and $t_h < t_T$, **E1**, with $t_c < t_T$ and $t_h < t_T$, **E2**, with $t_c < t_T$ and $t_h > t_T$, and **E3**, with $t_c > t_T$ and $t_h > t_T$. When the addresses are correctly grouped, and both addresses from the same group are not in the same row, t_c cannot be smaller than t_h since a row hit (t_h) is faster than a row conflict (t_c). Depending on the addresses chosen, the case labeled E2 might occur even when correct addressing functions are used. The probability of this happening is discussed below. By default, 1 % of the available addresses is grouped, which is 10 MiB of 1 GiB. If all selected addresses are contiguous, 10 MiB of memory are distributed over n_{banks} banks. In the case of 32 DRAM banks, this results in $\frac{10\,\text{MiB}}{32} = 320\,\text{KiB}$ per DRAM bank. With the assumption that the addresses completely populate contiguous rows with a row size of 8 KiB, the selected addresses populate $\frac{320\,\text{KiB}}{8\,\text{KiB}} = 40$ rows. So, the probability of one randomly selected address being in one chosen row is $\frac{1}{40} = 2.5\,\%$. Following this, the probability of two randomly selected addresses being

in the same DRAM bank is 2.5 %. Therefore, case E2 is doubtful when correct DRAM bank addressing functions are used, since two randomly selected addresses from the same DRAM bank would have to be in the same row, which happens at a probability of 2.5 % in the worst case. Under the assumption that the tested DRAM bank addressing functions are correct, the cases E1, C1, and E3 are statistically relevant.

In contrast to the approach described in Section 4.1, the threshold is manually specified and not measured for this experiment. The threshold impacts the success rate, as shown in Figure 7.3. The statistically relevant cases E1, C1, and E3 depend on the selected threshold as shown below:

E1: Since both timings (t_c and t_h) are lower than the threshold, row conflicts are misclassified as row hits. The threshold is selected too high (for $t_T \geq 875$ in the graph shown in Figure 7.3).

C1: Since row conflicts (t_c) are slower and row hits (t_h) are faster than the threshold, both cases are classified correctly. The threshold is set correctly (for $875 \geq t_T \geq 795$ in the graph shown in Figure 7.3).

E3: Since both timings (t_c and t_h) are higher than the threshold, hits are misclassified as conflicts. The threshold is too low (for $t_T \leq 795$ in Figure 7.3).

Note that the three cases overlap and merge, so the submitted threshold values describe a range and not a specific value. In these measurements, two types of single errors can occur: **(e1)** a row hit is too slow and misclassified as conflict; and **(e2)** a row conflict is too fast and misclassified as hit.

Figure 7.4 shows heat maps for both errors. Row conflicts are expected when comparing addresses from the same DRAM bank, so there are n_{banks} different cases (one for each DRAM bank), as shown in Figure 7.4a. Row hits are expected between addresses from one DRAM bank and addresses from any other DRAM bank, as shown in Figure 7.4b. The heat map shows no values on the diagonal since addresses from the same bank are not expected to be row hits.

As shown in Figure 7.4b, there is a pattern in the error number depending on the addresses of which DRAM banks are compared to each other. When comparing an address from bank 0 to another address, the number of errors is lower for the following banks: (2, 5, 7, 9, 11, 12, 14, 16, 18, 21, 23, 25, 27, 28, 30). Similarly, comparing an address from bank 1 with another address has fewer errors for the following banks: (3, 4, 6, 8, 10, 13, 15, 17, 19, 20, 22, 24, 26, 29, 31). So, there are two groups of DRAM banks for which the error rate is lower when addresses are selected from banks

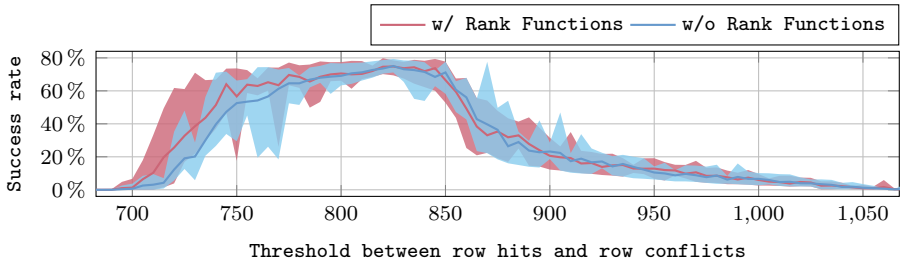


Figure 7.5.: Average, minimum and maximum success rate (10 measurements) of our verification approach with and without using rank addressing functions. The graph without rank addressing functions is the same one shown in Figure 7.3.

within the same group. At the same time, the error rate is higher when selecting two addresses from banks in different groups.

This grouping can be described by a *grouping function* taking bank number (0–31) as input and returning an output of one bit equivalent to the number of the groups. The function can be represented by a bitmask that selects bits of the input and applies a bitwise XOR to them. When bank numbers are represented in binary, the addressing function `0xd` (`0b01101`) can be applied similarly to the DRAM bank addressing functions to get a resulting bit determining the group.

The error rate is lower when two addresses from different banks within the same bank group are selected, so we restrict the measurements only to those cases. We compare the success rates with or without applying the additional DRAM addressing function. The result of this experiment is shown in Figure 7.5.

When additional rank addressing functions are used, the success rate is higher in the $700 \leq t_T \leq 780$ range. For $t_T \leq 700$, the success rate is nearly 0 for both graphs. When $t_T \geq 780$, the success rates for both graphs are similar.

The range $700 \leq t_T \leq 780$ has a similar upper border as $t_T \leq 795$, as discussed in case *E3*, in which both timings (t_c and t_h) are higher than the threshold. So, the number of row hits misclassified as row conflict is lower when rank addressing functions are used.

Some row hits are faster than others, as shown in Figure 7.4b. However, in contrast to the patterns in the heatmaps that occurred in the range

Table 7.5.: Success rate at the threshold (t_T) with a maximum difference with and without DRAM rank addressing functions (RF) yielding the best results in Section 4.2 (10 measurement average). No rank functions found on other systems.

System	RF	t_T	% w RF	% w/o RF	Δ_{RF}
S402	`2`	540	66.16 %	38.77 %	27.40 %
S404	`13`	730	38.54 %	20.18 %	18.36 %
S501	`6,8,10`	255	77.32 %	42.86 %	34.46 %
S502	`6,8,10`	230	55.63 %	19.52 %	36.11 %

$920 \leq t_T \leq 980$ as well, the differences in the success rate graph are only relevant in the range of row hits being misclassified as row conflicts.

The memory controller issues *Rank Select* commands every time the rank changes, so it is assumed that accessing two addresses from the same rank that are on different banks (e. g., row hit) is faster than accessing two addresses from different ranks (e. g., row hit). This effect occurs only on systems with multiple ranks (we tested systems with 1 and 2), as further evaluated in Section 5.2. Therefore, we conclude that the effect is related to the DRAM rank.

5.2. Experimental Evaluation

We perform the steps described in Section 5.1 for experimental evaluation. If a DRAM rank addressing function is derived, it is submitted to the verification tool. The maximum distance between the success rates with and without using the rank addressing function is measured. The results are shown in Table 7.5.

On both DDR3 systems (*S301* and *S302*), it was not possible to see any patterns in the number of row hits that were too slow, similar to the one shown in Figure 7.4b. Therefore, we could not apply a rank addressing function and skipped the evaluation of the rank addressing function.

For the DDR4 systems, patterns occurred only on systems with two ranks (*S402* and *S404*), there were no recognizable patterns for the systems with one rank (*S401* and *S403*). Therefore, we evaluate rank addressing functions only on *S402* and *S404*. On *S402*, we identify the rank addressing function `2`. We observe a maximum difference of 27.40 %. On *S404*, the rank addressing function is `13`. The same data as in Section 5.1 was used

for the evaluation. The maximum distance of 18.36 % was reached at a threshold of 730.

As the DRAM addressing functions on *S503* and *S504* reached very low success rates in our experiments (see Section 4.2), these systems were excluded from the evaluation of rank addressing functions. In contrast to the DDR4 systems discussed before, *S501* and *S502* had correct DRAM addressing functions. Recognizing a pattern in the number of row hits that were too slow was also possible. From this pattern, we derived the rank addressing functions *6,8,10* for both systems. On *S501*, the maximum difference between using and not using the rank addressing functions is 34.46 %. For *S502* it is 36.11 %.

The row-conflict side-channel can be used to detect rank functions equivalent to bank functions, which is the case when only a single bit is set in the rank function mask (e. g., `2`, `8`). However, most rank function masks (e. g., `13`, `6`, `10`) have multiple bits set, e. g., combine more than one bank addressing function. Thereby, the row-conflict side-channel itself is not sufficient to detect them.

6. Row-Conflict Covert Channel on DDR5

This section presents the first row-conflict covert channel on DDR5, including cross-VM, with transmission speeds up to 2.23 Mbit/s. The sender and receiver have *no shared memory* and encode data into same-bank row conflicts, similar to previous covert channel designs: A low timing (absence of a row conflict) corresponds to a '1' and a high timing (row conflict) corresponds to a '0'.

We use a time-sliced protocol, synchronized via `rdtsc` and a 75 %-majority vote to decide whether the accesses within a time slice were mainly row conflicts or not, i. e., a '0'-bit or not. For cross-VM, the timestamp counter (TSC) can have different values but run at the same speed in each VM. Hence, we synchronize by transmitting a predefined sequence at a predefined rate. The receiver reads the sequence and can adjust its offset to the timestamp counter. After 8 bits were received, the receiver increases the offset by $\frac{1}{20}$ of the specified transmission window.

Suppose the receiver encounters a byte that is either `10101010`, i. e., `0xaa`, or `01010101`, i. e., `0x55` (one bit shifted); the current offset is stored for the first valid sequence. Likewise, the offset of the first following sequence

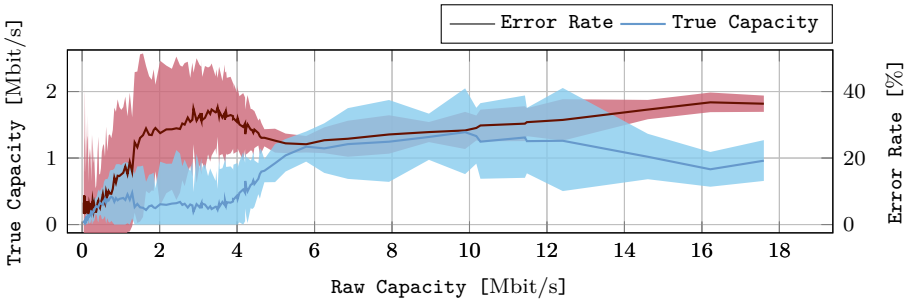


Figure 7.6.: Raw and true capacity of our covert channel on *S502* with 99 % confidence intervals.

that is neither `0xaa` nor `0x55` is stored for the following invalid sequence. Afterward, the average of both offset values is used as offset during the rest of the transmission.

Next, it is required to synchronize the border of bytes. This is done by transmitting two bytes of `0x00`. Because the sequence `10101010` ends with a `0`, the receiver should receive $1 + 8 \times 2 = 17$ zeroes in a row. After receiving 17 zeroes in a row, the receiver starts to receive a new byte. Finally, the byte `0xaa` is sent again to verify that the synchronization was successful.

Experimental Evaluation. We transmit 6 000 randomly generated bytes. The raw capacity, i. e., the number of bits sent divided by the transmission time, and the error rate, i. e., bit-edit distance divided by the number of bits, is in Figure 7.6.[3] We tested 25 gradually decreasing window sizes per system. The true capacity varies slightly with the error rate.

We perform experiments on multiple test systems and calculate the true capacity. Table 7.6 shows the raw capacity, the error rate, and the true capacity we reach. The covert channel did not work on the systems with AMD CPUs (*S403*, *S404*, *S503*, *S504*). The reason might be that Jattke et al. [12] found that AMD requires offsets for specific physical addresses, which was not considered.

On the Intel systems with DDR3, the true capacity differs significantly. This can be explained by the error rate on *S301*, which is significantly

[3] Shannon's noisy-channel coding theorem yields the true capacity T as $T = r \cdot (1 + ((1-p) \cdot \log_2(1-p) + p \cdot \log_2(p)))$.

Table 7.6.: True capacity of the covert channel on multiple systems where it worked.

System	Error Rate	Raw Capacity	True Capacity
S301	39.22 %	0.27 Mbit/s	0.01 Mbit/s
S302	21.74 %	9.14 Mbit/s	2.23 Mbit/s
S401	43.66 %	3.72 Mbit/s	0.16 Mbit/s
S402	24.01 %	3.21 Mbit/s	0.66 Mbit/s
S501	25.88 %	7.39 Mbit/s	1.29 Mbit/s
S502	28.30 %	9.89 Mbit/s	1.39 Mbit/s

higher than on *S302*, even though the raw capacity is significantly lower. The error rate on *S301* was even higher at higher capacities. Therefore, the true capacity of 0.01 Mbit/s was reached at a raw capacity of 0.27 Mibit/s. In contrast, the true capacity on *S302* is 2.23 Mbit/s at a raw capacity of 9.14 Mbit/s.

On *S401*, the error rate of 43.66 % is significantly higher than 24.01 % on *S402*. Even though a raw capacity of 3.72 Mbit/s and 3.21 Mbit/s is close, the big difference in the error rates leads to a significantly different true capacity. Therefore, the true capacity is 0.16 Mbit/s on *S401* and 0.66 Mbit/s on *S402*.

In contrast to the previous experiments, the true capacity of both DDR5 systems is similar. On *S501*, the true capacity is 1.29 Mbit/s at a raw capacity of 7.39 Mbit/s with an error rate of 25.88 %. On *S502*, the true capacity is 1.39 Mbit/s at a raw capacity of 9.89 Mbit/s with an error rate of 28.30 %.

7. Website Fingerprinting Attack

We utilize the Drama side channel to mount a website fingerprinting attack. We measure memory access patterns while accessing websites using Firefox. We hypothesize that the browser's memory access patterns depend on the website rendered at that moment. For evaluation, we train a machine learning (ML) model to classify the websites based on measured memory accesses. We verify our fingerprinting approach by classifying 100 websites with an F_1 score of 84 % on DDR4 and 74 % on DDR5.

Fingerprinting Procedure. First, we spawn a process that measures the access times to addresses on different system memory banks. At the same time, we access a website with Firefox and wait 8 s for the website to be rendered. We then stop Firefox and the measuring process, aggregate the data to reduce data size, and prepare them for ML model training or evaluation. Afterward, we use the aggregated data to train our ML model. Finally, we reaccessed the websites, measured and aggregated memory access data, and used our ML model to predict which website we had accessed.

Access Time Measurements. We take DRAM addressing functions, allocate two 2 MiB hugepages on the system. Then, we resolve the mapping of the virtual addresses to physical addresses using `/proc/self/pagemap`. Pagemap is **unnecessary** for an actual attack because Heckel et al. [8] showed that we can dynamically group addresses based on access times. However, for the purpose of demonstration, we used it to reduce the initialization time and increase the stability for the experimental evaluation. We then start n threads for measuring. Each thread measures the memory access times of a specific DRAM bank. If the measured access time was bigger than the threshold between row hit and row conflict, the access time and timestamp are stored in a buffer. We measure the loading of each website for approximately 8 s. The number of threads n is set to $n_{\text{proc}} - 2$, where n_{proc} is the number of logical CPU cores in the system. Hence, there are still two CPU cores left for Firefox and system.

Aggregation of Data. Next, we take the files created in the previous step. We specify a window size and aggregate the number of row conflicts in that window. We then store the data in a three-dimensional array. We use a window size of 100 µs. The first dimension contains the number of row conflicts within the specified window. The second dimension contains the banks, e. g., one first-dimension list for each bank measured. The third dimension contains multiple measurements; in our case, 100 accesses the same website.

Description of the ML Model. Our ML model consists of 9 convolutional layers in groups of three with max pooling and dropout layers in between. The output of the convolutional layers is then flattened, and the final prediction is made after three dense layers. The input of a single website to the model is a 3 dimensional spectrogram with the dimensions time, frequency, and DRAM bank.

Experimental Evaluation. For experimental evaluation, we access 100 websites 100 times each. Afterward, we use 80 % of the measurements to train our ML model and 20 % to test our model. On a test system with DDR5 (*S502*), we reach an overall F_1 score of 74 % and plot the predictions of our model in Figure 7.7b. On a test system with DDR4 (*S401*), we reach an overall F_1 score of 84 % and plot the predictions of our model in Figure 7.7a. *S502* has 24 logical cores (22 threads for measurement) and *S401* has 20 logical cores (18 threads for measurement). Because each thread measures a single DRAM bank, we can measure 16 of 16 banks on *S401*, so we measure accesses of Firefox to all DRAM banks. On *S502*, we can only measure 22 of 64 banks, so 34.38 % of the DRAM accesses performed by Firefox (assuming equal distribution of accesses over all banks). We hypothesize that this is the reason for the lower accuracy on *S502*.

8. Conclusion

In this paper, we introduced a novel approach to reliably verifying DRAM addressing functions and function components from software. A first systematic analysis of 5 DRAM function reverse-engineering tools on 10 different system configurations showed significant variance in the success rate of these tools, from 0 % to 92.9 %. We discovered the previously unknown rank selection side channel and reverse engineer its function on two DDR4 and two DDR5 systems. These results enable novel DDR5 row-conflict side-channel attacks, which we demonstrated in two scenarios: a covert channel with 1.39 Mbit/s, and a website fingerprinting attack with an F_1 score of 84 % on DDR4 and 74 % on DDR5. We conclude that as reverse-engineering of DRAM address functions remains relevant, our new verification methodology provides a cheap and reliable alternative to verification using expensive physical measurements.

Acknowledgment

This work was funded by the Deutsche Forschungsgemeinschaft under grant number 503876675 and the Austrian Science Fund under grant number 10.55776/I6054, as well as the European Union under grant number ROF-SG20-3066-3-2-2.

This preprint has not undergone any post-submission improvements or corrections. The version of Record of this contribution will be published in the proceedings *Computer Security - ESORICS 2025*

References

[1] AMD. BIOS and Kernel Developer's Guide (BKDG) for AMD Family 16h Models 00h-0Fh Processors. 2015. URL (p. 104).

[2] Alessandro Barenghi, Luca Breveglieri, Niccoló Izzo, and Gerardo Pelosi. Software-only reverse engineering of physical DRAM mappings for rowhammer attacks. In: International Verification and Security Workshop (IVSW). 2018 (p. 104).

[3] Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack. In: ISCA. 2022 (p. 103).

[4] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In: S&P. 2020 (pp. 101, 104, 105, 107, 108).

[5] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In: S&P. 2018 (pp. 100, 101, 104).

[6] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016 (p. 104).

[7] Youngkwang Han and John Kim. A Novel Covert Channel Attack Using Memory Encryption Engine Cache. In: DAC. 2019 (p. 103).

[8] Martin Heckel and Florian Adamsky. Reverse-Engineering Bank Addressing Functions on AMD CPUs. In: DRAMSec Workshop. 2023 (pp. 101, 104, 105, 107, 108, 119).

[9] Christian Helm, Soramichi Akiyama, and Kenjiro Taura. Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE. 2020 (pp. 101, 104, 105).

[10] Suman Jana and Vitaly Shmatikov. Memento: Learning Secrets from Process Footprints. In: S&P. 2012 (p. 103).

[11] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: Rowhammering in the Frequency Domain. In: S&P. 2021 (p. 104).

[12] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms. In: USENIX Security. 2024 (pp. 101, 104, 105, 107, 108, 117).

[13] Jonas Juffinger, Fabian Rauscher, Giuseppe La Manna, and Daniel Gruss. Secret Spilling Drive: Leaking User Behavior through SSD Contention. In: NDSS. 2025 (p. 103).

[14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: ISCA. 2014 (pp. 100, 104).

[15] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In: USENIX Security. 2022 (p. 104).

[16] Ajaykumar Kushwaha, Ajay Jain, Mahendra Patel, and Biswabandan Panda. Golmaal: Thanks to the Secure TimeCache for a Faster DRAM Covert Channel. In: DRAMSec. 2022 (p. 103).

[17] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In: S&P. 2020 (p. 101).

[18] Butler W Lampson. A note on the confinement problem. In: Communications of the ACM (1973) (p. 103).

[19] Lukas Maar, Jonas Juffinger, Thomas Steinbauer, Daniel Gruss, and Stefan Mangard. KernelSnitch: Side-Channel Attacks on Kernel Data Structures. In: NDSS. 2025 (p. 103).

[20] Michele Marazzi and Kaveh Razavi. RISC-H: Rowhammer Attacks on RISC-V. In: DRAMSec Workshop. 2024 (p. 104).

[21] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Reverse Engineering Intel Complex Addressing Using Performance Counters. In: RAID. 2015 (p. 103).

[22] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. C5: Cross-Cores Cache Covert Channel. In: DIMVA. 2015 (p. 103).

[23] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In: NDSS. 2017 (p. 103).

[24] Thomas Moscibroda and Onur Mutlu. Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems. Tech. rep. Feb. 2007 (p. 100).

[25] Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda. Reducing Memory Interference in Multi-Core Systems via Application-Aware Memory Channel Partitioning. In: MICRO. 2011. DOI: 10.1145/2155620.2155664 (p. 100).

[26] Keisuke Okamura and Yoshihiro Oyama. Load-Based Covert Channels between Xen Virtual Machines. In: Symposium on Applied Computing (SAC). 2010 (p. 103).

[27] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security. 2016 (pp. 100, 101, 103–105, 107, 108).

[28] Antoon Purnal and Ingrid Verbauwhede. Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache. In: arXiv:1908.03383 (2019) (p. 103).

