FLIPPYRAM: A Large-Scale Study of Rowhammer Prevalence

Martin Heckel
Hof University of Applied Sciences
martin.heckel.2@hof-university.de

Nima Sayadi
Hof University of Applied Sciences
nima.sayadi@hof-university.de

Jonas Juffinger
Graz University of Technology
jonas.juffinger@tugraz.at

Carina Fiedler
Graz University of Technology
carina.fiedler@tugraz.at

Daniel Gruss
Graz University of Technology
daniel.gruss@tugraz.at

Florian Adamsky
Hof University of Applied Sciences
florian.adamsky@hof-university.de

***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.

Network and Distributed System Security (NDSS) Symposium 2026
23 - 27 February 2026 , San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.241810
www.ndss-symposium.org

I. 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 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

[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.

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, given that 50 % of datasets without bit flips failed in the DRAM reverse-engineering stage.

II. BACKGROUND

In this section, we provide background on DRAM, Rowhammer and related works.

A. 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].

B. 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]. Rowhammer can be exploited via JavaScript [11, 41]. These bit flips can be exploited, 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 Rowhammer 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 Activation 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] https://github.com/iisys-sns/flippyram (archived version at https://doi.org/10.5281/zenodo.17881765)

C. 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 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.

III. 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 1 depicts the overall workflow of our study, consisting of six steps: The first and the last step are discussed in Section III-A, i. e., user study design, data collection, and privacy aspects. Steps 2 and 4 are discussed in Section III-B, 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 III-C. Step 5, the execution of up to 7 state-of-the-art Rowhammer tools, is discussed in Section III-D.

A. 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.

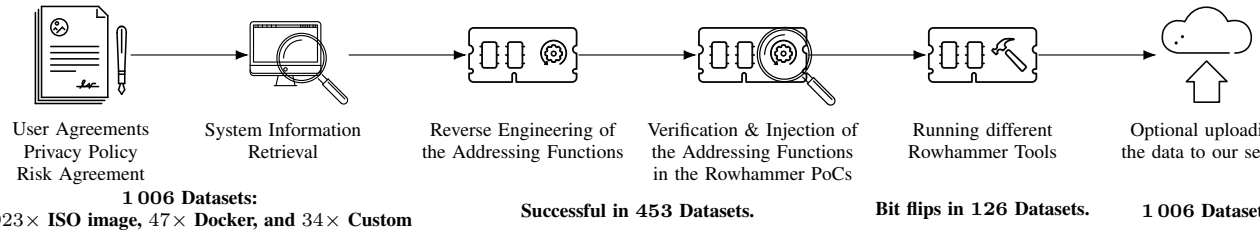


Fig. 1. The workflow of our software to automatically test if a system is vulnerable to the Rowhammer effect. It starts right after someone booted our ISO image or ran our docker container.

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.

B. 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[I]. 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 III-D. We also integrated 5 DRAM reverse-engineering tools to recover the DRAM addressing functions, as discussed in Section III-C. 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.

C. 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.

D. 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).

E. 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.

IV. 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 III, 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.

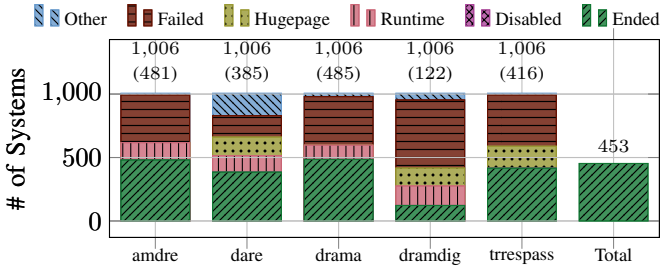


Fig. 2. Number of systems with specific states of different Addressing Function Reverse-Engineering tools. The number above the bar depicts the absolute number of datasets. The number in braces shows the number of datasets on which the tool had the *Ended* state. The *Total* bar depicts the number of 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).

A. Address Function Reverse-Engineering

We analyze the execution of different DRAM addressing reverse-engineering tools, as summarized in Figure 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.

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.