[29] Yi Qin and Chuan Yue. Website Fingerprinting by Power Estimation Based Side-Channel Attacks on Android 7. In: TrustCom/BigDataSE. 2018 (p. 103).

[30] Fabian Rauscher, Andreas Kogler, Jonas Juffinger, and Daniel Gruss. IdleLeak: Exploiting Idle State Side Effects for Information Leakage. In: NDSS. 2024 (p. 103).

[31] Gururaj Saileshwar, Christopher W Fletcher, and Moinuddin Qureshi. Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion. In: ASPLOS. 2021 (p. 103).

[32] Mark Seaborn. Exploiting the DRAM rowhammer bug to gain kernel privileges. 2015. URL (p. 104).

[33] Mark Seaborn. How physical addresses map to rows and banks in DRAM. 2015. URL (p. 101).

[34] Mark Seaborn and Thomas Dullien. Exploiting the DRAM Rowhammer bug to gain kernel privileges. In: Black Hat USA. 2015 (p. 101).

[35] Benjamin Semal, Konstantinos Markantonakis, Raja Naeem Akram, and Jan Kalbantner. Leaky Controller: Cross-Vm Memory Controller Covert Channel On Multi-Core Systems. In: ICT Systems Security and Privacy Protection (SEC). 2020 (p. 103).

[36] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. Robust Website Fingerprinting Through The Cache Occupancy Channel. In: USENIX Security. 2019 (p. 103).

[37] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. Exploiting data-usage statistics for website fingerprinting attacks on Android. In: ACM Conference on Security & Privacy in Wireless and Mobile Networks. 2016 (p. 103).

[38] Kshitij Sudan, Niladrish Chatterjee, David Nellans, Manu Awasthi, Rajeev Balasubramonian, and Al Davis. Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement. In: ASPLOS. 2010. DOI: 10.1145/1735970.1736045 (p. 100).

[39] Victor van der Veen and Ben Gras. DramaQueen: Revisiting Side Channels in DRAM. In: DRAMSec. 2023 (p. 103).

[40] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. Dramdig: A Knowledge-assisted Tool to UncoverDRAM Address Mapping. In: Design Automation Conference (DAC). 2020 (pp. 101, 104, 105, 107, 108).

[41] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In: CCS. 2017 (pp. 100, 103).

[42] Zhenghong Wang and Ruby B Lee. Covert and Side Channels due to Processor Architecture. In: ACSAC. 2006 (p. 103).

[43] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the Hyper-space: High-bandwidth and Reliable Covert Channel Attacks inside the Cloud. In: ACM Transactions on Networking (2014) (p. 103).

[44] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud. In: USENIX Security. 2012 (p. 103).

[45] Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of L2 cache covert channels in virtualized environments. In: CCSW. 2011 (p. 103).

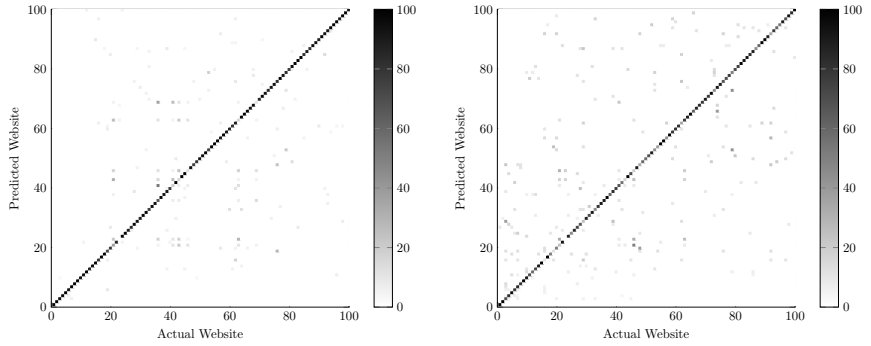
[46] Yuval Yarom and Katrina Falkner. Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. In: USENIX Security. 2014 (p. 103).

[47] Ruiyi Zhang, Taehyun Kim, Daniel Weber, and Michael Schwarz. (M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels. In: USENIX Security. 2023 (p. 103).

[48] Zhi Zhang, Wei He, Yueqiang Cheng, Wenhao Wang, Yansong Gao, Minghua Wang, Kang Li, Surya Nepal, and Yang Xiang. BitMine: An end-to-end tool for detecting rowhammer vulnerability. In: TIFS 16 (2021), pp. 5167–5181 (p. 104).

9. Appendix

9.1. Confusion Matrix Website Fingerprinting



(a) Website confusion matrix on DDR4 (*S401*), with F_1 score 86.7 %.

(b) Website confusion matrix on DDR5 on *S502*, with F_1 score 74.0 %.

Figure 7.7.: Confusion matrices. The actual website is shown on the x axis, the website predicted by the model is shown on the y axis.

8

Epistemology of Rowhammer Attacks: Threats to Rowhammer Research Validity

Publication Data

Martin Heckel, Hannes Weissteiner, Florian Adamsky, and Daniel Gruss. Epistemology of Rowhammer Attacks: Threats to Rowhammer Research Validity. In: ESORICS. 2025

Contributions

Main author.

Epistemology of Rowhammer Attacks: Threats to Rowhammer Research Validity

Martin Heckel[1,2], Hannes Weissteiner[2], Florian Adamsky[1], Daniel Gruss[2]
firstname.lastname@hof-university.de
firstname.lastname@tugraz.at
[1]*Institute of Information Systems (iisys) Hof University of Applied Sciences* Hof, Germany
[2]*Institute of Information Security (ISEC) Graz University of Technology* Graz, Austria

Abstract

The Rowhammer effect is a disturbance error in DRAM that attackers can trigger from software. The first publication on Rowhammer in 2014 evaluated 129 Dual In-line Memory Modules (DIMMs) on an FPGA and showed that 110 DIMMs are affected, indicating that Rowhammer is a widespread issue. However, until now, no case outside of academia is known in which Rowhammer was used for attacks, indicating a stark discrepancy between the attention Rowhammer receives and its real-world relevance.

This paper systematically analyzes 32 offensive Rowhammer papers, including 48 experiments. However, we avoid finger-pointing but identify six threats to the validity and relevance of Rowhammer research results and give multiple examples. The threats include small sample sizes, overestimated attacker capabilities, unrealistic attack scenarios, non-comparability of the results, age and wear of hardware, and sub-optimal attack performance metrics. Additionally, we provide recommendations with detailed justification to the scientific community to mitigate those threats: (1) pre-experimental testing of DIMM integrity, (2) increasing and broadening the DIMM sample size, (3) expanding reproduction studies of published work, (4) defining attacks in real-world conditions and distinguishing them from theoretical ones, (5) publishing DIMM manufacturing data, (6) documenting DIMM wear and, (7) leveraging multiple metrics for bit flip evaluations.

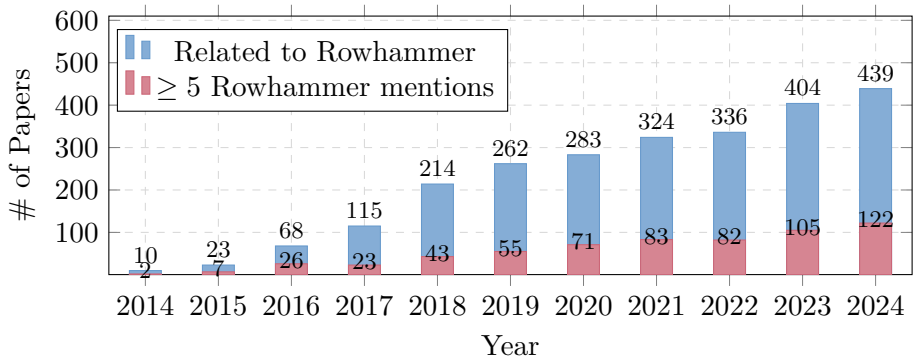


Figure 8.1.: Number of published research papers related to Rowhammer per year, and number of papers that mention rowhammer multiple times. We analyzed 2 509 papers identified by a Google Scholar search and counted the number of occurrences of the word "Rowhammer". If a paper has ≥ 5 occurrences of that word, we count it as a Rowhammer paper. This metric might include some paper that focus on another topic, but still provides an estimation of the number of publications related to Rowhammer.

1. Introduction

The main memory, Dynamic Random-Access Memory (DRAM), remains crucial in all computer devices. The demand for higher storage capacity yields a high density of DRAM memory cells. However, the industry has reached a point where scaling becomes a problem. Scaling the capacitors and transistors beyond 40 nm is challenging [49] and can result in disturbance errors.

These disturbance errors were initially assumed to have little to no security implications [73]. Later, Kim et al. [42] showed that an attacker can trigger bit flips in DRAM rows by reading from nearby rows rapidly, which is known as *Rowhammer*. In recent years, researchers developed sophisticated exploits based on Rowhammer. These exploits achieve, for instance, privilege escalation on desktop computers [74, 21, 69, 26, 1, 78, 19, 14, 47, 28, 72, 82, 12, 54, 39, 30, 37], mobile devices [85, 91, 14, 51, 45], and even on cloud systems [70, 5, 87, 79], all without a software vulnerability. Over the years, the number of scientific papers related to Rowhammer[1] increased, as shown in Figure 8.1.

[1]The results for the keyword "hammer" were almost identical.

With so many scientific publications, system administrators ask: *Should we integrate Rowhammer into our threat analysis?* However, to the best of our knowledge, Rowhammer has not been used in real-world attacks, such as malware or ransomware. It might be unrealistic to see malware or ransomware based on Rowhammer, but *we don't know if Rowhammer would be an attack vector usable for such attacks.* National or state actors could use Rowhammer as part of their attack chain. Overall, the lack of real-world attacks contradicts the number of Rowhammer publications from academia. There is a stark discrepancy between the attention Rowhammer's research has in the academic community and the relevance of Rowhammer in real-world attacks.

In this paper, we show multiple threats to Rowhammer research validity and discuss their influence on the overall validity of Rowhammer research. We analyze 32 publications that perform 48 experiments regarding these threats and show how relevant these threats are regarding these publications. We focus on offensive Rowhammer research since these publications typically perform experiments on how good attacks work on specific systems, resulting in the difference between academic and real-world estimation of exploitability. Finally, we show how researchers can prevent those threats in future research.

We point out cases where specific threats undermined the validity of previous work's experimental evaluation. *Identifying these threats would not have been possible without the tremendous effort put into these prior works.* We crucially build upon them for identification and do not want to point fingers at previous work. Instead, we want to provide recommendations to improve the validity of Rowhammer's research in general for future work.

We identify the following threats to Rowhammer research validity:

$\mathcal{T}$1 Small Sample Sizes. Most of the publications related to Rowhammer use a small sample size for their experimental evaluation, sometimes only a sample size of 1. Small sample sizes are insufficient to show that an attack works in general, and it raises the question of the prevalence of Rowhammer. An attack might only work under specific conditions or not work and yield results because the Dual In-line Memory Module (DIMM) is not functioning correctly, e. g., Target Row Refresh (TRR) is not working properly.

$\mathcal{T}$2 Overestimated Attacker Capabilities. The assumption of unrealistic capabilities of attackers leads to an overestimation of the impact of

an attack. Many attacks require specific preconditions, e. g., elevated privileges to get the physical addresses mapped to virtual addresses or access to 1 GB hugepages, etc. Some preconditions render an attack ineffective for practical exploitation, e. g., requiring elevated privileges to perform a privilege escalation attack.

$\mathcal{T}$3 Unrealistic Attack Scenarios. In some publications, the authors use special hardware like FPGAs to have fine control over the DRAM commands or overclock DIMMs in the BIOS. These are unrealistic attack scenarios, and it is unclear if such attacks would work in a real-world scenario.

$\mathcal{T}$4 Results are not comparable with other Publications. The susceptibility of systems to Rowhammer depends strongly on the system itself and environmental parameters. For example, Orosa et al. [65] showed that the number of bit flips triggered depends on the temperature. Due to the lack of specifying and monitoring environmental parameters and the fact that each research group uses different systems for experimental evaluation, the results of multiple experimental evaluations are not comparable.

$\mathcal{T}$5 Age and Wear of Hardware not specified. There are indicators that Rowhammer bit flips may "burn in", similar to the malicious aging of circuits [41]. Thus, when a specific bit flip is triggered many times in DRAM, the number of activations to trigger the bit flip can decrease. TRR, a proprietary Rowhammer mitigation, might have to be adjusted over time to mitigate new patterns or improve performance. Therefore, the age of a DIMM is relevant information for estimating specific properties of a DIMM.

$\mathcal{T}$6 The Number of Bit flips is a bad Comparison Metric. Typically, the number of bit flips is used as a comparison metric. However, it strongly depends on the system used for experimental evaluation. Therefore, it is impossible to compare the effectiveness of different existing approaches without repeating them in the same setup. Additionally, this metric does not provide any information regarding exploitability.

Contributions. Our work makes the following contributions:

1. We perform a meta-analysis and evaluate potential threats to Rowhammer research validity using 32 publications that performed 48 experiments.

2. We identify six threats to the validity of Rowhammer research and provide a detailed justification.

3. We identify 8 recommendations to our community that help mitigate threats to validity in Rowhammer research.

Outline. Section 2 provides background. Section 3 overviews threats to Rowhammer research validity. Section 5 analyzes sample sizes of prior work, Section 6 analyzes attack scenarios, Section 7 analyzes empirical results and comparability, and Section 8 discusses the influence of aging and wear. Section 9 analyzes comparison metrics from prior works. Section 10 concludes.

2. Background and Related Work

This section provides background on DRAM, Rowhammer, and related work.

DRAM. In DRAM, data is stored in cells consisting of capacitors and transistors, organized in an array of rows and columns. A *wordline* connects all transistors in a row, i. e., all cells in a row are accessed at once. The charge from the cells is amplified and forwarded to the *row buffer* (either SRAM or a feedback loop of the *bitlines*). Reading a cell drains the capacitor's charge, i. e., the row buffer has to be written back to the DRAM array before another row can be loaded. The memory controller must periodically refresh capacitors that lose charge over time. DDR3 [32] and DDR4 [33] use a refresh interval of 64 ms for each cell, and DDR5 [34] uses 32 ms, i. e., refresh commands must be issued for each row within this interval. Refreshes are typically performed in batches [18].

DRAM banks are located on multiple DRAM chips, and are organized in *ranks*, with one or more ranks on a DIMM. DIMMs are connected to the CPU with buses called *channels*.

DRAM Addressing. The kernel maps virtual to physical addresses using page tables. Physical addresses are mapped to different devices and their spatial components by the memory controller. For DRAM, the memory controller determines, e. g., channel, DIMM, rank, bank, row, and column, using *DRAM addressing functions*. These functions can be linear, essentially an XOR combination of physical address bits, or non-linear.

Addressing functions were published for some models [3, 25] but not recent ones.

Reverse-engineering linear DRAM addressing functions has been demonstrated using, e. g., timing [67, 86, 14, 23, 30] and performance counters [24]. However, non-linear DRAM addressing functions remain a challenge.

Rowhammer. When two rows in the same DRAM bank are accessed alternatingly, they are loaded into the row buffer and written back every time they are accessed, incurring numerous accesses the DRAM array. A high number of accesses to the DRAM array can lead to disturbance errors, typically in spatially nearby cells [42], called *Rowhammer*. If DRAM cells leak enough charge, their value is inverted at the sense amplifier. These bit flips have to happen before the next refresh, as cell charge is restored at refresh, i. e., fully charged or discharged. The accessed rows are called *aggressor rows*, and the rows likely to have bit flips afterward are called *victim rows*. Initially, different patterns like Single-Sided [42], Double-Sided [21], or One-Location [19] were used. Newer approaches [14, 28, 30] fuzz these patterns to bypass TRR.

In 2014, Kim et al. [42] published the first scientific analysis of Rowhammer. They showed that 110 of the 129 DIMMs in their FPGA-based setup are affected by Rowhammer. They also demonstrated bit flips on one Intel Sandy Bridge, Ivy Bridge, Haswell, and one AMD Piledriver system using one 2 GB DDR3 DIMM.

In 2015, Seaborn and Dullien [75] presented two Rowhammer exploits: A NaCl sandbox escape and a local privilege escalation based on flipping bits in page-table entries (PTEs). One year later, Razavi et al. [70] showed that Rowhammer can be exploited in a cross-VM scenario.

In 2014, vendors started to deploy mitigations against Rowhammer [8]. One of the first approaches was to double the refresh rate, as suggested by Kim et al. [42]. However, they already reported that lowering the refresh interval from 64 ms to 8.2 ms may degrade performance by 11 % to 35 %.

Another approach is to use Error Correction Code (ECC) DRAM to correct bit flips. However, Cojocar et al. [9] showed that even ECC does not prevent Rowhammer when high numbers of bit flips occur. Later, vendors introduced Target Row Refresh (TRR), a mechanism that tracks DRAM accesses and refreshes potential victim rows between regular refreshes. TRR implementations are proprietary and adjusted for new DIMMs when new attacks are published. Still, multiple publications bypassed TRR [19,

14, 28, 30]. However, there are multiple other approaches for mitigations: Some are based on counting activations [64, 4], and some on the location of rows in DRAM [7], some on cryptographic checksums [36].

Related Work. Mutlu and Kim [59] were the first to provide a retrospective of Rowhammer attacks and defenses. They surveyed the existing research papers at that time and discussed them in detail. Additionally, they focused on their previously proposed hardware mitigation PARA [42]. They also discussed Rowhammer attacks on other memory technologies, such as NAND flash.

Loughlin et al. [53] created a taxonomy for existing mitigations and proposed a memory controller extension against future attacks. They also described the limitations of countermeasures and argued that there is a disconnect between existing hardware and proposed software mitigations from the community. On the meta-level, they suggested DRAM vendors should publish precise information about their defenses to help build more effective mitigations.

Naseredini [60] surveyed of Rowhammer attacks and defenses, categorizing research into attack techniques and mitigation strategies. He analyzed them year by year and created an overview of different approaches over the years.

Recently, Zhang et al. [93] systematized Rowhammer attacks and defenses on commodity systems. They establish a unified framework to analyze Rowhammer attacks, grouping them by origins, methodologies, and objectives. They also classify various defense mechanisms including ECC and TRR.

These works provide an excellent overview of Rowhammer but do not systemically analyze problems in the research methodology that threaten validity.

3. Threats to Rowhammer Research Validity

In this section, we describe six threats to Rowhammer research validity. We identify potential problems and propose mitigations to establish a rigorous scientific process for future Rowhammer research, based on a representative

set of Rowhammer publications. The results of these high-quality, peer-reviewed publications led to the insights and recommendations presented in this paper.

3.1. $\mathcal{T}1$ Sample Sizes is Too Small

With small sample sizes, deriving general claims in empirical settings is impossible. Testing a Rowhammer attack on a single DIMM shows an attack is theoretically possible. However, a DIMM is a complex piece of electronics. Multiple potential causes exist for bit flips [58]:

- **Bad memory cells** can introduce random bit-flips.
- **Temperature** outside the operating range can impact reliability.
- **Cosmic rays** can hit DIMMs, yielding completely random bit flips.
- **Voltage fluctuations** by the power supply can introduce faults.
- **Manufacturing variations** can make a DIMM more vulnerable.
- **Electrical properties** of the motherboard (e. g., path length differences, impedance issues, or faulty contacts) can affect reliability.

Some attacks may only work due to undocumented preconditions or faulty hardware. These attacks are not reproducible, reducing trust in their validity. To reduce the influence of these factors, a higher sample size is required, ideally using different test systems. Additionally, higher sample sizes allow for the estimation of the prevalence of Rowhammer, i. e., the fraction of affected DIMMs.

A reasonable estimation of the prevalence of Rowhammer is essential: If the estimate is too low, Rowhammer research may become underrepresented despite of it's high impact. If the estimate is too high, too much effort might be put into solving a problem that only has little real-world implications.

3.2. $\mathcal{T}2$ Dependence on Elevated Attacker Privileges

In 2015, Seaborn and Dullien [75] demonstrated two exploits based on Rowhammer: A NaCl sandbox escape and a local privilege escalation based on PTEs. Consequently, obtaining virtual-to-physical address mappings was made privileged [44]. In newer attacks, other concepts like uncached memory [45], Transparent Hugepages (THPs) [70, 39], or 1 GB Hugepages [28, 30] were used. Many exploitation techniques from prior work rely on very particular prerequisites and *have been mitigated as a*

reaction to the publication of these techniques by changing default configurations or requiring elevated privileges for vulnerable interfaces. Therefore, most systems with default configurations do not meet these prerequisites anymore. Elevated attacker privileges make the attack more difficult to reproduce and may decrease trust in the empirical results. As a result, Rowhammer research may become a niche area where findings are only relevant to other Rowhammer studies and lack broader implications.

3.3. $\mathcal{T}3$ Uncertain Practical Applicability

Another threat to the validity of Rowhammer research is the uncertain practical applicability of results on off-the-shelf hardware. Some experimental evaluations of Rowhammer attacks are performed on specialized hardware, e. g., FPGAs, with the advantage of fine-grained control over DRAM commands.

Additionally, some Rowhammer attacks work on commodity hardware, yet require extreme parameters for DRAM operation, e. g., extreme overclocking. Thus, these attacks require physical access and control over firmware settings.

Rowhammer simulators like Hammertime [76] and Hammulator [81] enable faster development of Rowhammer attacks and defenses by providing faster and more deterministic bit flips. However, while this enables better comparability of different Rowhammer attacks, it has the disadvantage of not being a real system. Emulators provide good metrics for comparisons, but replacing experimental evaluation with simulators might increase the difference between academic results and the real-world exploitability of Rowhammer. Such research is essential for understanding the Rowhammer effect. However, such foundational research cannot be directly applied to real-world attacks. Follow-up work is needed.

3.4. $\mathcal{T}4$ Comparability across Publications

The position and number of bit flips during a Rowhammer attack depend on environmental parameters such as temperature [65]. Additionally, they rely on the systems and DIMMs that are evaluated. Thus, directly comparing different approaches is impossible, as most publications use different setups.

In some publications, the experimental setups are not described sufficiently. For example, CPU models, DIMM model numbers, Kernel versions, etc. are often missing. The memory controller is directly integrated into the CPU. Thus, different CPU models may have different memory access behaviors. Other kernel versions may influence the attack. For example, the change in the permission of `/proc/pid/pagemap` [44] made the attacks more difficult, as users cannot obtain physical memory addresses. Thus, due to undocumented hard- and software, experiments are often not reproducible anymore.

The physical environment is often not documented, e. g., the temperature of the DIMMs depends on whether the test system is in an office environment or a climate-controlled server room. Therefore, the environmental effects that affect experimental results are unknown, making it hard to compare them.

Another problem is that different DIMMs, even if they are the same model, are affected differently by Rowhammer [50]. While one DIMM might yield a high number of bit flips, another DIMM of the same model might not be susceptible to Rowhammer at all. This diversity makes results hard to reproduce and hinders comparing novel and existing attacks.

3.5. $\mathcal{T}5$ Unspecified Age and Wear of Hardware

Typically, the DIMMs used in experimental setups are not documented. Scientific papers aim for general applicability rather than singling out specific manufacturers, but documentation of the used hardware is essential for reproducibility. Additionally, aging generally affects circuits and their reliability, and Karimi et al. [41] showed that this can be sped up maliciously. Thus, the DIMMs' manufacturing date and wear are crucial to contextualizing the experimental evaluation.

In DRAM, bit flips induced by Rowhammer can "*burn in*" [41], i. e., they can become more likely when triggered many times. Consequently, the susceptibility of DIMMs used for Rowhammer research has increased over the years.[2] Typically, the usage in prior Rowhammer experiments is not documented for the DIMMs used in the experimental evaluation. Therefore,

[2] Karimi et al. [41] show that ICs degrade over time in general. Following, we strongly assume that the same effect occurs in DRAM, which is implemented with ICs. If future publications would specify information on age and wear, this claim could be verified. This note was not included in the camera-ready version of the paper.

it is hard to compare the effectiveness of attacks between experiments on different DIMMs.

The algorithms used for Rowhammer attack detection in TRR are not specified, most likely differ between manufacturers and even DIMM models. Therefore, effectiveness of TRR depends on the specific model or even manufacturing date of a DIMM. Vendors may adjust TRR to mitigate published attacks in newer DIMM generations. However, when the DIMM model is unknown, it is impossible to estimate which specific attacks are mitigated by TRR.

Age and usage of a DIMM may affect the Rowhammer susceptibility and, thus, they may influence the results of empirical evaluations. However, both parameters are usually not documented, increasing the difficulty of reproducing results. In addition, it decreases the comparability of publications. Both effects might reduce the trust in experimental results.

3.6. $\mathcal{T}6$ Suboptimal Metrics for Comparison

In the current research, the susceptibility of a system to Rowhammer is often expressed in the number of bit flips found in a given time or memory area. However, these metrics are not standardized. For example, Kang et al. [39] used bit flips per hour. Other work [61] used minimal activations until the first bit flips occurred, which is also known as *hammer count*. In contrast, Jattke et al. [28] used multiple measurements, including a total bit flips found in a given time and total number of bit flips over a sweep[3] of 256 MiB. Ridder et al. [72] used the percentage of times they observed bit flips at a vulnerable location. Thus, the metrics presented in different publications can not be used to compare the performance of attacks across publications. Therefore, to compare a novel attack to existing work, researchers must reproduce the prior attack on their hardware with their measurements. Due to the limited reproducibility of Rowhammer attacks, this is an unrealistic demand. Thus, the number of unique exploitable bit flips can be a better metric to estimate the performance of novel attacks.

Different exploitation strategies depend on *exploitable* bit flips, e. g., bit flips that occur at specific offsets and in particular directions. Typically, only one exploitable bit flip is required for a successful exploit chain. Thus, the attack runtime until the first exploitable bit flip may express a good

[3] When Blacksmith [28] found an effective pattern, it *sweeps* over the same contiguous memory region and reports the number of bit flips.

estimation of the real-world applicability of a specific attack. Additionally, new insights on exploitation techniques may lead to novel exploit chains, allowing better estimations of the importance of Rowhammer outside the academic world. While the raw number of bit flips, either scaled by time or a number of accesses, can still be used to compare different attack strategies on a consistent test setup, it does not provide a universally comparable metric across different machines.

4. Methodology

We started with a Google Scholar search for the word "Rowhammer" and found 2 509 publications. Google Scholar also includes publications that mention the word only once. Thus, we checked if a paper has ≥ 5 occurrences of that word; we ended up with 463 publications (including presentation, bachelor theses, etc.). Then, we manually filtered for peer-reviewed papers that perform Rowhammer attacks and ended up with 55 papers. Then, we filtered for papers in highly ranked conferences (CORE ranking A or A*) and ended up with 22 papers. After that, we had multiple meetings with researchers from different groups that have published Rowhammer attack papers in the past to discuss the papers and ask if relevant papers were missing. In the end, we selected 32 publications with 48 experimental evaluations. Some of the selected studies include experimental evaluations for different approaches to different types of systems. The list of papers we used in our analysis is in Appendix 10.

5. Analysis of Sample Sizes

We survey the sample sizes of 32 publications with 48 experimental evaluations. The results are shown in Figure 8.2.

The average sample size (e. g., tested DIMMs, mobile phones, single-board computers, etc. depending on the experiment) is 10.60, and the median sample size is 3.5, while most experiments used a sample size of 1. Of 48 experiments we analyzed, 16 used a sample size of 1, which limits the ability to draw general conclusions. We cannot exclude the possibility that these experiments depend on broken or faulty DIMMs and do not work on other systems. For these experiments, there is no information regarding the prevalence.

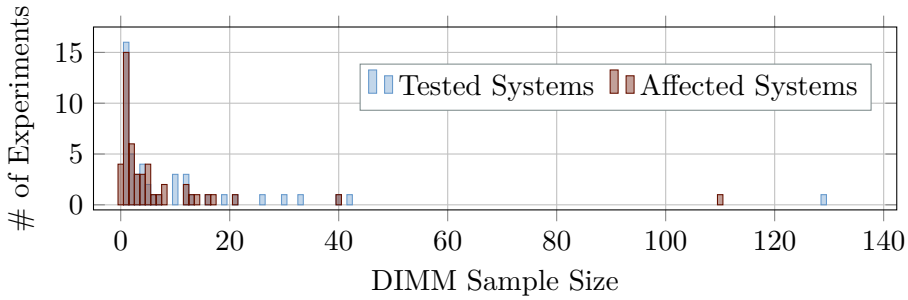


Figure 8.2.: Number of experiments for specific sample sizes and number of affected items.

There are 19 experiments with a sample size between 2 and 10, and 6 with a sample size between 11 and 20. Three experiments have a sample size between 21 and 30, 2 between 31 and 40, and 1 between 41 and 50. After that, there is a gap until 129 DIMMs are analyzed by Kim et al. [42]. On average, 7.33 items are affected with a median of 2.0. Most experiments report 1 affected item.

The number of affected items (DIMM, Mobile Device, etc. depending on the experimental setup) was 0 in 4 experiments, so a specific attack or approach did not work. We group experiments based on the type of system verified, so it is possible that an attack worked on one system type but not the other: The experimental evaluation of HalfDouble [45] shows that it works on ARM-based devices but not on any x86-based devices. Most experiments identified 1 affected item. The small affected sample size does not allow for general conclusions, as the results may depend on broken or not correctly working DIMMs.

There are 20 experiments with a number of affected items between 2 and 10. 6 experiments show that 11 to 20 items are affected, 1 experiment reports 21, and 1 experiment reports 40 affected items. The experiment from Kim et al. [42] reports 110 affected DIMMs, the highest number of affected items. As discussed in Section 3.1, there are multiple reasons that bit flips can occur that are not caused by Rowhammer. While some effects, like cosmic rays, are uncontrollable and very rare, other issues, like bad memory cells, can affect multiple measurements. Therefore, our first recommendation, $\boldsymbol{\mathcal{R}1}$, is to test DIMMs for any faults that may affect the accuracy of the experimental results.

> **$\mathcal{R}$1**: DIMMs used in empirical research must be tested for other problems, e. g., using Memtest86 (except for integrated Rowhammer tests), to ensure that no other (non-Rowhammer) problems are present.

Our second recommendation **$\mathcal{R}$2** is to increase the sample size to ≥ 30 DIMMs. This number is more of a rule-of-thumb from the central limit theorem than a strict cut-off for every experiment. Still, it is frequently referenced as a minimum viable sample size to achieve at least some diversity and statistical reliability. Additionally, we recommend including multiple manufacturers, different capacities, and various speeds to demonstrate a broader coverage.

> **$\mathcal{R}$2**: Increase the sample size to ≥ 30 DIMMs total, spread across 3 major vendors, each with at least 2 different capacities.

Our third recommendation **$\mathcal{R}$3** is to encourage the scientific community to do more reproduction studies, like Gerlach et al. [16].

> **$\mathcal{R}$3**: Do more reproduction studies of published work to gain more insights regarding the prevalence. More venues should accept reproduction studies.

6. Dependence on Elevated Attacker Privileges

This section reviews 32 publications and analyzes the experimental setup of 48 experiments. Figure 8.3 illustrates the results. The majority (68.57 %) of experimental setups use x86 systems. We hypothesize that this is the case because many tools already exist for x86, so they can be reused and adjusted. In 6 setups (12.5 %), mobile devices, such as smartphones and Chromebooks, were analyzed. Seven experimental setups (14.6 %) use an FPGA to send commands directly to the tested DIMMs. A RISC-V-based lab system was used in 1 experimental setup. For two setups, test systems were not described in detail, making it hard to reproduce them and impossible to estimate their practical impact.

FPGA experiments do not reflect realistic attack scenarios. FPGA-based setups send commands directly to the DIMMs. While allowing for greater control over the behavior of the DIMMs, this approach does not reflect a realistic attack scenario, where an attacker can only indirectly instruct the memory controller to access specific regions. Also, these setups may use specific parameters, e. g., timings, that are unavailable or uncommon on commodity systems. Also, it may not even be possible

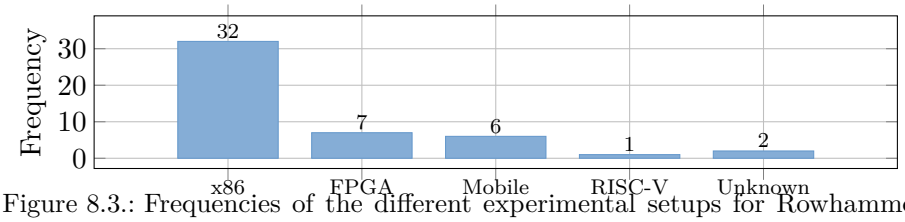


Figure 8.3.: Frequencies of the different experimental setups for Rowhammer experiments.

to configure a commodity system to use the parameters, e. g., timings, used by the FPGA-based setups. While the results of these experiments show essential insights into the DIMMs' low-level behavior, they do not represent a realistic attack scenario. Therefore, the relevance of these experiments for real-world attacks is limited.

Attacks require access to pagemap file. In Linux, `/proc/⟨pid⟩/pagemap` maps virtual to physical addresses. Access to this file is limited to privileged users since 2015 [44]. We found that 5 experimental setups require access to the pagemap file. Therefore, this requires either a severely outdated kernel or privileged access. In the first case, many other exploits, e. g., DirtyCOW [68], can be used to escalate privileges. In the second case, the attacker already has elevated privileges, so no further escalation is necessary.

Attacks require elevated privileges. We found that 6 experimental evaluations require 1 GiB Hugepages. Once these Hugepages are requested from the kernel and mounted somewhere, they can be mapped without elevated privileges. However, requesting and mounting Hugepages require elevated privileges. Therefore, these attacks are only realistic when the attacker has elevated privileges, or the system has requested and mounted 1 GiB Hugepages which are not used by another process (otherwise, the process of the attacker would not be able to map it). In the first case, no privilege escalation is necessary since the attacker already has root privileges. The latter case is exploitable but requires a specific, non-default system configuration.

Attacks require special OS settings. Razavi et al. [70] showed that exploiting Kernel Same-page Merging (KSM) combined with Rowhammer to trigger bit flips on another KVM guest on the same host is possible. However, this requires KSM to be enabled, which is not the case by default for most Linux distributions, except for special ones like Proxmox

VE[4]. The attack also requires the attacker's process to be started in the attacker's VM before the process of the victim is started—similarly, Bosman et al. [5] exploited memory deduplication on Windows with a Rowhammer attack. Memory deduplication is a feature from Hyper-V and is not enabled by default on Windows Server[5]. In our survey, we found that only 11 experimental evaluations assume a realistic attack scenario exploitable on a commodity system with default configuration. Since the prevalence of affected systems is not known as described in Section 5, no statistically significant estimations on the number of systems affected by Rowhammer can be made. Most publications introduce attacks assuming unrealistically high capabilities of the attacker or uncommon system configurations. Other publications require a custom memory controller based on an FPGA. Therefore, attacks should be classified based on the required preconditions. For example, attacks that require specific, non-standard configurations, should provide a reasonable explanation of why this configuration is realistic. Attacks that assume an unrealistically capable adversary should be clearly labeled as such. We recommend in $\boldsymbol{\mathcal{R}4}$ distinguishing between attacks that are possible in theory and attacks exploitable in a realistic experimental setup.

> $\boldsymbol{\mathcal{R}4}$: Attacks should only be classified as such when assessed under realistic attack scenarios, and there should be a more apparent distinction between actual attacks and potential (theoretical) attacks.

7. Comparability across Publications

Orosa et al. [65] showed that the number and position of bit flips depend on environmental parameters, e. g., temperature. Thus, comparing results from the same experimental setup is difficult when environmental parameters are unknown. Out of 48 experimental setups inspected, only 2 [65, 54] verified the impact of the temperature. In 2 other experiments, the authors reported a constant temperature [56, 89]. Two experiments measured the impact of the refresh interval t_{REFI}, but did not specify the temperature. 44 (91.6 %) did not specify the temperature and 46 experiments (95.83 %)

[4] At the time of publication of [70], KSM was enabled by some large cloud providers. Some of them disabled KSM as a response. This note was not included in the camera-ready version of the paper.

[5] At the time of publication of [5], Windows client machines had enabled PageCombining by default. As a response to the attack, Microsoft disabled this feature by default. This note was not included in the camera-ready version of the paper.

did not specify t_{REFI}. The refresh interval t_{REFI} is defined to be 64 ms on DDR3 and DDR4 and 32 ms on DDR5. However, some mitigations set t_{REFI} to 32 ms on DDR3. In total, 42 experimental evaluation setups did not document any environmental parameters. Typically, no experimental evaluation of prior work is performed when a new attack is published. Due to the variability between experimental setups, new attacks' performance can not be compared to prior work.

Environmental parameters known to have effects on the susceptibility of systems to Rowhammer should be controlled, monitored, and documented in future work, In the case of temperature, we recommend keeping the room at a fixed, measured temperature or measuring the temperatures with the integrated sensors of the lab systems. t_{REFI} should be measured and documented for each system. Additionally, we should encourage reproducing prior experiments on different test setups to gain some "ground truth", which allows for better confidence when comparing different approaches.

8. Unspecified Age and Wear of Hardware

DIMMs used for security research could be highly susceptible to Rowhammer because bit flips could "burn in" [41, 42]. However, most publications do not include the manufacturing date or wear of the DIMMs. This can lead to a self-increasing effect as DIMMs are used in more experiments over time [41, 42]. Since DDR4, most DIMMs support TRR, a Rowhammer mitigation based on detecting Rowhammer patterns. However, TRR is an umbrella term for many different (proprietary) vendor implementations. We assume that more recent TRR versions protect against more recent Rowhammer attacks. However, only 7 of the experimental evaluations we analyzed specify the manufacturing dates. It is impossible to estimate whether the hypothesis that older DIMMs are more strongly affected by older attacks is true. We recommend in **$\mathcal{R}$5** that authors should publish the manufacturing date of DIMMs.

> **$\mathcal{R}$5**: Authors should publish the manufacturing data of the DIMMs used in experimental evaluation.

In contrast to the manufacturing data, most papers specify the DRAM generation used in the experimental evaluation. Figure 8.4 gives an overview of different DRAM generations and the number of experimental evaluations that used them. We show that 13 experimental setups utilize DDR3 DIMMs and 22 utilize DDR4 DIMMs. In contrast, only 1 experimental

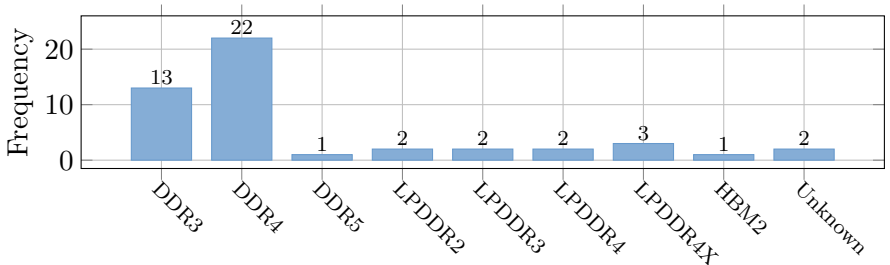


Figure 8.4.: Frequencies of different DRAM types in the analyzed experimental evaluations.

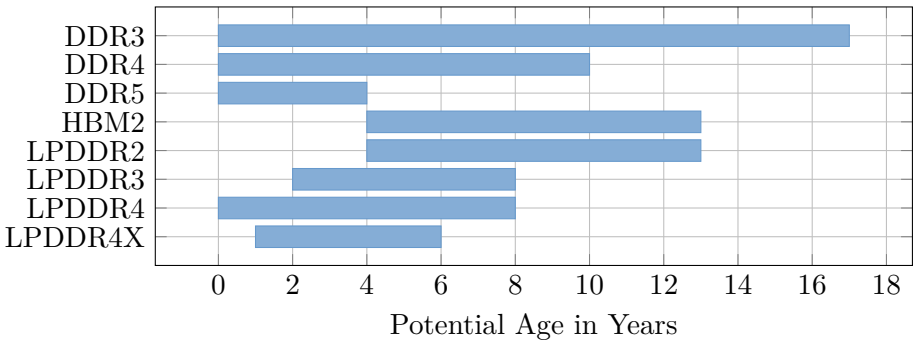


Figure 8.5.: Estimated potential age of DIMMs at the time of experimental evaluation.

evaluation was done on DDR5, even though it was released in 2020. The number of experiments performed on LPDDR is much lower: There are 2 experiments on LPDDR2, LPDDR3, and LPDDR4 each. For LPDDR4X, there are 3 experiments. Only 1 experiment analyzed Rowhammer on HBM2. Two publications did not mention which DRAM generation they used for experimental evaluation.

The generation of DRAM can be used to derive information regarding the age of the tested DIMM. Taking the year of the publication and the DRAM standard into account, and assuming that the DIMM was not manufactured after the standard for the next generation was available, we calculate the potential age of DIMMs used for experimental evaluation. The results are shown in Figure 8.5

When using this approach, the potential minimum age of a DDR3 DIMM, while used in an experiment, is 0 years for a publication from 2014 [42] when assuming the DIMM was manufactured in 2014 since the DDR4 standard

was released in 2014. The maximum age of DDR3 DIMMs used in any studies is 17 years in the case of a publication from 2024 [39] when assuming the DIMM was manufactured in 2007, immediately after the standard was launched. Different DRAM generations and standards have varying age ranges, affecting the reliability and features of DIMMs. Therefore, **$\mathcal{R}$6** states that authors should document the actual manufacturing dates of the used DIMMs. Additionally, they should provide an estimation of the wear of the DIMMs, e. g., how long and often they were used for Rowhammer evaluation. This would allow estimations on the reliability on the DIMMs, based on their age and wear.

$\mathcal{R}$6: Authors should submit information about the DIMMs' wear in experimental evaluation.

9. Suboptimal Metrics for Comparison

Kim et al. [42] were the first to use the number of bit flips as an absolute metric. This metric depends on the execution strategy, e. g., if the same memory area is scanned multiple times. Also, some works count only unique bit flips, while others count all. In that context, *uniqueness* may be based on the same memory cell, access patterns, and data stored in cells before performing the Rowhammer attack. Additionally, the number of bit flips strongly depends on the experiment's runtime or the size of the scanned memory area. This approach is used in multiple papers in our survey [42, 85, 91, 51, 54, 39, 14].

Some papers use the relative number of bit flips as an alternative. This approach aims to make the absolute number of bit flips comparable. By normalizing the number of bits flips against a reference value, e. g., the size of the scanned memory area or the scan time, this metric estimates how affected a single setup, or DIMM, actually is. However, these relative metrics are still influenced by the same factors as the absolute number of bit flips. This approach is used in multiple publications [42, 16, 62, 19, 85, 45, 87, 37, 56, 72, 79, 30]. In contrast to counting all bit flips, other publications count only exploitable bit flips. Exploitable bit flips is a better metric for estimating the impact of potential exploitation. However, this strongly depends on the definition of *exploitable*: Attacks based on bit flips in the Page Frame Number (PFN) part of a PTE [9, 28, 45, 30]. It was also shown that cryptographic algorithms can be attacked by flipping bits in the keys [70, 9, 28, 30]. Bit flips can target opcodes in binaries

and libraries [19, 9, 28, 30]. There are also attacks based on bit flips in URLs [70]. Thus, in **$\mathcal{R}$7**, authors should include multiple metrics for bit flips.

> **$\mathcal{R}$7**: Authors should use multiple metrics for bit flips to allow for better comparisons to other works.

10. Conclusion

We systematically analyzed 32 publications with 48 experimental evaluations and identified six major threats to Rowhammer's research validity. We have shown that in 33 % of the experiments, the sample size is only 1; therefore, many other factors could be the reason for bit flips. From the overall 32 x86 consumer hardware, only 22 described an approach to get physical addressing information. Half of these 22 experiments on x86 required unrealistically high capabilities of an attacker (e. g., root privileges), making them an isolated problem in academia. We found that the experimental results are often incomparable because environmental parameters are not controlled or documented, and inconsistent units for bit flips have been used. Additionally, 25 analyzed publications do not document the age and wear of used hardware. We developed the following 7 recommendations with detailed justification to improve future Rowhammer research: pre-experimental testing of the DIMMs, increasing sample size, value reproduction studies, defining attacks in real-world conditions and distinguishing them from theoretical ones, publishing more information about the used DIMMs, including wear, and using multiple units for bit flips evaluation.

Acknowledgment

This work was funded by the Deutsche Forschungsgemeinschaft under grant number 503876675 and the Austrian Science Fund under grant number 10.55776/I6054, as well as the European Union under grant number ROF-SG20-3066-3-2-2.

This preprint has not undergone any post-submission improvements or corrections. The version of Record of this contribution will be published in the proceedings *Computer Security - ESORICS 2025*

References

[1] Misiker Tadesse Aga, Zelalem Birhanu Aweke, and Todd Austin. When good protections go bad: Exploiting anti-DoS measures to accelerate Rowhammer attacks. In: HOST. 2017 (p. 129).

[2] Misiker Tadesse Aga, Zelalem Birhanu Aweke, and Todd Austin. When good protections go bad: Exploiting anti-DoS measures to accelerate Rowhammer attacks. In: HOST. 2017 (p. 158).

[3] AMD. BIOS and Kernel Developer's Guide (BKDG) for AMD Family 16h Models 00h-0Fh Processors. 2015. URL (p. 133).

[4] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. ANVIL: Software-based protection against next-generation Rowhammer attacks. In: ACM SIGPLAN Notices 51 (2016), pp. 743–755 (p. 134).

[5] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In: S&P. 2016 (pp. 129, 143).

[6] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In: S&P. 2016 (p. 158).

[7] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In: USENIX Security. 2017 (p. 134).

[8] Chromium Issue Tracker. Security: NaCl sandbox escape via DRAM "rowhammer" memory corruption. 2014. URL (p. 133).

[9] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In: S&P. 2019 (pp. 133, 146, 147).

[10] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In: S&P. 2019 (p. 158).

[11] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript. In: USENIX Security. 2021 (p. 158).

[12] Michael Fahr Jr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray Perlner, Arkady Yerukhimovich, et al. When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer. In: CCS. 2022 (p. 129).

[13] Michael Fahr Jr, Hunter Kippen, Andrew Kwong, Thinh Dang, Jacob Lichtinger, Dana Dachman-Soled, Daniel Genkin, Alexander Nelson, Ray Perlner, Arkady Yerukhimovich, et al. When Frodo Flips: End-to-End Key Recovery on FrodoKEM via Rowhammer. In: CCS. 2022 (p. 158).

[14] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In: S&P. 2020 (pp. 129, 133, 134, 146).

[15] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In: S&P. 2020 (p. 158).

[16] Lukas Gerlach, Fabian Thomas, Robert Pietsch, and Michael Schwarz. A Rowhammer Reproduction Study Using the Blacksmith Fuzzer. In: European Symposium on Research in Computer Security. 2023 (pp. 141, 146).

[17] Lukas Gerlach, Fabian Thomas, Robert Pietsch, and Michael Schwarz. A Rowhammer Reproduction Study Using the Blacksmith Fuzzer. In: ESORICS. 2023 (p. 158).

[18] Google. Measuring the DRAM refresh rate by timing memory accesses. 2015. URL (p. 132).

[19] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In: S&P. 2018 (pp. 129, 133, 146, 147).

[20] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In: S&P. 2018 (p. 158).

[21] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016 (pp. 129, 133).

[22] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016 (p. 158).

[23] Martin Heckel and Florian Adamsky. Reverse-Engineering Bank Addressing Functions on AMD CPUs. In: Workshop on DRAM Security (DRAMSec). 2023 (p. 133).

[24] Christian Helm, Soramichi Akiyama, and Kenjiro Taura. Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE. 2020 (p. 133).

[25] Intel. Intel Xeon Processor E5 v4 Product Family: Datasheet Volume 2: Registers. 2016 (p. 133).

[26] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In: SysTEX. 2017 (p. 129).

[27] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In: SysTEX. 2017 (p. 158).

[28] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: Rowhammering in the Frequency Domain. In: S&P. Nov. 2021 (pp. 129, 133–135, 138, 146, 147).

[29] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: Rowhammering in the Frequency Domain. In: S&P. 2021 (p. 158).

[30] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms. In: USENIX Security. 2024 (pp. 129, 133–135, 146, 147).

[31] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms. In: USENIX Security. 2024 (p. 158).

[32] JEDEC Solid State Technology. DDR3 SDRAM Standard. 2012. URL (p. 132).

[33] JEDEC Solid State Technology. DDR4 SDRAM Standard. 2021. URL (p. 132).

[34] JEDEC Solid State Technology. DDR5 SDRAM Standard. 2024. URL (p. 132).

[35] Sangwoo Ji, Youngjoo Ko, Saeyoung Oh, and Jong Kim. Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks. In: AsiaCCS. 2019 (p. 158).

[36] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. CSI: Rowhammer - Cryptographic Security and Integrity against Rowhammer. In: S&P. 2023 (p. 134).

[37] Jonas Juffinger, Sudheendra Raghav Neela, Martin Heckel, Lukas Schwarz, Florian Adamsky, and Daniel Gruss. Presshammer: Rowhammer and Rowpress without Physical Address Information. In: DIMVA. 2024 (pp. 129, 146).

[38] Jonas Juffinger, Sudheendra Raghav Neela, Martin Heckel, Lukas Schwarz, Florian Adamsky, and Daniel Gruss. Presshammer: Rowhammer and Rowpress without Physical Address Information. In: DIMVA. 2024 (p. 158).

[39] Ingab Kang, Walter Wang, Jason Kim, Stephan van Schaik, Youssef Tobah, Daniel Genkin, Andrew Kwong, and Yuval Yarom. SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism. In: USENIX Security. 2024 (pp. 129, 135, 138, 146).

[40] Ingab Kang, Walter Wang, Jason Kim, Stephan van Schaik, Youssef Tobah, Daniel Genkin, Andrew Kwong, and Yuval Yarom. SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism. In: USENIX Security. 2024 (p. 158).

[41] Naghmeh Karimi, Arun Karthik Kanuparthi, Xueyang Wang, Ozgur Sinanoglu, and Ramesh Karri. MAGIC: Malicious Aging in Circuits/Cores. In: ACM TACO. 2015 (pp. 131, 137, 144).

[42] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: ISCA. 2014 (pp. 129, 133, 134, 140, 144–146).

[43] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: ISCA. 2014 (p. 158).

[44] Kirill A. Shutemov. Pagemap: Do Not Leak Physical Addresses to Non-Privileged Userspace. 2015. URL (pp. 135, 137, 142).

[45] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In: USENIX Security. 2022 (pp. 129, 135, 140, 146).

[46] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In: USENIX Security. 2022 (p. 158).

[47] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In: S&P. 2020 (p. 129).

[48] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In: S&P. 2020 (p. 158).

[49] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In: International Symposium on Computer Architecture (ISCA). 2009 (p. 129).

[50] Dawei Li, Di Liu, Yangkun Ren, Ziyi Wang, Yu Sun, Zhenyu Guan, Qianhong Wu, and Jianwei Liu. FPHammer: A Device Identification Framework based on DRAM Fingerprinting. In: TrustCom. 2023 (p. 137).

[51] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. Nethammer: Inducing Rowhammer Faults through Network Requests. In: SILM Workshop. 2020 (pp. 129, 146).

[52] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. Nethammer: Inducing Rowhammer Faults through Network Requests. In: SILM Workshop. 2020 (p. 158).

[53] Kevin Loughlin, Stefan Saroiu, Alec Wolman, and Baris Kasikci. Stop! Hammer Time: Rethinking Our Approach to Rowhammer Mitigations. In: Workshop on Hot Topics in Operating Systems. 2021 (p. 134).

[54] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlikçi, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In: ISCA. 2023 (pp. 129, 143, 146).

[55] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlıkçı, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In: ISCA. 2023 (p. 158).

[56] Michele Marazzi and Kaveh Razavi. RISC-H: Rowhammer Attacks on RISC-V. In: Workshop on DRAM Security (DRAMSec). 2024 (pp. 143, 146).

[57] Michele Marazzi and Kaveh Razavi. RISC-H: Rowhammer Attacks on RISC-V. In: DRAMSec Workshop. 2024 (p. 158).

[58] Memtest86. Troubleshooting Memory Errors. 2021. URL (p. 135).

[59] Onur Mutlu and Jeremie S. Kim. RowHammer: A Retrospective. In: IEEE TCAD (2020) (p. 134).

[60] Amir Naseredini. Exploring the Horizon: A Comprehensive Survey of Rowhammer. 2023. arXiv: `2310.06950 [cs.CR]`. URL (p. 134).

[61] Ataberk Olgun, F. Nisa Bostanci, Ismail Emir Yuksel, Oguzhan Canpolat, Haocong Luo, Geraldo F. Oliveira, A. Giray Yaglikci, Minesh Patel, and Onur Mutlu. Variable Read Disturbance: An Experimental Analysis of Temporal Variation in DRAM Read Disturbance. In: HPCA. 2025 (p. 138).

[62] Ataberk Olgun, Majd Osseiran, A Giray Yağlikçi, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. An Experimental Analysis of RowHammer in HBM2 DRAM Chips. In: DSN. 2023 (p. 146).

[63] Ataberk Olgun, Majd Osseiran, A Giray Yağlıkçı, Yahya Can Tuğrul, Haocong Luo, Steve Rhyner, Behzad Salami, Juan Gomez Luna, and Onur Mutlu. An Experimental Analysis of RowHammer in HBM2 DRAM Chips. In: DSN. 2023 (p. 158).

[64] Ataberk Olgun, Yahya Can Tugrul, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F Oliveira, and Onur Mutlu. ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation. In: USENIX Security. 2024 (p. 134).

[65] Lois Orosa, Ulrich Rührmair, A Giray Yaglikci, Haocong Luo, Ataberk Olgun, Patrick Jattke, Minesh Patel, Jeremie Kim, Kaveh Razavi, and Onur Mutlu. Spyhammer: Using rowhammer to remotely spy on temperature. In: arXiv:2210.04084 (2022) (pp. 131, 136, 143).

[66] Lois Orosa, Ulrich Rührmair, A Giray Yağlıkçı, Haocong Luo, Ataberk Olgun, Patrick Jattke, Minesh Patel, Jeremie Kim, Kaveh Razavi, and Onur Mutlu. SpyHammer: Using RowHammer to Remotely Spy on Temperature. In: arXiv:2210.04084 (2022) (p. 158).

[67] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security. 2016 (p. 133).

[68] Phil Oester. CVE-2016-5195. 2016. URL (p. 142).

[69] Rui Qiao and Mark Seaborn. A New Approach for Rowhammer Attacks. In: HOST. 2016 (p. 129).

[70] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In: USENIX Security. 2016 (pp. 129, 133, 135, 142, 143, 146, 147).

[71] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In: USENIX Security. 2016 (p. 158).

[72] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript. In: USENIX Security. 2021 (pp. 129, 138, 146).

[73] Jerome H. Saltzer and M. Frans Kaashoek. Principles of Computer System Design: An Introduction. 2009 (p. 129).

[74] Mark Seaborn. Exploiting the DRAM rowhammer bug to gain kernel privileges. Mar. 2015. URL (p. 129).

[75] Mark Seaborn and Thomas Dullien. Exploiting the DRAM Rowhammer bug to gain kernel privileges. In: Black Hat USA. 2015 (pp. 133, 135).

[76] Andrei Tatar. Hammertime: a software suite for testing, profiling and simulating the Rowhammer DRAM defect. 2018. URL (p. 136).

[77] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In: RAID. 2018 (p. 158).

[78] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating software mitigations against rowhammer: a surgical precision hammer. In: RAID. 2018 (p. 129).

[79] Andrei Tatar, Radhesh Krishnan, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer Attacks over the Network and Defenses. In: USENIX ATC. 2018 (pp. 129, 146).

[80] Andrei Tatar, Radhesh Krishnan, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer Attacks over the Network and Defenses. In: USENIX ATC. 2018 (p. 158).

[81] Fabian Thomas, Lukas Gerlach, and Michael Schwarz. Hammulator: Simulate Now - Exploit Later. In: DRAMSec. 2023 (p. 136).

[82] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G Shin. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In: S&P. 2022 (p. 129).

[83] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G Shin. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In: S&P. 2022 (p. 158).

[84] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In: CCS. 2016 (p. 158).

[85] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In: CCS. 2016 (pp. 129, 146).

[86] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. Dramdig: A Knowledge-assisted Tool to UncoverDRAM Address Mapping. In: Design Automation Conference (DAC). 2020 (p. 133).

[87] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: USENIX Security. 2016 (pp. 129, 146).

[88] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: USENIX Security. 2016 (p. 158).

[89] A Giray Yağlikçi, Haocong Luo, Geraldo F De Oliviera, Ataberk Olgun, Minesh Patel, Jisung Park, Hasan Hassan, Jeremie S Kim, Lois Orosa, and Onur Mutlu. Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. In: DSN. 2022 (p. 143).

[90] A Giray Yağlıkçı, Haocong Luo, Geraldo F De Oliviera, Ataberk Olgun, Minesh Patel, Jisung Park, Hasan Hassan, Jeremie S Kim, Lois Orosa, and Onur Mutlu. Understanding RowHammer Under Reduced Wordline Voltage: An Experimental Study Using Real DRAM Devices. In: DSN. 2022 (p. 158).

[91] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. Triggering Rowhammer Hardware Faults on ARM: A Revisit. In: ASHES Workshop. 2018 (pp. 129, 146).

[92] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. Triggering Rowhammer Hardware Faults on ARM: A Revisit. In: ASHES Workshop. 2018 (p. 158).

[93] Zhi Zhang, Decheng Chen, Jiahao Qi, Yueqiang Cheng, Shijie Jiang, Yiyang Lin, Yansong Gao, Surya Nepal, Yi Zou, Jiliang Zhang, and Yang Xiang. SoK: Rowhammer on Commodity Operating Systems. In: Asia CCS. 2024. DOI: 10.1145/3634737.3656998 (p. 134).

Appendix

Table 8.1 overviews the analyzed Rowhammer studies.

Author	Pattern	Memory Type	Environment	Test Setup	Focus	Sample size	Flips observed on	Year
Kim et al. [43]	One-Location	DDR3	Unspecified	FPGA	Bit Flips	129 DIMMs	110 DIMMs	2014
Qiao2016	?	?	Unspecified	Unspecified	Exploitation	?	?	2016
Bosman et al. [6]	Double-Sided	DDR3	Unspecified	1 Lab System	Exploitation	1 DIMM	1 DIMM	2016
van der Veen et al. [84]	Double-Sided	LPDDR2	Unspecified	1 Smartphone	Exploitation, Bit Flips	1 Smartphone	1 Smartphone	2016
van der Veen et al. [84]	Double-Sided	LPDDR3	Unspecified	26 Smartphones	Exploitation, Bit Flips	26 Smartphones	17 Smartphones	2016
van der Veen et al. [84]	Double-Sided	LPDDR4	Unspecified	1 Smartphone	Exploitation, Bit Flips	1 Smartphone	0 Smartphones	2016
Razavi et al. [71]	Double-Sided	DDR3	Unspecified	1 Lab System	Exploitation	1 DIMM	1 DIMM	2016
Xiao et al. [88]	Single-Sided, Double-Sided	DDR3	Unspecified	5 Lab Systems	Exploitation, Bit Flips	5 DIMMs	4 DIMMs (only done on 4)	2016
Xiao et al. [88]	Single-Sided, Double-Sided	DDR4	Unspecified	1 Lab System	Exploitation, Bit Flips	1 DIMM	0 DIMMs (not done on DDR4)	2016
Gruss et al. [22]	Double-Sided	DDR3	Unspecified	2 Lab Systems	Bit Flips	6 DIMMs	5 DIMMs	2016
Gruss et al. [22]	Double-Sided	DDR4	tREFI	2 Lab Systems	Bit Flips	4 DIMMs	2 DIMMs	2016
Jang et al. [27]	Double-Sided	DDR4	Unspecified	1 Lab System	Exploitation, Bit Flips	1 DIMM	1 DIMM	2017
Aga et al. [2]	Single-Sided, Double-Sided	DDR4	Unspecified	1 Lab System	Bit Flips	4 DIMMs	3 DIMMs	2017
Gruss et al. [20]	One-Location	DDR3	Unspecified	2 Lab Systems	Exploitation, Bit Flips	4 DIMMs	4 DIMMs	2018
Gruss et al. [20]	One-Location	DDR4	Unspecified	1 Lab System	Exploitation, Bit Flips	2 DIMMs	2 DIMMs	2018
Tatar et al. [77]	Single-, Double-Sided, Amplified	DDR3	Unspecified	2 Lab Systems	Exploitation	33 Memory Setups?	14 Memory Setups?	2018
Lipp et al. [52]	Double-Sided	DDR4	Unspecified	3 Lab Systems	Exploitation, Bit Flips	1 DIMM	1 DIMM	2018
Lipp et al. [52]	One-Location	LPDDR2	Unspecified	1 Smartphone	Exploitation, Bit Flips	1 Smartphone	1 Smartphone	2018
Tatar et al. [80]	Double-Sided	DDR3	Unspecified	2 Lab Systems	Bit Flips	4 DIMMs	4 DIMMs	2018
Zhang et al. [92]	Double-Sided	LPDDR3	Unspecified	1 Single Board Computer	Bit Flips	1 Single Board	1 Single Board	2018
Cojocar et al. [10]	Double-Sided	?	Unspecified	?	Exploitation	?	?	2019
Ji et al. [35]	Double-Sided	DDR3	Unspecified	1 Lab System	Bit Flips	16 DIMMs	12 DIMMs	2019
Kwong et al. [48]	Single-Sided, Double-Sided	DDR3	Unspecified	1 Lab System	Bit Flips	2 DIMMs	2 DIMMs	2020
Frigo et al. [15]	Many-Sided	DDR4	Unspecified	1 Lab System	Bit Flips	42 DIMMs	13 DIMMs	2020
Frigo et al. [15]	Many-Sided	LPDDR4X	Unspecified	13 Mobile Devices	Bit Flips	13 Mobile Devices	5 Mobile Devices	2020
de Ridder et al. [11]	Many-Sided	DDR4	Unspecified	3 Lab Systems	Bit Flips	5 DIMMs	3 - 5 DIMMs (not clarified)	2021
Jattke et al. [29]	Fuzzed (Blacksmith)	DDR4	Unspecified	10 Lab Systems	Bit Flips	40 DIMMs	40 DIMMs	2022
Jattke et al. [29]	Fuzzed (Blacksmith)	LPDDR4X	Unspecified	JEDEC developer board	Bit flips	19 Chips	16 Chips	2022
Kogler et al. [46]	Half-Double	DDR4	Unspecified	FPGA	Bit Flips	3 DIMMs	2 DIMMs	2022
Kogler et al. [46]	Half-Double	LPDDR4X	Unspecified	7 Mobile Devices	Bit Flips	7 Mobile Devices	5 Mobile Devices	2022
Kogler et al. [46]	Half-Double	DDR4	Unspecified	1 Notebook	Bit Flips	1 Notebook	0 Notebooks	2022
Kogler et al. [46]	Half-Double	LPDDR4	Unspecified	2 MiniPCs	Bit Flips	2 MiniPCs	0 MiniPCs	2022
Tobah et al. [83]	Double-Sided	DDR3	Unspecified	1 Lab System	Exploitation, Bit Flips	3 DIMMs	3 DIMMs	2022
Tobah et al. [83]	Many-Sided	DDR4	Unspecified	3 Lab Systems	Exploitation, Bit Flips	3 DIMMs	3 DIMMs	2022
Orosa et al. [66]	Single-Sided	DDR4	Temperature	FPGA	Bit Flips	12 DIMMs	12 DIMMs	2022
Yağlıkçı et al. [90]	Double-Sided	DDR4	50C	FPGA	Bit Flips	30 DIMMs (272 Chips)	64 Chips	2022
Fahr Jr et al. [13]	Double-Sided	DDR3	Unspecified	1 Lab System	Exploitation	2 DIMMs	>= 1 DIMM	2022
Gerlach et al. [17]	Fuzzed (Blacksmith)	DDR4	Unspecified	4 Lab Systems	Bit Flips	10 DIMMs	8 DIMMs	2023
Olgun et al. [63]	Double-Sided	HBM2	Unspecified	FPGA	Bit flips	1 Chip	1 Chip	2023
Luo et al. [55]	Single-Sided	DDR4	Temperature	FPGA	Bit Flips	21 DIMMs	21 DIMMs	2023
Luo et al. [55]	Single-Sided	DDR4	Unspecified	1 Lab System	Bit Flips	1 DIMM	1 DIMM	2023
Juffinger et al. [38]	Fuzzed (Blacksmith)	DDR4	Unspecified	Lab Systems	Bit Flips	12 DIMMs	6 DIMMs	2024
Juffinger et al. [38]	Single-Sided	DDR4	Unspecified	Lab Systems	Bit Flips	12 DIMMs	2 DIMMs	2024
Marazzi and Razavi [57]	Double-Sided	DDR4	23C	1 Lab System (RISC-V)	Bit Flips	1 DIMM	1 DIMM	2024
Kang et al. [40]	Many-Sided	DDR3	Unspecified	1 Lab System	Bit Flips	1 DIMM	? DIMMs	2024
Kang et al. [40]	Many-Sided	DDR4	Unspecified	1 Lab System	Bit Flips	2 DIMMs	2 DIMMs (not clarified)	2024
Jattke et al. [31]	Fuzzed (Zenhammer)	DDR4	Unspecified	3 Lab Systems	Bit Flips	10 DIMMs	8 DIMMs	2024
Jattke et al. [31]	Fuzzed (Zenhammer)	DDR5	Unspecified	1 Lab System	Bit Flips	10 DIMMs	1 DIMM	2024

Table 8.1.: Overview of the analyzed Rowhammer studies.

9

FlippyRAM: A Large-Scale Study of Rowhammer Prevalence

Publication Data

Martin Heckel, Nima Sayadi, Jonas Juffinger, Carina Fiedler, Daniel Gruss, and Florian Adamsky. FlippyR.AM: A Large-Scale Study of Rowhammer Prevalence. In: NDSS. 2026

Contributions

Main author.

FlippyRAM: A Large-Scale Study of Rowhammer Prevalence

Martin Heckel[1,2], Nima Sayadi[1], Jonas Juffinger[2], Carina Fiedler[2], Daniel Gruss[2], Florian Adamsky[1]
[1]firstname.lastname@hof-university.de
[2]firstname.lastname@tu-graz.at

[1]*Institute of Information Systems (iisys) Hof University of Applied Sciences* Hof, Germany
[2]*Institute of Information Security (ISEC) Graz University of Technology* Graz, Austria

Abstract

Rowhammer is a disturbance error in Dynamic Random-Access Memory (DRAM) that can be deliberately triggered from software by repeatedly reading, i. e., hammering, proximate memory locations in different DRAM *rows*. While numerous studies evaluated the Rowhammer effect, in particular how it can be triggered and how it can be exploited, most studies only use a small sample size of Dual In-line Memory Modules (DIMMs). Only few studies provided indication for the prevalence of the effect, with clear limitations to specific hardware configurations or FPGA-based experiments with precise control of the DIMM, limiting how far the results can be generalized.

In this paper, we perform the first large-scale study of the Rowhammer effect involving 1 006 data sets from 822 systems. We measure Rowhammer prevalence in a fully automated cross-platform framework, FlippyRAM, using the available state-of-the-art software-based DRAM and Rowhammer tools. Our framework automatically gathers information about the DRAM and uses 5 tools to reverse-engineer the DRAM addressing functions, and based on the reverse-engineered functions uses 7 tools to mount Rowhammer. We distributed the framework online and via USB thumb drives to thousands of participants from December 30, 2024, to June 30, 2025. Overall, we collected 1 006 datasets from systems with various CPUs, DRAM generations, and vendors. Our study reveals that out of 1 006 datasets, 453 (371 of the 822 unique systems) succeeded in the first stage of reverse-engineering the DRAM addressing functions, indicating

that successfully and reliably recovering DRAM addressing functions remains a significant open problem. In the second stage, 126 (12.5 % of all datasets) exhibited bit flips in our fully automated Rowhammer attacks. Our results show that fully-automated, i. e., weaponizable, Rowhammer attacks work on a lower share of systems than FPGA-based and lab experiments indicated but with 12.5 % enough to be a practical vector for threat actors. Furthermore, our results highlight that the two most pressing research challenges around Rowhammer exploitability are more reliable reverse-engineering addressing functions, as 50 % of datasets without bit flips failed in the DRAM reverse-engineering stage, and reliable Rowhammer attacks across diverse processor microarchitectures[1], as only 12.5 % of datasets contained bit flips. Addressing each of these challenges could **double** the number of systems susceptible to Rowhammer and make Rowhammer a more pressing threat in real-world scenarios.

1. Introduction

Dynamic Random-Access Memory (DRAM) is the predominant main memory technology in today's computer systems. It is cost-effective, efficient, and has a large capacity, i. e., it can contain gigabytes of data. A DRAM array consisting of thousands of cells of transistors and capacitors, each storing a single bit, and has to be refreshed periodically to prevent data loss, i. e., bit flips, due to capacitor charge leakage. Disturbance effects can additionally influence capacitor charge: The Rowhammer effect [28] is triggered by frequent activations of memory rows, e. g., due to accesses, draining enough charge from nearby rows to cause inaccessible bits in memory to flip. Since Rowhammer memory access patterns can be run from unprivileged software, it can serve as a software-based fault attack, undermining the security of the entire system [43].

Since academic Rowhammer research started in 2014, it has gained significant attention from both the research community and industry, particularly in three directions for Rowhammer exploitability: The **first** line of works addresses the challenge of how to exploit bit flips, e. g., by flipping bits in the page table entries (PTEs) [43], flipping bits in secret keys [3], flipping bits in binaries [10], or flipping bits in neural network learned parameters [33]. The different works involve exploitation techniques such

[1] Since the integration and closed beta test time frame was more than two months, we could not integrate the ZenHammer tool [20] into our framework.

as templating memory for exploitable bit flips and then releasing the memory such that the victim places its own data at this specific location [43, 10], spraying memory and increasing probabilities that random bit flips are exploitable [43], or blind hammering using speculative execution as an oracle to observe whether the random bit flip occurred in an exploitable location [30]. The **second** line of works focuses on how to trigger Rowhammer and induce bit flips required for the above category of works. While initial works accessed rows alternatingly [28, 43], either in a single-sided or double-sided fashion, later works discovered that the access sequence plays a significant role [11, 8, 41, 19], that accesses to decoy rows can be necessary [8], or that it is necessary to access rows in greater distance to induce flips [30]. The **third** line of works focuses on finding memory addresses that map to specific DRAM locations, e. g., by reverse-engineering DRAM addressing functions [39, 50, 8, 14, 20] and using these functions to select suitable memory locations, or by getting physical address information which initially has been user-accessible [43] but has been removed in later versions of the Linux kernel [29]. More recent works use either huge pages [8, 19, 20], massage memory allocations for contiguity [30], or use side channels to leak physical address information or achieve contiguity [30]. However, none of these works had its main focus on the real-world prevalence of the Rowhammer effect itself.

Given this lacking understanding of the current real-world relevance of Rowhammer, we identify two research questions to **answer whether Rowhammer is a realistic and relevant real-world threat**:

RQ1 Is the currently available tooling sufficient to weaponize the Rowhammer effect for real-world exploits?

RQ2 Given Rowhammer susceptibility ranging between 30 % and 100 % as reported in prior works in lab setups, which real-world systems are attackable with Rowhammer? More specifically, is it a relevant attack vector for real-world threat actors given the variety of hardware configurations in the wild?

In this paper, we address both research questions by performing a large-scale study of Rowhammer prevalence. The basis of our large-scale study is a new framework, FlippyRAM, which allows us to run Rowhammer tests on a large number of systems with minimal user interaction and full automation of all steps to run Rowhammer on a system. The framework utilizes 5 tools for DRAM addressing function reverse-engineering and runs 7 Rowhammer tools. Afterwards, the results are uploaded to a server when

the user agrees to participate in our study. The framework is designed to be easily deployable across different hardware configurations and includes a comprehensive set of Rowhammer attack patterns.

The core of our work is a user study, in which the participants ran our framework on their systems. The study was conducted from December 30, 2024 to June 30, 2025, during which we collected a total of 1 006 datasets from 822 unique systems (users are able to run our framework multiple times on the same system). We did not collect information about which user uploaded which dataset for privacy reasons. Therefore, we do not know the exact number of participants. The participants were recruited primarily by distributing thousands of USB thumb drives with our framework at conferences and events, as well as through online channels.

We address **RQ1** by evaluating whether current Rowhammer tooling can be fully automated and used to reliably trigger bit flips across a wide range of real-world systems, highlighting the challenges in weaponizing Rowhammer for practical exploitation. We show that out of 1 006 datasets, automated DRAM addressing function reverse-engineering was only possible on 453 (45 %). Considering unique systems, DRAM addressing function reverse-engineering was possible on 371 (45.1 %) out of 822 systems. We then show that missing DRAM bank addressing functions are the main reason for Rowhammer tools to not run on the systems. Even with found DRAM bank addressing functions on 453 datasets, all our tested tools combined were only able to flip bits in 126 datasets (27.8 % of datasets where DRAM addressing functions were identified). Additionally, three out of five reverse-engineering tools and all Rowhammer tools effectively hammering DDR4 memory require 1 GB huge pages. While we configured 1 GB huge pages on our FlippyRAM system they are typically not available on victim user devices without using elevated privileges.

To answer **RQ2**, we analyze the proportion of datasets that are susceptible to Rowhammer attacks in more detail. This includes examining the success rates of different attack techniques and toolchains, and hardware configurations like CPU vendor and DRAM generation. We show that Rowhammer tools work almost uniquely on DDR3 or DDR4 DRAM, only in rare cases on both. We confirm the finding from Jattke's et al. [20] that almost all current tools work primarily on Intel CPUs. Finally, we compare our results with prior lab-based studies, providing an updated perspective on the real-world prevalence of Rowhammer susceptibility. In total, 93 (11.3 %) of the 822 unique systems in our dataset are susceptible to Rowhammer attacks. Our findings contextualize previous estimates

and clarify the extent to which Rowhammer remains a relevant threat in contemporary hardware.

We conclude that significant progress has been made in understanding and mitigating Rowhammer attacks from the perspective of lab-controlled environments, but the real-world applicability of Rowhammer remains challenging. Surprisingly, this challenge is not due to lacking susceptibility to the Rowhammer effect in general, but rather due to the difficulty of reliably and automatically reverse-engineering DRAM addressing functions and the lack of reliable Rowhammer tools that work across diverse hardware configurations, in particular across different processor microarchitectures[1]. It is not unlikely that the susceptibility of DDR5 memory is underestimated due to the lack of reliable methods both for reverse-engineering DRAM addressing functions and for hammering. Still, a share of about 10 % to 20 % is a relevant mass for threat actors to consider utilizing Rowhammer in real-world attacks, and with improved techniques this may even increase to the range of 20 % to 40 %.

In summary, we make the following contributions:

- We conduct the first large-scale, fully automated study of Rowhammer prevalence on real-world systems in a user study, collecting 1 006 datasets from 822 systems across diverse platforms, DRAM generations, and vendors.
- For this purpose, we develop FlippyRAM, the first end-to-end automated Rowhammer open-source[2] framework. FlippyRAM works across platforms and fully automates all attack steps including reverse-engineering DRAM addressing functions using 5 state-of-the-art tools, and executing 7 state-of-the-art Rowhammer tools, enabling hands-off testing at scale.
- We perform a detailed analysis of the 1 006 data sets we collected, considering CPU and DRAM vendors, DRAM generations, and hammering techniques. On a high level, out of 453 datasets (371 of the 822 unique systems) succeeded in the first stage of reverse-engineering the DRAM addressing functions, 126 datasets (27.8 % of datasets where the first stage worked) exhibited bit flips.
- We identify that the key open challenges for future Rowhammer research are the reliable automation of DRAM address function recovery,

[2] `https://github.com/iisys-sns/flippyram`

given that 50 % of datasets without bit flips failed in the DRAM reverse-engineering stage.

2. Background

In this section, we provide background on DRAM, Rowhammer and related works.

2.1. DRAM

DRAM cells consist of capacitors and transistors organized in rows and columns, which are grouped into banks and ranks on a DIMM [39]. Communication between the CPU's memory controller and the DIMM occurs via channels. On an access to a DRAM cell, the memory controller performs an *activation*: opening the corresponding row and reading the data into a so-called *row buffer*. Further accesses to the same row can then be served from the row buffer. Row activations are destructive, so the DRAM chip needs to write back the content before activating another row. Thus, it is slower to read from different rows on the same bank (*row conflict*) compared to reading from rows on different banks. Hence, addressing functions are designed to distribute contiguous memory across banks, ranks, and channels to avoid conflicts and utilize parallelization. Since capacitors lose charge over time, DRAM needs periodic recharging, e. g., every 64 ms [21, 22, 23].

2.2. Rowhammer

Kim et al. [28] were the first to discover that disturbance errors, i. e., bit flips, in DRAM memory can be deliberately induced by frequently accessing nearby memory rows. Due to these frequent activations, additional charge leakage can occur in physically adjacent rows – a phenomenon known as *Rowhammer*. An adversary can exploit this side effect to induce bit flips in memory. The accessed rows are called *aggressor rows* and the rows prone to bit flips *victim rows*.

In recent years, the research community has developed many sophisticated Rowhammer attacks [11, 39, 48, 40, 10, 8, 41, 19, 20, 27]. These works use different hammering patterns such as single-sided [28], double-sided [11],

many-sided [8, 41], and the half-double Rowhammer pattern [30]. Other approaches do not use static patterns, but randomize patterns using fuzzing [19, 20]. These patterns require locality awareness of the DRAM to co-locate aggressor and victim rows. Since proprietary addressing functions are not documented, previous work reverse-engineered addressing functions utilizing the bank-conflict side channel [39, 50, 8, 14, 20] or performance counters [17]. The bank-conflict side channel exploits the fact that the access time of memory addresses belonging to the same bank is slow because they share a single row buffer, and memory addresses to different banks are fast because they have their own row buffer. Addressing functions are typically linear, in particular on systems where the total number of DRAM components is a power of two. On other systems functions may be non-linear and reverse-engineering these functions remains an open problem. Single-sided Rowhammer [43], One-Location Rowhammer [10], and One-Location RowPress [35, 26] even work without addressing functions.

Besides the focus on Intel x86 CPUs, Rowhammer also works on AMD CPUs [20], and non-x86 architectures, such as Arm [48, 52, 30] and RISC-V [36] processors. Rowhammer can even be exploited via JavaScript [11, 41]. Based on these bit flips the community has developed sophisticated exploits, e. g., using PTE spraying to flip bits in the PFN, resulting in access to arbitrary memory [43]. In Veen et al. [48], an unprivileged Android app uses the Rowhammer effect for privilege escalation to acquire root privileges on stock Android devices. Gruss et al. [10] showed opcode flipping by flipping bits in a predictable way in userspace binaries to bypass isolation mechanisms. Rowhammer attacks have been demonstrated on various DRAM technologies, e. g., DDR3 [28, 4, 40, 51, 11, 10, 45, 46, 24, 32, 47, 27], DDR4 [11, 18, 1, 10, 34, 8, 41, 19, 30, 47, 38, 9, 35, 26, 36, 27, 20], DDR5 [20], LPDDR2 [48, 34], LPDDR3 [48, 52], LPDDR4 [48, 30], and LPDDR4X [8, 19, 30] Recently, Lin et al. [33] demonstrated Rowhammer on GPUs with GDDR6 memory.

A first Rowhammer mitigation was an increased refresh rate. However, as discussed by Kim et al. [28], this does not completely mitigate Rowhammer or brings a significant performance overhead. Another approach was to use Error Correction Code (ECC), shown to be ineffective later by Cojocar et al. [6]. Vendors also utilized the memory controller to detect Rowhammer attacks and refresh potential victim rows, which is called pseudo Target Row Refresh (pTRR).

Starting with DDR4, DRAM vendors started to implement on-chip Target Row Refresh (TRR), which tracks accesses to the DRAM array, detects Rowhammer attacks, and refreshes potential victim Rows. However, multiple publications have shown that the implementation of pattern detection was not sufficient to detect all patterns that triggered bit flips [10, 8, 19, 20]. DDR5 introduced Per-Row Activiation Counting (PRAC) to precisely count activations and, thereby, enable more effective Rowhammer mitigations. However, Jattke et al. [20] identified bit flips on a DDR5 DIMM as well.

There are many further Rowhammer mitigation approaches, e. g., based on cryptographic checks [25], spatial segmentation [5, 31, 49], or counting of activations [37, 2]

2.3. Related Work

Several works drew conclusions on Rowhammer prevalence based on their experiments. Kim et al. [28] tested 129 DRAM modules and found that 110 modules were affected, i. e., 85 %. However, their experiments were FPGA-based and the number of modules that can be attacked successfully from software may be lower. Seaborn and Dullien [43] found 15 out of 29 (52 %) laptops to have bit flips after hammering with their specific Rowhammer test. However, the DRAM addressing functions they used for the double-sided hammering are specific to a certain dual-channel dual-rank DDR3 memory setup which may not have been present in all tested machines, i. e., the number of actually affected devices may be higher. More recently Frigo et al. [8] found 13 out of 42 (31 %) DIMMs to be vulnerable to a software-based attacker using a more sophisticated Rowhammer technique to bypass the TRR Rowhammer mitigation on DDR4. Jattke et al. [19] tested Blacksmith, a fuzzer for Rowhammer access patterns, on 40 DDR4 DIMMs and found all of them to be vulnerable (100 %). He et al. [12] analyze 33 DIMMs and conduct an empirical study of factors that contribute to Rowhammer bit flips. They observed only 6 affected DIMMs (18 %). Other works studied significantly fewer modules, insufficient to draw conclusions about the real-world relevance of Rowhammer. Heckel et al. [16] argue that all works in the domain of Rowhammer suffer from issues that make it difficult to assess the real-world relevance of Rowhammer: (1) All recent works test less than 50 DIMMs with a small set of CPUs and most works even less than 5 DIMMs; (2) experiments are run on lab machines with full control and

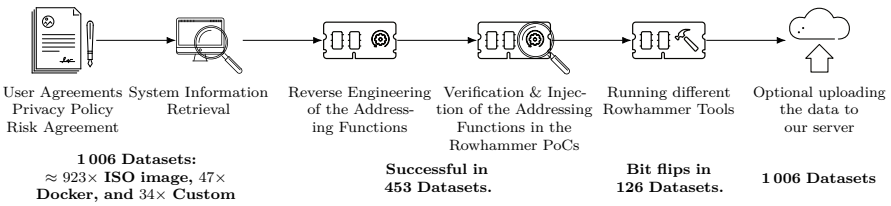


Figure 9.1.: The workflow of our software to automatically test if a system is vulnerable to the Rowhammer effect. It starts right after someone booted our ISO image or ran our docker container.

known hardware-software configurations; (3) results from FPGA setups are not directly transferable to real systems; and Rowhammer tools often require sophisticated tweaks to work on different systems, e. g., setting the reverse-engineered DRAM addressing functions, which is not possible in real-world scenarios without full automation of the process.

3. Methodology

Our central methodology is a large-scale user study to measure Rowhammer susceptibility on a wide range of devices supplied by our study participants. We assume participants to be non-experts and to have little to no knowledge about the topic. Hence, the framework needs to be fully automated and ideally require no user interaction. Through this user study, we can answer how far the available tooling can be used in end-to-end automated Rowhammer testing (**RQ1**) by adapting and deploying state-of-the-art tools in an end-to-end automated fashion. To answer the second research question (**RQ2**), we then measure on all systems in our user study, whether they exhibit bit flips in Rowhammer tests, providing us with a real-world share of systems affected by Rowhammer.

Figure 9.1 depicts the overall workflow of our study, consisting of six steps: The first and the last step are discussed in Section 3.1, i. e., user study design, data collection, and privacy aspects. Steps 2 and 4 are discussed in Section 3.2, covering the information retrieval and the injection of the reverse-engineered DRAM addressing functions into the Rowhammer tools. Step 3, the DRAM address reverse-engineering, involving 5 state-of-the-art tools, is discussed in Section 3.3. Step 5, the execution of up to 7 state-of-the-art Rowhammer tools, is discussed in Section 3.4.

3.1. User Study Design

In a user study, we can achieve far higher sample sizes than prior work. Participants also learn whether their system is vulnerable to our Rowhammer tests. In this subsection, we detail our user study design and discuss associated risks.

Explicit Consent and Data Corruption Risk. Since the Rowhammer effect is a fault attack that could, in principle, cause damage to hardware or data of devices under test, we took precautions and still informed users about potential risks (e. g., possible data corruption or hardware malfunction). We recommended using our bootable USB stick (or ISO image) provided by us, leaving user disks untouched, strictly avoiding user data corruption. Still, the user must explicitly consent, acknowledging the risks for our software to perform any tests. By default, all results are stored locally for the user, and all raw data and log files are compressed as a ZIP archive on the USB drive or stored in a designated output directory when using Docker. This method of data handling ensures that detailed results are preserved for offline analysis and that no information is lost even if multiple tests are run sequentially. The user then either manually visits our website on an internet-connected device to upload the results, or uses the FlippyRAM framework directly to upload the collected results to a central server for further analysis, requiring to connect the system to the internet, again in both cases with explicit consent by the user. If the user does not consent, the results remain on the local media for the user's inspection, but are not uploaded and evaluated as part of our large-scale study. For this reason, we also do not know how many users actually ran the tests, only how many consented to upload their results.

Privacy. Besides risks to the user's data and hardware, we also informed users about privacy implications, specifically what data we collect, how we use it, and how we protect it. FlippyRAM stores each tools' output (e. g., addressing functions recovered), system information retrieved by our scripts, and it logs any observed bit flips while running as well as the execution time and physical memory addresses. The information collected could, in theory, be used to fingerprint and identify systems [42, 7]. Knowing physical addresses of bit flips could be used in actual Rowhammer attacks. Hence, we received no user consent to share these data sets but store them for the purpose of this study in a database on an encrypted device. The framework generates a summary with a brief overview of how many bit flips each tool discovered (if any) for the user. We provide a

contact point to users with concerns or if they want their data deleted at any time.

(Non-)Linkability of Datasets. In general, we ensure that datasets are not linkable to specific users. We do not store any information about the user, e. g., name or email address, in the datasets. The datasets are checked via their hash for uniqueness, i. e., the same dataset cannot be uploaded twice. For each unique dataset, the user gets a unique random string (a token) to later on prove that they participated in the study, e. g., to claim a compensation option such as a T-shirt or voucher lottery. The random string is stored in a separate independent database, without any link to datasets, to track which tokens have been claimed. Hence, participants remain unknown to us if they do not send the token, so we also cannot provide details on the number of individual participants in our study. For participants claiming a compensation option, we have the link between the participant and the token, but there is no link between the token and the dataset and, therefore, no link between the user and the dataset. For data access and deletion requests, we ask the user to provide us with information that uniquely identifies the dataset to delete, e. g., the hash of the dataset or output of *dmidecode* on the system for which we should delete the dataset.

Recruitment. We recruited participants for our study in various ways with the goal of reaching sample sizes that are at least one order of magnitude larger than any prior Rowhammer study, i. e., around 1 000 datasets. The main path was to distribute our bootable ISO image on USB thumb drives. We purchased around 3 000 USB drives and flashed our ISO image onto each USB thumb drive one-by-one manually. We distributed the USB drives to participants during and after a scientific talk at a major security conference with more than 1 500 attendees and after a lightning talk at a Linux conference. We also advertised the study via social media, at different universities, and in our classes. We published the hash of the USB drive for verification that the stick was not modified and contained the correct image. This way, overall, we collected 1 006 of datasets between December 30, 2024 and June 30, 2025 from 822 unique systems.

Recruitment Bias. Our study is biased in the recruitment towards security researchers, students, and other technology-affine users rather than a representative sample of the general population. However, these users are part of the general population and indeed (as we also see in the collected data) use hardware from the same vendors (e. g., Intel and AMD)

and the same DRAM manufacturers (e. g., Micron, Samsung, Hynix) as the general population. No bias was introduced by the software the users ran, as 47 of all datasets we received were generated with the Docker image, and 923 with our bootable ISO image, i. e., completely independent of the operating system and other software the users run. Only 34 datasets we received were generated differently, e. g., via direct script execution. Prior Rowhammer studies were run in a lab setting with hardware acquired by the researchers, which will likely have stronger biases. Instead, we indeed study Rowhammer in a real-world setting with real-world hardware of our study participants, which aligns with the primary goal of our work.

Compensation of study participants. First, users learn whether their system is vulnerable to our Rowhammer tests. Beyond this, we provided USB thumb drives with the FlippyRAM framework, which participants could keep as a small gift. We also offered the first ten participants who ran our test framework at least ten times a T-shirt with the FlippyRAM logo. Participants that ran the framework at least once had the option to enter a lottery to win one of ten 10 € vouchers. For participating students in our classes, we also offered e. g., a small number of extra credits for participating in the study as a small incentive. All incentives are independent of the specific hardware of the participants, and were provided before any results were analyzed, i. e., no bias was introduced by the incentives.

3.2. FlippyRAM Framework

Since our framework must be fully automated, given that we want non-expert users to be able to run it, we designed it as a bootable ISO image or as a docker container[1]. Users receive the bootable ISO image on a USB thumb drive, which they can boot from, requiring no installation or technical expertise. Our framework orchestrates all steps of the Rowhammer test, from initialization to result collection—with minimal to no user interaction. We integrated 7 state-of-the-art Rowhammer tools that were available in time for our study[1] to get a broader assessment of the susceptibility to Rowhammer for each target system as discussed in Section 3.4. We also integrated 5 DRAM reverse-engineering tools to recover the DRAM addressing functions, as discussed in Section 3.3. Hence, our framework initializes the system, e. g., when booted from the ISO image, collects information about the system that is required or beneficial for the tools (e. g., about DRAM modules and CPU), runs tools for addressing

function reverse-engineering, Rowhammer tools, and finally uploads the results to our server if the user agrees. A text-based user interface shows information about the progress and allows users to specify the desired total runtime for the Rowhammer tests.

Information Collection. The system information FlippyRAM collects includes hardware characteristics, e. g., about DRAM modules, CPU models, and CPU features. Thereby, FlippyRAM obtains the number of memory banks, ranks, channels, and DIMMs, according to the system-provided information. We observe that not all systems provide complete information about the DRAM modules, e. g., when *decode-dimms* fails or the data in the Serial Presence Detect (SPD) record of the DIMM is incomplete. In this case, FlippyRAM falls back to a detection of the number of DRAM banks and the DRAM row-hit row-conflict threshold from prior work [14]. If all methods fail, the framework will only use Rowhammer tools that do not require addressing functions and reverse engineering.

Injecting DRAM Addressing Functions into Rowhammer Tools. FlippyRAM supplies each tool with the most likely DRAM addressing functions reverse-engineered on the system. However, we cannot guarantee that the reverse-engineered addressing functions are correct. We also adjust other system-specific settings, such as timing thresholds or memory sizes. Therefore, FlippyRAM patches the source code of the Rowhammer tools or generates input and parameters to use the correct parameters for the specific system. If the reverse-engineering of addressing functions fails, FlippyRAM skips all Rowhammer tools that require addressing functions.

3.3. Reverse-Engineering DRAM Addressing Functions

We integrated 5 state-of-the-art reverse-engineering tools into FlippyRAM to recover DRAM addressing functions. These tools systematically test for a large number of addresses how physical addresses map to the DRAM structure (channels, DIMMs, ranks, banks, rows, columns). The result are addressing functions to translate physical addresses to corresponding DRAM structure information. The tools are proof-of-concepts that researchers validated on one or more systems. However, they may be unreliable or not terminate on real-world configurations, and they do not recover addressing functions on all systems. Hence, we time-slice the

execution of the tools based on the total runtime specified by the user. We run TRRespass RE [8] if the total runtime is at least 3 h, which is the minimum total runtime a user can select. When the runtime is at least 4 h, we additionally run DRAMA [39], if it is at least 5 h, we additionally run Dare [20], if it is at least 6 h, we additionally run AMDRE [14], and if it is at least 7 h, we additionally run DRAMDig [50].

On systems where no 1 GiB pages are available, we had to disable TRRespass RE [8], Dare [20], and DRAMDig [50], that depend on 1 GiB pages. All other tools were run based on the total runtime specified by the user. After running the reverse-engineering tools, we use the verification approach by Heckel et al. [15] to determine which of the reverse-engineered DRAM addressing functions are most likely correct.

FlippyRAM provides the addressing functions to the Rowhammer tools either via command line parameters or as source code and recompilation. If a tool, e. g., Blacksmith [19], fails to initiate due to wrong address functions, FlippyRAM can adjust parameters and rebuild it on the fly and then rerun the tool: For instance, trying other address functions (if more functions than required to address the banks were identified) or a different row conflict threshold. This adaptive mechanism increases the chances to find a working configuration without manual interaction. Additionally, for some tools, we improved the logging to provide better parseable output. However, we did not change any of the functionality of the tools.

3.4. Rowhammer Prevalence Test using Rowhammer Tools

FlippyRAM executes Rowhammer tools to test for Rowhammer susceptibility. The framework currently includes Blacksmith [19], TRRespass [8], FlipFloyd [10], RowPress [35], RowhammerJS (native code, flush-based proof-of-concept, double-sided hammering) [11], HammerTool [13], and Rowhammer-Test (single-sided hammering) [44]. FlippyRAM monitors each tool's execution and if a tool finishes early or throws an error, FlippyRAM reallocates the remaining time to other tools or reconfigures the tool.

Some Rowhammer tools require 1 GiB hugepage, which requires CPU support and a sufficient amount of memory, e. g., more than 4 GiB of DRAM. On systems where no 1 GiB pages are available, we had to disable Blacksmith [19], TRRespass [8], and RowPress [35], that depend on 1 GiB

pages. Most tools also require DRAM bank addressing functions. When no DRAM addressing functions were identified, Blacksmith [19], TRRespass [8], and RowPress [35], RowhammerJS [11], and HammerTool [13] cannot run and are disabled. After the reverse-engineering of the DRAM addressing functions is done, the remaining runtime is split between all Rowhammer tools that are executed (e. g., not disabled).

3.5. Analysis of the Datasets

It should be noted that a single system can be tested multiple times, so we distinguish between unique systems (822) and datasets (1 006). We take the 1 006 datasets and check whether a tool did not run in a dataset, which can have multiple reasons as described below. We then analyze the number of datasets on which specific Rowhammer tools found bit flips and the number of bit flips found in total by the different tools.

We also analyze the susceptibility depending on the Runtime of FlippyRAM, the vendors of the CPUs used in the systems, and the DDR generation used in the systems. In both cases, we distinguish between systems that are affected by Rowhammer, systems that are not affected, and systems on which the reverse-engineering of DRAM addressing function failed. It should be noted that two of our Rowhammer tools, namely FlipFloyd and Rowhammer-Test, do not require DRAM bank addressing functions. Therefore, we count a system as *affected* when at least one tool identified bit flips, even if the addressing functions were not identified on that system. Otherwise, we count a system as having no addressing functions when no addressing functions were found or as not affected when addressing functions were found but no bit flips.

We then analyze the susceptibility of single DIMMs based on the vendor of the DRAM chips and the frequency of the DIMMs. For these measurements, we only consider datasets on which DRAM bank addressing functions were identified. Similar to the previously described case, we count a system as *affected* when a bit flip happened, even when no addressing functions were identified. However, we count a system only as *not affected* when DRAM addressing functions were not identified and no bit flips occurred. For this evaluation, we only consider these datasets and skip all datasets with failed DRAM addressing function reverse-engineering.

We are unable to map bit flips monitored on a system to unique DIMMs in that system. Therefore, we consider all DIMMs of a system equally

affected, even though only one of them might actually be affected. When the total number of bit flips on a system is bigger than or equal to the number of DIMMs on that system, all DIMMs count as affected (e. g., 4 DIMMs in a system count as affected when at least 4 bit flips occurred). Otherwise, the number of bit flips is divided by the number of DIMMs (e. g., on a system with 4 DIMMs and 1 bit flip each DIMM counts as $\frac{1}{4}$ affected and $\frac{3}{4}$ not affected).

We evaluate the time until the first bit flip was detected by the different Rowhammer tools. Therefore, we only consider systems that were affected by the tool, e. g., the respective tool found at least one bit flip. We then measure the time until the first bit flip occurred. When the tool was started multiple times, we measure from each start and take the minimum across multiple runs. For measurements not directly related to the tools, e. g., time to the first bit flip depending on the CPU vendor, we consider only the fastest tool on that system.

4. Technical Results of Real-World Rowhammer Feasibility User Study

In this section, we address our first research question **RQ1**, on the feasibility of Rowhammer end-to-end automation for real-world systems. As described in Section 3, our framework performs multiple stages to test a system for susceptibility to Rowhammer, which may or may not succeed on a system. We analyze on how many systems the different steps and tools were executed successfully, and on how many systems they failed for various reasons.

In general, we distinguish between 8 different tool states:

Ended. The tool completed execution. This is not an indication of success or failure i. e., whether valid addressing functions were identified or whether bit flips occurred.

Disabled. Tool skipped to give more time to other tools.

Runtime. The tool was skipped due to total runtime limits to give more time to other tools.

Hugepage. Tool skipped due to unavailability of 1 GiB pages.

AFn RE. These tools were not run due to unsuccessful reverse-engineering of addressing functions.

Failed. The tool failed to run, e. g., due to a crash or timeout (each tool is started with a timeout to keep within the runtime specified by the user). We assume that most crashes happened due to implementation errors related to incomplete or missing error handling.

Build Failed. Some tools require dynamic source-code adjustments and recompilation. This step failed sometimes because the addressing functions did not match requirements of the tools, e. g., Blacksmith requires invertible functions.

Other. None of the above states applied, e. g., due to missing log files or corrupted log files.

4.1. Address Function Reverse-Engineering

We analyze the execution of different DRAM addressing reverse-engineering tools, as summarized in Figure 9.2, based on the previously defined tool states. We start the tools without manual interaction, so the parameters required by the tools are measured automatically. Therefore, they might not be precise due to a failed measurement and the lack of manual interaction to detect these cases. Therefore, tools with more preconditions (e. g., values that have to be adjusted specifically for the system) tend to have a higher rate of failure.

DRAMA reverse-engineering tool ended on the most datasets, namely 485 (48.2 %), followed by AMDRE which ended on 481 (47.8 %) datasets. TRRespass RE ended on 416 (41.4 %) datasets and Dare ended on 385 (38.3 %) datasets. Lastly, DRAMDig ended on only 122 (12.1 %) datasets.

Considering the reasons why the tools did not end, AMDRE failed on 376 systems, either by exceeding the specified timeout or by crashing. AMDRE was not started on 136 systems due to a lower runtime specified by the user. For Dare, there are three reasons for failure: On 169 datasets it failed, on 146 datasets no 1 GiB hugepage could be allocated, and on 130 datasets it was disabled due to total runtime limits. Similarly for DRAMDig: On 537 datasets it failed, on 141 datasets there was no 1 GiB page support, and on 154 datasets it was disabled due to total runtime limits.

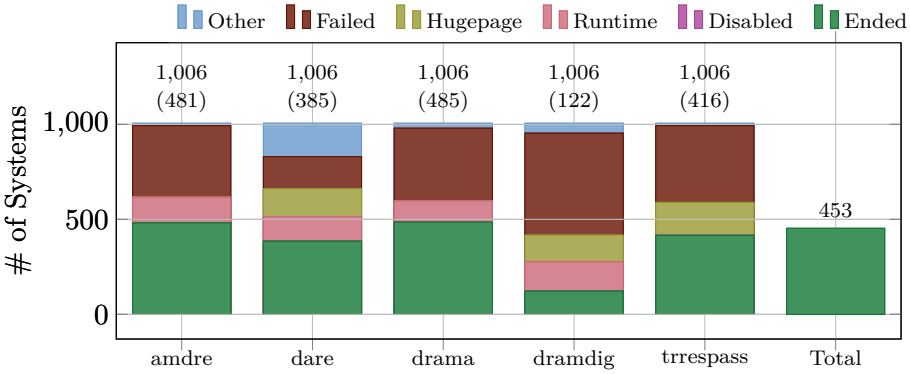


Figure 9.2.: Number of systems with specific states of different Addressing Function Reverse-Engineering tools. The number above the bar depicts the absolute number of datasets. The number in braces shows the number of datasets on which the tool had the *Ended* state. The *Total* bar depicts the number of datasets in which valid addressing functions were identified (even though a reverse-engineering tool has the state *Ended*, it does not mean that it returned the number of functions needed for the system).

DRAMA was disabled on 112 datasets due to total runtime limits and failed on 383 datasets. TRRespass RE was not disabled due to the runtime specified by the user. However, it failed on 404 datasets and was disabled on 173 datasets due to a lack availability of a 1 GiB hugepage. This is consistent with the fact that 146 datasets, according to the `cpuid` information, do not support 1 GiB pages and some systems may have too little memory for Linux to successfully allocate a 1 GiB page. It also highlights that more systems may succeed in this attack stage if the tools would not require 1 GiB pages.

Key Insight 1: In the majority of cases, the reverse-engineering tools failed to find DRAM addressing functions, either crashing or exceeding time limits, highlighting a gap that requires further research.

4.2. Rowhammer Tools

We also analyze the execution of the different Rowhammer tools based on the previously defined tool states, as summarized in Figure 9.3. The only tools that ran on over 50 % of the tested systems are FlipFloyd and Rowhammer-Test, which do not require DRAM addressing functions. For

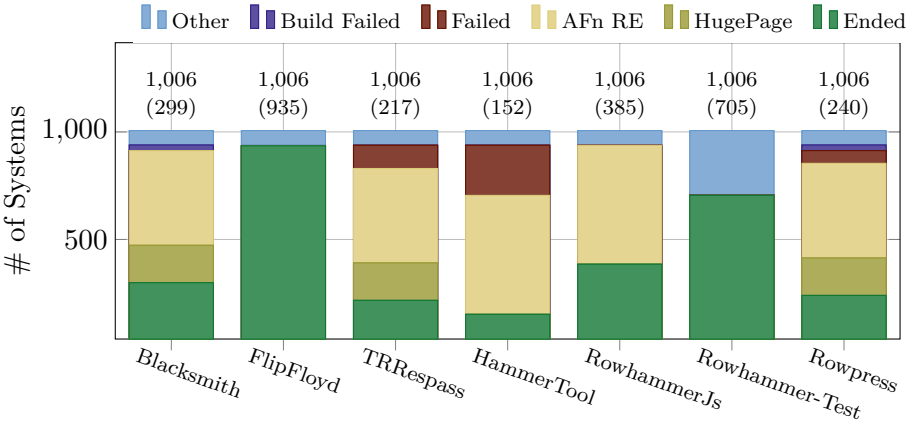


Figure 9.3.: Number of datasets with specific states of different Rowhammer tools. The number above the bar depicts the absolute number of datasets. The number in braces shows the number of datasets on which the tool had the *Ended* state, e. g., ran successfully.

the other tools, the main reason not to run were missing DRAM addressing functions, e. g., reverse-engineering them in the previous step failed. Some tools were skipped due to the lack of 1 GiB hugepages, and some tools failed to run, probably due to incorrect parameter settings on the system. For Blacksmith and RowPress, building failed on some systems.

Key Insight 2: Many Rowhammer tools do not work without DRAM bank addressing functions or 1 GiB hugepages. Providing fall-back mechanisms may result in a drastic increase in compatibility.

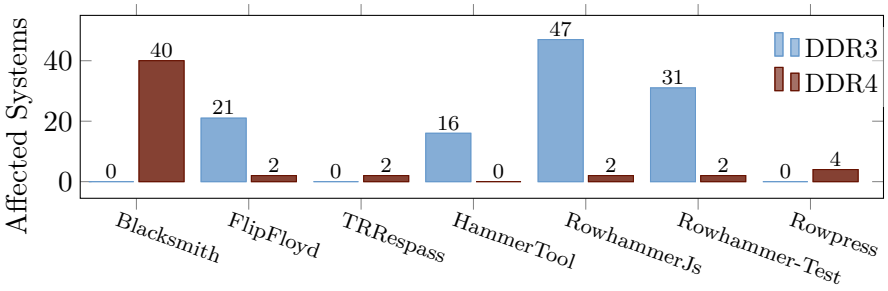
4.3. Success Rate of Rowhammer Tools

We analyze the Rowhammer susceptibility of different DRAM generations depending on the different tools we ran. Since we only observed bit flips on DDR3 and DDR4 systems, we focus on these memories in this subsection. We show the total number of affected systems (over both DDR generations) in Table 9.1 and Figure 9.4a. Additionally, we evaluate the number of bit flips triggered by different tools (see Figure 9.4b).

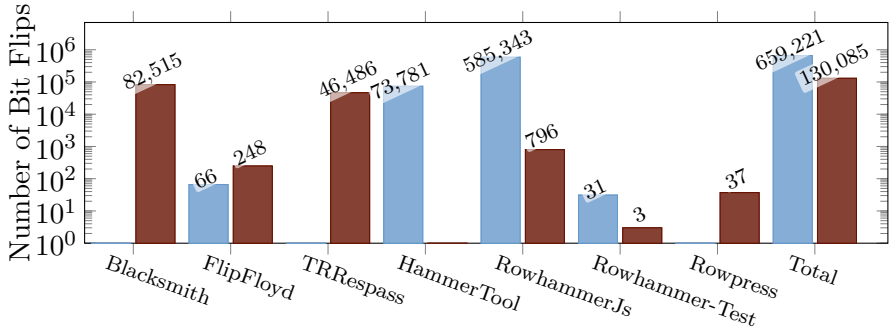
FlipFloyd [10], RowhammerJS [11], and Rowhammer-Test [44] induced bit flips predominantly in DDR3 DRAM but also in a few datasets with DDR4 DRAM. Of these, RowhammerJS yielded the best results regarding the number of affected datasets (49) and the number of bit flips

Table 9.1.: The success rates of the different Rowhammer tools.

Tool	# Ended Systems	# Affected Systems		min to 1. bit flip			
				Avg	σ	Min	Max
Blacksmith	299	40	13.4 %	62	26	37	88
FlipFloyd	935	23	2.46 %	115	186	0	758
TRRespass	217	2	0.922 %	0	0	0	0
HammerTool	152	16	10.5 %	12	14	1	38
RowhammerJs	385	49	12.7 %	28	37	0	156
Rowhammer-Test	705	33	4.68 %	93	163	0	617
Rowpress	240	4	1.67 %	21	12	9	38



(a) Number of datasets affected by different Rowhammer tools



(b) Number of bit flips triggered by different Rowhammer tools. Bars with a number of 0 are not shown in the graph due to the log y scale.

Figure 9.4.: Results grouped by Rowhammer Tool. The blue bars show the results for DDR3 and the red bars for DDR4. None of the tested DDR2 and DDR5 systems were affected, so they are omitted in this graph.

(586 139). Regarding the total number of affected datasets, it is followed by Rowhammer-Test (33 datasets and 34 bit flips), which is followed by FlipFloyd (23 datasets and 314 bit flips). The number of bit flips identified by RowhammerJs on DDR4 (796) is very low compared to the number on DDR3, which can be explained with the DDR4 mitigations: Normally, TRR should detect and mitigate double-sided Rowhammer attacks. However, it seems that TRR was unsuccessful on 2 datasets where multiple tools found bit flips despite TRR being active.

Blacksmith [19] (82 515 bit flips in 40 datasets), TRRespass [8] (46 486 bit flips in 2 datasets), and RowPress [35] (37 bit flips in 4 datasets) only induced bit flips on systems with DDR4. Compared to other tools that induced bit flips on systems with DDR4 and that required DRAM addressing functions, the number of bit flips identified by Rowpress is very low. HammerTool [13] only induced bit flips on systems with DDR3, specifically 73 781 bit flips on 16 datasets. Overall, the tools induced 789 306 bit flips in 126 out of 1 006 datasets.

Key Insight 3: 126 (12.5 %) out of 1 006 datasets are vulnerable to fully-automated Rowhammer attacks, based on our analysis.

These results show that DDR3 and DDR4 DRAM require very different hammer patterns to induce bit flips. On DDR3 DRAM, fast double-sided Rowhammer, as performed by RowhammerJS [11], is most effective because the DRAM does not contain any mitigations against this simple hammer pattern. DDR4 DRAM, on the other hand, nowadays contains Rowhammer mitigations. Therefore, complex access patterns, like performed by Blacksmith [19] are required to circumvent the mitigations. However, due to these complex access patterns, the minimum required activation count to induce bit flips on less vulnerable DDR3 DRAM is not reached.

Key Insight 4: For DDR3, simple patterns and hammering as fast as possible, implemented by RowhammerJS, were the most effective strategies in terms of the number of bit flips. Since most DDR4 DIMMs have TRR, pattern fuzzing strategies like Blacksmith were the most effective ones and found most bit flips.

Table 9.1 shows the average, minimum and maximum time to first bit flip. Considering the average time until the first bit flip, the tools rank by the following order starting with the fastest tool: TRRespass, HammerTool, RowPress, RowhammerJS, Blacksmith, Rowhammer-Test, FlipFloyd.

The two tools that do not require DRAM bank addressing functions (Rowhammer-Test and FlipFloyd) took the most time until the first bit flip occurs. Also, they were exclusively executed on systems where the reverse-engineering of DRAM addressing functions failed (see Section 3.4). Thus, the runtime on these systems was only split between both tools leading to a higher runtime compared to other systems and a higher chance to find bit flips with this tool. In general, both tools select random addresses to hammer, without using the DRAM bank addressing functions. For Rowhammer-Test the probability of two addresses being in the same DRAM bank is only $\frac{1}{n_{\text{banks}}}$ on a system with n_{banks} DRAM banks, leading to a correspondingly lower chance and longer time to find bit flips. For FlipFloyd, the authors report a higher time until bit flips occur than for other Rowhammer implementations (3.3 times slower than comparable hammering methods) [10].

Blacksmith performs a randomized fuzzing of several parameters to identify pattern that yield bit flips. Because these patterns have to reach the required activation count of the system and have to bypass the TRR implementation of the DIMM at the same time, finding such patterns is not trivial. This explains the higher runtime compared to tools with hard-coded patterns, like HammerTool, RowhammerJS, and Rowpress. Only tools that do not require DRAM bank addressing functions take longer than Blacksmith.

RowhammerJS, RowPress, and HammerTool use static Rowhammer access patterns computed from addressing functions and physical addresses. The average time until the first bit flip is in the same order of magnitude for RowhammerJS (μ =27.8 min, σ =37.2 min), RowPress (μ =21.3 min, σ =12.5 min) and HammerTool (μ =12.4 min, σ =14.4 min). The fastest tool was TRRespass, finding the first bit instantly within the first second, on both datasets where it found bit flips. In total, TRRespass found 46 486 bit flips on these 2 datasets, showing that they are highly susceptible.

Key Insight 5: The minimum time the first bit flip occurred ranges from 0 min to 115 min on average, depending on the system configuration, which is a practical attack time frame on real-world systems.

4.4. Rowhammer Susceptibility by FlippyRAM Runtime

We analyze the system's susceptibility based on the total runtime of FlippyRAM. The DRAM bank addressing reverse-engineering tools are

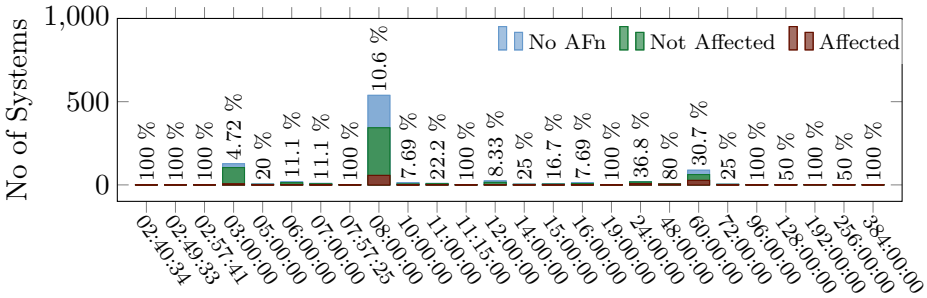


Figure 9.5.: Number of datasets affected by Rowhammer grouped by Runtime. The bars of affected and not affected datasets stack to the total number of datasets with the runtime. The percentage depicted above the bars it the percentage of affected datasets with that runtime.

run based on the total runtime as described in Section 3.3. Therefore, systems with more runtime are more likely to have correct DRAM bank addressing functions since more reverse-engineering tools were executed. It should also be noted that the number of systems with a specific runtime differs. For example, the default runtime was set to 8 h, so the runtime for most dataset (538 of 1 006) is 8 h. Figure 9.5 shows the number of datasets affected by Rowhammer grouped by the runtime selected by the user.

Regarding the number of datasets with a specific runtime, there are three characteristic peaks: One at 3 h, one at 8 h and one at 60 h. We set the minimum runtime of our toolset to 3 h, so people that wanted to participate but not run the experiment longer than strictly required, selected a runtime of 3 h. There is another peak at 8 h, which is the default runtime. The third bigger peak is at 60 h, which is the time for which we tested the systems in the computer rooms of our institution. On 3 systems our tool ran shorter than 3 h, this happens when the last scheduled Rowhammer tool finishes early.

We do not consider the runtimes where 100 % are affected, i. e., only a single dataset had that runtime. When skipping these cases, we see a trend with increasing runtime: We see 4.72 % affected with a runtime of 3 h, 10.6 % affected with a runtime of 8 h, and 30.7 % affected with a runtime of 60 h. The difference between the runtime of 3 h and 8 h can be explained by the disabled addressing function reverse-engineering tools (cf. Section 3.3). However, the trend also continues for runtimes of more than 7 h, where all reverse-engineering tools are enabled. Based on these results,

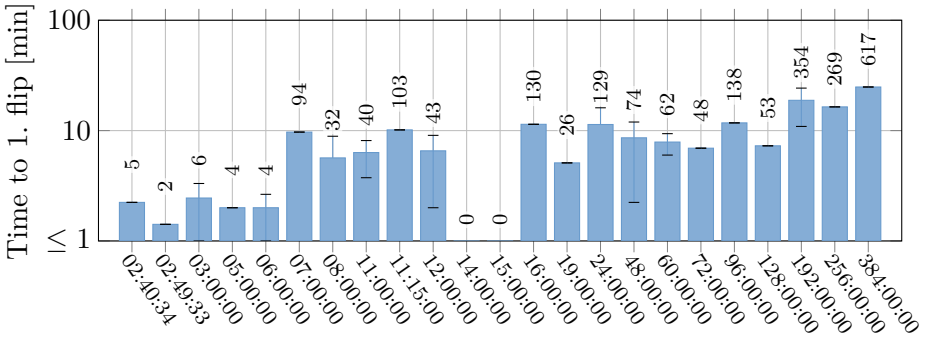


Figure 9.6.: Time until the first bit flip grouped by total runtime. The bars show the average time, the error bars the standard deviation of the fastest tool on each dataset. Low values rounded to 1 minute.

we conclude that longer testing leads to more systems being correctly identified as susceptible to Rowhammer.

Figure 9.6 shows the time until the first bit flip by runtime. In general, an increasing trend can be seen: The higher the runtime of our framework, the higher the average time until the first bit flip. This can be explained with the previous insight: A longer runtime leads to system being detected as affected that would not have been detected with a shorter runtime. Therefore, the bit flips that are detected occur later, which also increases the time until the first bit flips are detected.

Key Insight 6: On systems that are barely susceptible, it can take very long to find the first bit flips, on one system it took 617 min. Longer testing times lead to more accurate detection of systems as affected and higher times until the first bit flip is detected.

5. Detailed Analysis of Rowhammer Susceptibility by System Characteristics

In this section, we address our second research question **RQ2**, to analyze in more detail which systems are vulnerable to Rowhammer. We analyze susceptibility depending on e. g., DRAM generation, DRAM speed, and CPU vendor.

Table 9.2.: A summary of the results on hardware susceptibility by DRAM generation and CPU vendor.

DRAM	CPU	# Systems	# Affected Systems		min to 1. bit flip Avg	σ	Min	Max
DDR3	Intel	286	80	28 %	62	113	0	617
DDR3	AMD	15	2	13.3 %	42	42	0	85
DDR4	Intel	365	43	11.8 %	55	42	9	130
DDR4	AMD	137	1	0.73 %	0	0	0	0

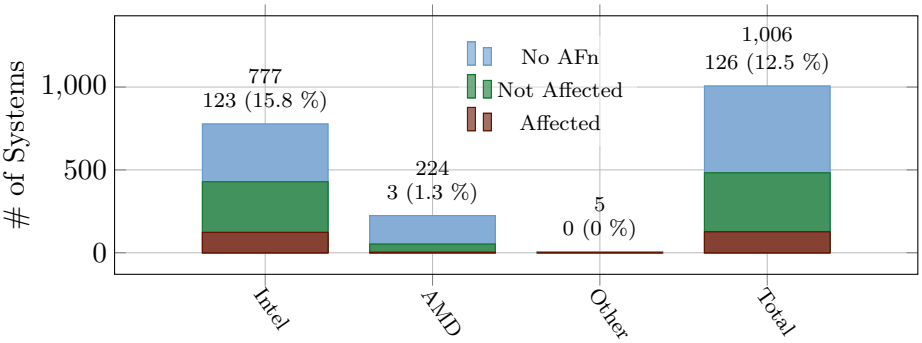


Figure 9.7.: Datasets affected by Rowhammer grouped by CPU Vendor. The bars of affected, not affected, and AFn failed datasets stack to the total number of datasets with the CPU vendor. The numbers depicted above the bars are the absolute number of datasets on the top and the percentage of affected datasets on the bottom. A dataset counts as *affected* when at least one tool identified at least one bit flip. It counts as *No AFn* when no bit flips were identified and no addressing functions were found on the dataset. It counts as *Not Affected* when addressing functions were available, but no bit flips were found.

5.1. Susceptibility by CPU Vendor

We analyze the susceptibility of systems to Rowhammer depending on the CPU vendor. While Rowhammer is a bug in the DRAM, the CPU plays a huge part in the attack. For example, Jattke et al. [20] recently showed that due small differences between Intel and AMD CPUs, changes are required for Rowhammer attacks working on Intel, to also work on AMD. The majority of systems in our dataset use CPUs from Intel and AMD, with only 5 datasets with other CPU vendors. Figure 9.7 provides an overview of these results.

Out of 1 006 tested datasets in total, 777 have Intel CPUs, of which 123 (15.8 % of total datasets) are affected. On 349 datasets with Intel CPUs, no bit flips occurred and no addressing functions could be identified. Of the 224 datasets with AMD CPUs, 3 (1.34 % of total datasets) are affected and on 172 no bit flips occurred and no addressing functions could be identified. Note that the ZenHammer Rowhammer fuzzer by Jattke et al. [20], which is the first tool yielding good results on systems with AMD CPUs, was not part of our framework, as described in Section 3.2. Finally, none of the 5 CPUs from other vendors are affected and on 3 of them, no addressing functions were identified.

Based on the CPU vendor, 15.8 % of Intel Systems and 1.34 % of datasets with AMD CPUs are affected. We assume that the higher susceptibility of Intel CPUs is due to the fact that Rowhammer research mainly focused on Intel-based systems in the last decade [43, 39]. Just recently, the focus of Rowhammer research shifted slightly towards AMD [14, 20].

Even though our bootable image only works on systems with the x86-64 architecture, some participants got the setup running on other architectures in a few cases, e. g., using the Docker container or manually downloaded and executed the scripts from GitHub. On datasets with neither an AMD nor an Intel CPU (labelled as *Other* in Figure 9.7), the CPU model was either not detected (which happened on 2 datasets), or it was one of the following: *VIA Nano U3100* (x86-64), *Cortex-A7* (Arm), or a *sifive u74-mc* (RISC-V).

The times until the first bit flip was detected are shown in Table 9.2. On Intel systems, the average time to the first bit flip is 60.8 min (σ =108 min). On AMD systems it is 28.4 min (σ =39.8 min), however, the small sample size of only 3 systems limits the significance of this result. The longest time to a bit flip was 617 min (roughly 10 h) on an Intel system.

Key Insight 7: Most tools work more reliably on Intel-based systems. Hence, the number of affected AMD systems may be higher than our results suggest.

5.2. Susceptibility by DRAM Generation

We analyze the Rowhammer susceptibility depending on the DRAM generation. The majority of systems in our dataset use DDR3, DDR4, and DDR5 DRAM, with a few systems containing low-power (LPDDR)

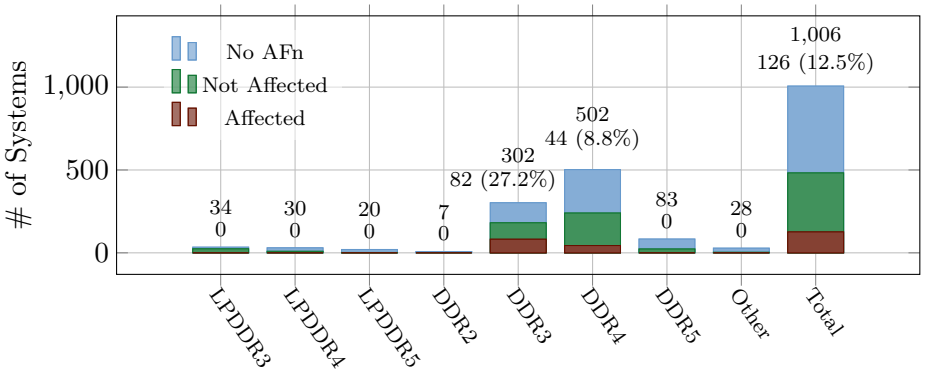


Figure 9.8.: Datasets affected by Rowhammer grouped by DRAM Generation. The bars of affected, not affected, and No AFn datasets stack to the total number of datasets with the DRAM Generation. The numbers depicted above the bars are the absolute number of datasets on the top and the number and percentage of affected datasets on the bottom.

and DDR2 DRAM. In the category *Other* belong memory types that `dmidecode` could not identify, e. g., when the field `Type` was `Other` or `<OUT OF SPEC>`. Figure 9.8 provides an overview of the results.

Out of the 302 datasets with DDR3 DRAM, 82 (27.2 %) are affected. It takes on average 60.4 min ($\sigma = 111$ min) to get the first bit flip. Out of the 502 datasets with DDR4 DRAM, 44 (8.76 %) are affected However, a total of 224 datasets use an AMD CPU and numbers with AMD-adjusted tools may be significantly higher (cf. Section 5.1). Focusing only on the 388 DDR4 with Intel CPUs, 43 (11.1 %) are affected. It takes on average 39.9 min ($\sigma = 43.4$ min) to flip the first bit on datasets with DDR4 DRAM, and the fastest time was 0 min. Other DRAM generations were not affected by the tested tools.

Initially, Rowhammer was discovered on DDR3 [28], so Rowhammer research has been conducted on DDR3 from the beginning. Following the original publication, vendors began implementing hardware-mitigations with DDR4. Multiple approaches bypass these proprietary mitigations implemented by the vendors [10, 8, 19]. However, because the details of the implementation are not known, the ability of these approaches to trigger bit flips strongly depends on the mitigations. Therefore, the percentage of affected datasets with DDR4 (8.76 %) is lower than that of DDR3 (27.2 %). The fraction of affected datasets with DDR4 is still

Table 9.3.: The results on hardware susceptibility by DRAM vendor.

DRAM	DIMMs	Affected		min to 1. bit flip			
				Avg	σ	Min	Max
Samsung	321	102	31.7 %	102	160	0	617
Hynix	142	40.6	28.6 %	56	50	0	176
Micron	130	3.12	2.4 %	2	4	0	10
Other	179	40.8	22.8 %	31	44	0	188

lower than for DDR3 when only considering Intel-based DDR4 datasets, for which 11.1 % are affected. Jattke et al. [20] were the first to find bit flips on a system with DDR5 DRAM. However, we were unable to identify bit flips on any of the 83 DDR5 systems our participants tested. It should be considered that we did not add Zenhammer [20], the only tool that has detected bit flips on DDR5, to our test suite, as discussed in Section 3.2.

Key Insight 8: We observe the most bit flips on systems with DDR3, followed by systems with DDR4 DRAM. Other systems may have no bit flips as the available tooling is not suitable for these without adjustments.[1]

On average, it takes longer to find bit flips on DDR3 than on DDR4 although more DDR3 systems are affected. We hypothesize that this is primarily related to the runtime of the experiments: The datasets with runtimes of 128 h and above were measured on systems with DDR3 DRAM.

5.3. Susceptibility by DRAM Vendor

We analyze the susceptibility of datasets to Rowhammer depending on the DRAM vendor. When a dataset is affected by Rowhammer, we consider it for further evaluation. When it is not affected, we consider it only when the reverse-engineering of DRAM addressing functions was successful on the dataset and when the dataset has an Intel CPU, as we evaluate *affected DIMMs* and not systems where no addressing functions were found or where no bit flips occurred due to an AMD CPU not supported by the tools used in our framework.

The majority of datasets has DIMMs from Samsung, Hynix, or Micron. However, there are also many datasets with DRAM labelled to be other

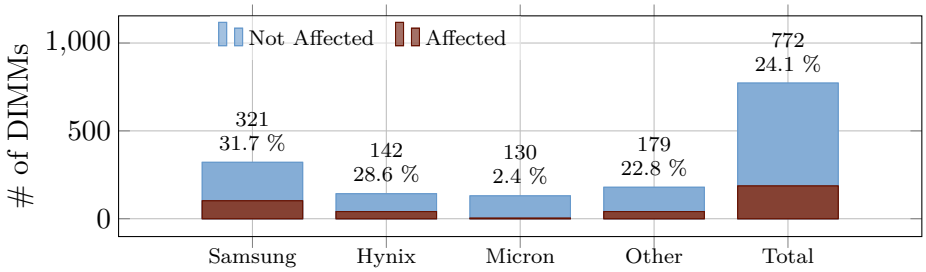


Figure 9.9.: DIMMs affected by Rowhammer grouped by Vendor. The bars of affected, and not affected systems stack to the total number of DIMMs from that Vendor. The numbers depicted above the bars are the absolute number of DIMMs on the top and the percentage of affected DIMMs on the bottom.

vendors (e. g., resellers like Corsair, A-DATA, G-SKILL, etc.), or where the Serial Presence Detect (SPD) record of the DIMMs did not state any vendor. Therefore, we group by *Samsung*, *Hynix*, *Micron*, and *Other*

Systems can have multiple DIMMs from different vendors, which is the case for 43 datasets. Because we do not know which DIMMs of a system are affected and which are not when a bit flip is identified on that system, we consider all DIMMs of the system to be affected. For example, a system with two DIMMs from Samsung and two DIMMs from Hynix counts as 2× Samsung and 2× Hynix. We count all DIMMs of a dataset as partly affected when the number of bit flips is lower as the number of DIMMs as described in Section 3.5. Figure 9.9 provides an overview of these results.

Our results (cf. Table 9.3) show that 102 (31.7 %) DIMMs from Samsung are affected, which is the highest percentage. The susceptibility of DIMMs from Hynix (28.6 %) and Micron (2.4 %) are lower. For DIMMs that could not be mapped to a vendor, 22.8 % are affected.

The first bit flip on Samsung DIMMs occurs on average after 102 min ($\sigma = 160$ min), the longest time of all vendors. It is followed by DIMMs from Hynix with an average time of 56 min ($\sigma = 50.4$ min) until the first bit flip. For DIMMs from Micron, the first bit flip occurred after 1.95 min ($\sigma = 3.79$ min). DIMMs where the vendor could not be mapped show the first flip on average after 31.4 min ($\sigma = 44.4$ min).

We hypothesize that this is related to the distribution of affected DIMMs over the different DRAM vendors: While the fraction of affected DIMMs is similar for DRAM from Samsung and Hynix, the absolute number of

Table 9.4.: Summary of hardware susceptibility by DRAM frequency.

DRAM	DIMMs	Affected		min to 1. bit flip			
				Avg	σ	Min	Max
1333	96	33.4	34.8 %	17	27	0	103
1600	190	60	31.6 %	100	142	0	617
2133	100	20	20 %	32	52	0	130
2400	87	5	5.75 %	38	0	38	38
2667	137	59	43.1 %	52	31	9	88
3600	8	0.25	3.12 %	98	0	98	98
Other	103	8.75	8.5 %	57	56	0	147

affected DIMMs is significantly higher for Samsung (102) than for other vendors (40.8), Hynix (40.6), and Micron (3.12).

We hypothesize that DRAM from Micron is less affected due to differences in the proprietary TRR implementation: We assume that either TRR in Micron DIMMs works better at detecting current Rowhammer attack implementations, or that current Rowhammer attack implementations are (possibly inadvertently, due to proprietary implementations) tailored to bypass TRR implementations of other vendors[3].

Key Insight 9: DRAM from Samsung, Hynix, and third-party resellers is similarly affected by Rowhammer. We found bit flips in only 2.4 % if Micron DIMMs. This contrasts prior work that did not find such a stark difference between the three manufacturers [28, 19, 35].

5.4. Susceptibility by DRAM Frequency

In this section, we analyze the susceptibility of systems to Rowhammer depending on the frequency of the DIMMs. Most systems in our datasets use the DRAM frequencies shown in Table 9.4 and Figure 9.10. We only consider DIMMs susceptible to Rowhammer where DRAM addressing function reverse-engineering worked and the systems hat Intel CPUs. Because we do not know which DIMMs of a system are affected and which

[3]The authors of the TRRespass [8] and Blacksmith [19] papers told us that vendor B is Micron in their experimental evaluation. They stated that Micron was arguably the most difficult vendor for DIMMs to trigger bit flips on. This note was not included in the camera-ready version of the paper.

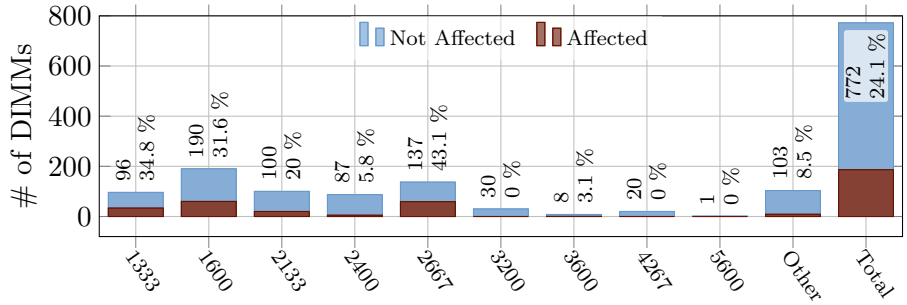


Figure 9.10.: Susceptibility of DIMMs to Rowhammer grouped by Speed (Frequency). The bars of affected and not affected DIMMs stack to the total number of DIMMs with the DRAM Speed (Frequency). The numbers depicted above the bars are the absolute number of DIMMs on the left side and the percentage of affected DIMMs on the right side.

not when a bit flip was identified on that system, we consider all DIMMs of the system at least partial affected as described in Section 3.5. We count the number of DIMMs in the system and group them by the speed (DRAM frequency).

In general, the susceptibility of DIMMs decreases with increasing DRAM frequency (cf. Figure 9.10), which is likely related to different DRAM generations as discussed below. There are two exceptions: One for 2 667 MT/s with 5.75 % and one for 3 600 MT/s with 3.12 %, which we discuss below. The affected systems with 1 333 MT/s to 1 600 MT/s are DDR3, and with 2 400 MT/s to 3 600 MT/s are DDR4 based on the DRAM generation reported by *dmidecode*. For the datasets with 2 133 MT/s, 4 are DDR3 and 70 are DDR4, so the majority of these DIMMs is DDR4.

The higher percentages of 43.1 % for DIMMs with a frequency of 2 667 MT/s can be explained with the systems that had a runtime of 60 h. As described above, we ran the test on systems in the computer lab rooms of our institution, which are all DDR4 and have DRAM with a frequency of 2 667 MT/s. Thereby, we identified multiple systems with a total of 54 DIMMs with a runtime of 60 h to be affected by Rowhammer. Due to this, 54 of the 59 DDR4 systems affected by Rowhammer have a frequency of 2 667 MT/s had a runtime of 60 h. A higher runtime increases the number of systems susceptible to Rowhammer as discussed above.

Regarding the DIMMs with a frequency of 3 600 MT/s, where 3.12 % are affected by Rowhammer, there is one system with 4 DIMMs that is

affected by Rowhammer. This system had a runtime of 24 h and FlipFloyd identified a single bit flip. Since we saw less bit flips than DIMMs in the system, this is counted as 0.25 affected and 7.75 not affected DIMMs.

For the average time until the first bit flip occurred, a trend can be seen with increasing DRAM frequency (except at 1 600 MT/s, see next paragraph): At the lowest, DIMMs with 1 333 MT/s have an average time of 16.6 min ($\sigma = 27.3$ min) until the first bit flip occurs. The highest frequency DIMMs where multiple are affected are DIMMs with 2 667 MT/s. There it takes on average 51.7 min ($\sigma = 31.4$ min) until the first bit flip occurs. For other DIMMs, the average time until the first bit flip occurs is 56.7 min ($\sigma = 56.3$ min).

Only the DIMMs with 1 600 MT/s do not follow this trend with an average time of 100 min ($\sigma = 142$ min) until the first bit flip occurs. The reason for this is that some datasets with these DIMMs had really high runtimes of, e. g., 192 h, 256 h, and 384 h. As discussed in Section 4.4, a higher runtime leads to increased times until the first bit flip occurs because they enable finding bit flips after the tools stopped running in other datasets. These high runtimes lead to a time until the first bit flip is detected of up to 617 min which increases the average value compared to the DIMMs with other, similar frequencies like 1 333 MT/s to 2 133 MT/s.

Key Insight 10: We find a decrease in the susceptibility to Rowhammer and a slight increase in the time until the first bit flip with higher DRAM transfer rate.

6. Limitations

In this section, we discuss limitations of our results.

Most addressing function revese-engineering tools failed on many systems as described in Section 4.1. Even the most successful tools only ran on less than 50 % of the datasets, and even this does not imply correct DRAM addressing functions were identified but only that the tool ran. Additionally 3 of 5 tools need support of 1 GiB hugepages, which was not available on some systems. Blacksmith identified bit flips on 40 DDR4 datasets out of a total of 44 datasets with DDR4 susceptible to Rowhammer. However, it was also started only on 299 out of 1 006 datasets, which is approximately one third. The results for DDR3 look similar: RowhammerJs, the tool that worked best on DDR3, identified bit flips on 49 datasets, of which 47 are

DDR3. In total, 82 datasets with DDR3 were susceptible to Rowhammer. RowhammerJs also only ran on 385 datasets, slightly more than a third of all datasets.

As discussed in Section 4.2, 5 of the 7 Rowhammer tools failed on most datasets due to a lack of correct DRAM addressing functions. Additionally 3 of the 7 tools require 1 GiB hugepages, which are not always available. Thus, most tools worked only on less than a third of the datasets. Hence, any susceptibility numbers determined in this study may underestimate the real-world susceptibility to Rowhammer.

It is important to note that neither 1 GiB hugepages nor physical address information is required to typically available in a scenario where Rowhammer is weaponized for attacks like privlege escalation. We also did not perform any attacks in our framework, but limited to a large-scale study on the prevalence of Rowhammer itself. Therefore, actually using our framework for real attacks would require serious engineering effort. As discussed in Section 3.2, we did not add Zenhammer published by Jattke et al. [20] to our test suite. According to the authors, Zenhammer yields good results on AMD and even identified a DDR5 DIMM susceptible to Rowhammer. Therefore, we expect the fraction of affected AMD systems to be higher in reality than the 1.34 % reported in this paper.

As shown in Section 5.2, the majority of systems tested in our large-scale study has either DDR3 or DDR4 DRAM. Both together make approximately 80 % of the 1 006 systems tested in our large-scale study. Therefore, we gain only little insight on other DRAM generations, for which we also did not identify a single system to be susceptible to Rowhammer.

For many DIMMs, there is no clear vendor in the Serial Presence Detect (SPD) record. We use the larger DRAM vendors—Samsung, Hynix, and Micron—and group resellers (e. g., G-SKILL, A-DATA, etc.) in the group *other*, which together account for nearly 25 % of the DIMMs considered in our vendor-based evaluation.

Because the best respective tools on DDR3 and DDR4 ran only on roughly one third of all systems, the number of 126 out of 1 006 (12.5 %) should be seen as a lower estimation. We hypothesize that the number of susceptible systems in reality is significantly higher than reported in this paper due to the discussed limitations. In future studies, the overall grade of automation needs to be improved, focusing on a better detection and correction of

errors in general. Especially, DRAM bank addressing function reverse-engineering needs improvements to yield better results when run fully automated.

7. Conclusion

We performed a large-scale Rowhammer study and collected 1 006 datasets on 822 unique systems from the participants of our study. We show that automated DRAM addressing function reverse-engineering works only in 453 of the 1 006 datasets we analyze (less than 50 %). Due to missing DRAM addressing function, Rowhammer tools that require these functions were skipped on more than 50 % of the datasets. On datasets with DDR3, simple pattern and fast hammering are effective, while more complicated patterns and fuzzing are more effective on datasets with DDR4 due to TRR mitigating simple patterns on most systems. We detect the most bit flips on datasets with DDR3 DRAM (27.2 affected). On DDR4, only 8.76 % of all datasets are affected. However, many datasets with DDR4 have AMD CPUs, which we could not properly detect since we did not add the ZenHammer [20] tool[1]. When only focusing on Intel-based datasets, 11.1 % of the datasets with DDR4 are affected. DRAM from Samsung (31.7 % affected), Hynix (28.6 % affected), and third-party resellers where we could not resolve the actual DRAM vendor (22.8 % affected) are susceptible in the same order of magnitude. In contrast to them, DRAM from Micron is only affected by 2.4 %, which is surprising [28, 19, 35].

In general, our results should be seen as a *best case* estimation, since it is very likely that more systems are affected. A better automated response to errors in the tooling, manual interaction, longer runtimes and tools optimized for AMD would very likely increase the number of bit flips and the number of affected systems.

Acknowledgment

This work was funded by the Deutsche Forschungsgemeinschaft (grant number 503876675), the Austrian Science Fund (grant number 10.55776/I6054), the European Research Council (ERC project FSSec 101076409), as well

as the European Union under grant number ROF-SG20-3066-3-2-2. Additional funding was provided by generous gifts from Red Hat and Google. See Appendix 8.1 for additional acknowledgements.

Ethics Considerations

None of the institutions of the authors had an institutional review board (IRB) or an ethics committee to obtain approval for our study. Hence, we discussed our study design with multiple privacy researchers to make sure we follow strict ethical principles. One of the problems we identified was the contradiction that security professionals suggest not to put untrusted USB thumb drives into their computers. On the other hand, we wanted to make it as easy as possible to encourage people, even those without technical skills, to participate in our study. We took multiple steps to counter that. First, we published the source code of the software from our study, allowing everyone to review it and providing complete transparency. Second, we recommend building the ISO on their own and using that or the Docker container instead. If that is not possible, or the participant lacks the necessary technical skills, the USB thumb drive is the last option. Third, the hash digest of the image was also published, thereby enabling participants to verify whether the USB thumb drive had been modified.

Another point of discussion was the compliance with the General Data Protection Regulation (GDPR). We informed the user about the data we save and process, as well as its purpose, and the participant needs to acknowledge this explicitly before continuing with our study. Additionally, we informed the user about the potential, albeit rare, damage that the experiment can cause. After the participant ran the experiment, we asked the user again if they were sure they wanted to upload the data to our server. The participant could also finish running the framework without the requirement to send any data or participate in our study. We protect the dataset from unauthorised access using current state of the art measures. Furthermore, we regularly perform encrypted backups of the collected datasets to avoid data loss.

References

[1] Misiker Tadesse Aga, Zelalem Birhanu Aweke, and Todd Austin. When good protections go bad: Exploiting anti-DoS measures to accelerate Rowhammer attacks. In: HOST. 2017 (p. 166).

[2] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks. In: ASPLOS (2016) (p. 167).

[3] Sarani Bhattacharya and Debdeep Mukhopadhyay. Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis. In: CHES. 2016 (p. 161).

[4] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In: S&P. 2016 (p. 166).

[5] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In: USENIX Security. 2017 (p. 167).

[6] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In: S&P. 2019 (p. 166).

[7] Bernhard Fischer, Daniel Dorfmeister, Harald Lampesberger, and Eckehard Hermann. Leveraging Rowhammer for Physically Unique and Non-tamperable Device Identification. In: Procedia Computer Science (2025) (p. 169).

[8] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In: S&P. 2020 (pp. 162, 165–167, 173, 174, 180, 186, 189).

[9] Lukas Gerlach, Fabian Thomas, Robert Pietsch, and Michael Schwarz. A Rowhammer Reproduction Study Using the Blacksmith Fuzzer. In: ESORICS. 2023 (p. 166).

[10] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another Flip in the Wall of Rowhammer Defenses. In: S&P. 2018 (pp. 161, 162, 165–167, 173, 178, 181, 186).

[11] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016 (pp. 162, 165, 166, 173, 174, 178, 180).

[12] Wei He, Zhi Zhang, Yueqiang Cheng, Wenhao Wang, Wei Song, Yansong Gao, Qifei Zhang, Kang Li, Dongxi Liu, and Surya Nepal. WhistleBlower: A System-Level Empirical Study on RowHammer. In: IEEE Transactions on Computers (2023) (p. 167).

[13] Martin Heckel and Florian Adamsky. Flipper: Rowhammer on Steroids. In: uASC. 2025 (pp. 173, 174, 180).

[14] Martin Heckel and Florian Adamsky. Reverse-Engineering Bank Addressing Functions on AMD CPUs. In: DRAMSec Workshop. 2023 (pp. 162, 166, 172, 173, 185).

[15] Martin Heckel, Florian Adamsky, Jonas Juffinger, Fabian Rauscher, and Daniel Gruss. Verifying DRAM Addressing in Software. In: ESORICS. 2025 (p. 173).

[16] Martin Heckel, Hannes Weissteiner, Florian Adamsky, and Daniel Gruss. Epistemology of Rowhammer Attacks: Threats to Rowhammer Research Validity. In: ESORICS. 2025 (p. 167).

[17] Christian Helm, Soramichi Akiyama, and Kenjiro Taura. Reliable Reverse Engineering of Intel DRAM Addressing Using Performance Counters. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE. 2020 (p. 166).

[18] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In: SysTEX. 2017 (p. 166).

[19] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. BLACKSMITH: Rowhammering in the Frequency Domain. In: S&P. 2021 (pp. 162, 165–167, 173, 174, 180, 186, 189, 193).

[20] Patrick Jattke, Max Wipfli, Flavien Solt, Michele Marazzi, Matej Bölcskei, and Kaveh Razavi. ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms. In: USENIX Security. 2024 (pp. 161–163, 165–167, 173, 184, 185, 187, 192, 193).

[21] JEDEC Solid State Technology Association. DDR3 SDRAM STANDARD. 2012. URL (p. 165).

[22] JEDEC Solid State Technology Association. DDR4 SDRAM Standard. 2021. URL (p. 165).

[23] JEDEC Solid State Technology Association. DDR5 SDRAM Standard. 2024. URL (p. 165).

[24] Sangwoo Ji, Youngjoo Ko, Saeyoung Oh, and Jong Kim. Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks. In: AsiaCCS. 2019 (p. 166).

[25] Jonas Juffinger, Lukas Lamster, Andreas Kogler, Maria Eichlseder, Moritz Lipp, and Daniel Gruss. CSI: Rowhammer - Cryptographic Security and Integrity against Rowhammer. In: S&P. 2023 (p. 167).

[26] Jonas Juffinger, Sudheendra Raghav Neela, Martin Heckel, Lukas Schwarz, Florian Adamsky, and Daniel Gruss. Presshammer: Rowhammer and Rowpress without Physical Address Information. In: DIMVA. 2024 (p. 166).

[27] Ingab Kang, Walter Wang, Jason Kim, Stephan van Schaik, Youssef Tobah, Daniel Genkin, Andrew Kwong, and Yuval Yarom. SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism. In: USENIX Security. 2024 (pp. 165, 166).

[28] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors. In: ISCA. 2014 (pp. 161, 162, 165–167, 186, 189, 193).

[29] Kirill A. Shutemov. Pagemap: Do Not Leak Physical Addresses to Non-Privileged Userspace. 2015. URL (p. 162).

[30] Andreas Kogler, Jonas Juffinger, Salman Qazi, Yoongu Kim, Moritz Lipp, Nicolas Boichat, Eric Shiu, Mattias Nissler, and Daniel Gruss. Half-Double: Hammering From the Next Row Over. In: USENIX Security. 2022 (pp. 162, 166).

[31] Radhesh Krishnan Konoth, Marco Oliverio, Andrei Tatar, Dennis Andriesse, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. ZebRAM: Comprehensive and Compatible Software Protection Against Rowhammer Attacks. In: USENIX OSDI. 2018 (p. 167).

[32] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. RAMBleed: Reading Bits in Memory Without Accessing Them. In: S&P. 2020 (p. 166).

[33] Chris S. Lin, Joyce Qu, and Gururaj Saileshwar. GPUHammer: Rowhammer Attacks on GPU Memories are Practical. In: USENIX Security. 2025 (pp. 161, 166).

[34] Moritz Lipp, Misiker Tadesse Aga, Michael Schwarz, Daniel Gruss, Clémentine Maurice, Lukas Raab, and Lukas Lamster. Nethammer: Inducing Rowhammer Faults through Network Requests. In: SILM Workshop. 2020 (p. 166).

[35] Haocong Luo, Ataberk Olgun, Abdullah Giray Yağlikçi, Yahya Can Tuğrul, Steve Rhyner, Meryem Banu Cavlak, Joël Lindegger, Mohammad Sadrosadati, and Onur Mutlu. RowPress: Amplifying Read Disturbance in Modern DRAM Chips. In: ISCA. 2023 (pp. 166, 173, 174, 180, 189, 193).

[36] Michele Marazzi and Kaveh Razavi. RISC-H: Rowhammer Attacks on RISC-V. In: DRAMSec Workshop. 2024 (p. 166).

[37] Ataberk Olgun, Yahya Can Tugrul, Nisa Bostanci, Ismail Emir Yuksel, Haocong Luo, Steve Rhyner, Abdullah Giray Yaglikci, Geraldo F Oliveira, and Onur Mutlu. ABACuS: All-Bank Activation Counters for Scalable and Low Overhead RowHammer Mitigation. In: USENIX Security. 2024 (p. 167).

[38] Lois Orosa, Ulrich Rührmair, A Giray Yaglikci, Haocong Luo, Ataberk Olgun, Patrick Jattke, Minesh Patel, Jeremie Kim, Kaveh Razavi, and Onur Mutlu. SpyHammer: Using RowHammer to Remotely Spy on Temperature. In: arXiv:2210.04084 (2022) (p. 166).

[39] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: USENIX Security. 2016 (pp. 162, 165, 166, 173, 185).

[40] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. In: USENIX Security. 2016 (pp. 165, 166).

[41] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript. In: USENIX Security. 2021 (pp. 162, 165, 166).

[42] Andre Schaller, Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Stefan Katzenbeisser, and Jakub Szefer. Intrinsic Rowhammer PUFs: Leveraging the Rowhammer effect for improved security. In: Hardware Oriented Security and Trust (HOST. 2017 (p. 169).

[43] Mark Seaborn. Exploiting the DRAM rowhammer bug to gain kernel privileges. 2015. URL (pp. 161, 162, 166, 167, 185).

[44] Mark Seaborn and Thomas Dullien. Test DRAM for bit flips caused by the rowhammer problem. 2015. URL (pp. 173, 178).

[45] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer. In: RAID. 2018 (p. 166).

[46] Andrei Tatar, Radhesh Krishnan, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer Attacks over the Network and Defenses. In: USENIX ATC. 2018 (p. 166).

[47] Youssef Tobah, Andrew Kwong, Ingab Kang, Daniel Genkin, and Kang G Shin. SpecHammer: Combining Spectre and Rowhammer for New Speculative Attacks. In: S&P. 2022 (p. 166).

[48] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic Rowhammer Attacks on Mobile Platforms. In: CCS. 2016 (pp. 165, 166).

[49] Victor van der Veen, Martina Lindorfer, Yanick Fratantonio, Harikrishnan Padmanabha Pillai, Giovanni Vigna, Christopher Kruegel, Herbert Bos, and Kaveh Razavi. GuardION: Practical Mitigation of DMA-Based Rowhammer Attacks on ARM. In: DIMVA. 2018 (p. 167).

[50] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. DRAMDig: A Knowledge-assisted Tool to Uncover DRAM Address Mapping. In: Design Automation Conference (DAC). 2020 (pp. 162, 166, 173).

[51] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: USENIX Security. 2016 (p. 166).

[52] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. Triggering Rowhammer Hardware Faults on ARM: A Revisit. In: ASHES Workshop. 2018 (p. 166).

8. Appendix

8.1. Further Acknowledgement

In addition to the acknowledgements for funding shown above, we also want to thank the people who supported us during the design, development, and conduct of our study.

We want to thank Lena Heimberger of Graz University of Technology for having many discussions about the design of our study regarding privacy and transparency, they significantly improved the final study design. We want to thank Katharina Schiller for creating the final FlippyRAM logo. We want to thank Claudia Ceh for creating the project website at `https://flippyr.am`. We want to thank Nico Bretschneider, Antje Heckel, Johanna Heckel, Theresa Heckel, Ulrich Heckel, and Paulina Zschippang for flashing 2 000 thumb drives over the Christmas Holidays in 2024.

We want to thank Jeremy Boy, Anna Pätschke, Thore Tiemann, and Thomas Eisenbarth from the University of Lübeck for distributing several hundred thumb drives to their students and asking them to participate in our large-scale study. We want to thank Andreas Schmidt for giving a Lightning talk on Chemnitz Linux Days and distributing thumb drives to the audience.

Finally, we want to thank our many participants who made this large-scale study possible. Among students at Hof University of Applied Sciences, Graz University of Technology, and the University of Lübeck, many others also participated. Even though most of the participants want to stay anonymous, we provide a list of some participants who agreed to be named in our Acknowledgements (in alphabetical order):

Malte Behrmann, Arne Bier, Micha Borrmann, Jeremy Boy, Colipedia, daef, Emrah Delanović, Jörg Elfring, Felix Fehlauer, Morgan Gothard, Brian Hobbs, Jan Christopher Kemnitzer, Daniel Kipp, Ingo Korb, @magni@chaos.social, Philippe Marschall, Justin McLemore, Natalie Mirelashvili, Malte Oeljeklaus, Adriel Ondas, Anja Ostovršnik, Anna Pätschke, Peter Rohrer, Alexander Schnell, Robin Leander Schröder, Michał Trojnara, Benjamin Walter, Qifan Wang, and David Ward.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used anything other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.