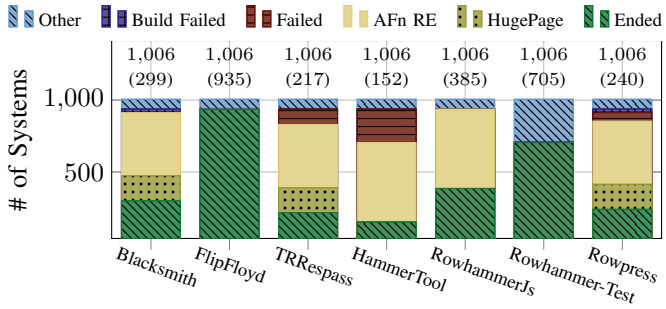


Fig. 3. Number of datasets with specific states of different Rowhammer tools. The number above the bar depicts the absolute number of datasets. The number in braces shows the number of datasets on which the tool had the *Ended* state, e. g., ran successfully.

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.

B. Rowhammer Tools

We also analyze the execution of the different Rowhammer tools based on the previously defined tool states, as summarized in Figure 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 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.

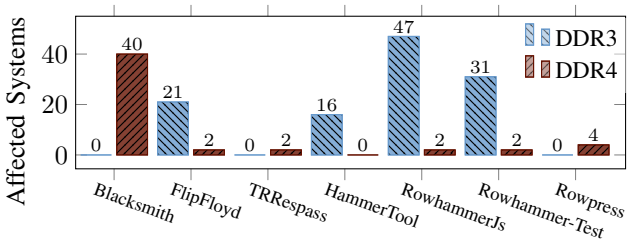
C. Success Rate of Rowhammer Tools

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 I and Figure 4a. Additionally, we evaluate the number of bit flips triggered by different tools (see Figure 4b).

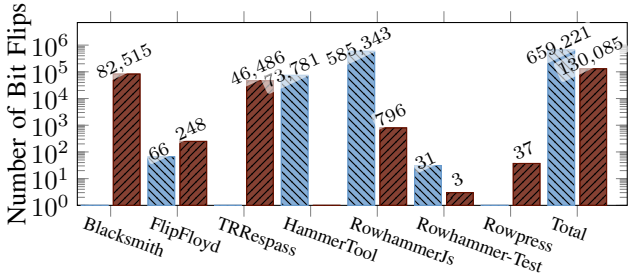
FlipFloyd [10], RowhammerJS [11], and Rowhammer-Test [44] induced bit flips predominantly in DDR3 DRAM but also in a few datasets with DDR4 DRAM. Of these, RowhammerJS yielded the best results regarding the number of affected datasets (49) and the number of bit flips (586 139). Regarding the total number 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

TABLE I
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.

Fig. 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.

(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 re-

quire 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 I 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 III-D). 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.

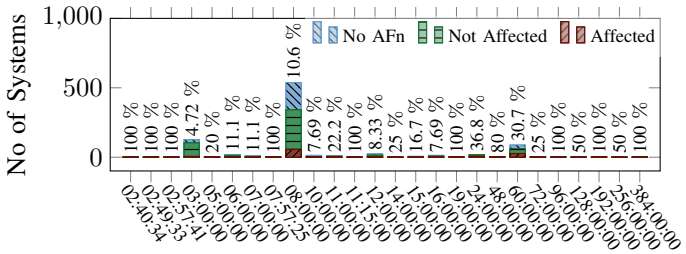


Fig. 5. Number of datasets affected by Rowhammer grouped by Runtime. The bars of affected and not affected datasets stack to the total number of datasets with the runtime. The percentage depicted above the bars it the percentage of affected datasets with that runtime.

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.

D. 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 run based on the total runtime as described in Section III-C. 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 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 III-C). However, the trend also continues for runtimes of more than 7 h, where all reverse-engineering tools are enabled. Based on these results, we conclude that longer testing leads to more systems being correctly identified as susceptible to Rowhammer.

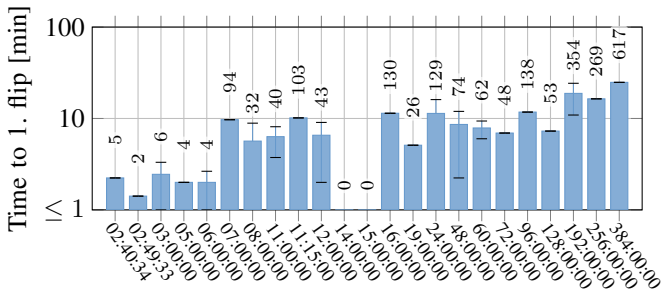


Fig. 6. Time until the first bit flip grouped by total runtime. The bars show the average time, the error bars the standard deviation of the fastest tool on each dataset. Low values rounded to 1 minute.

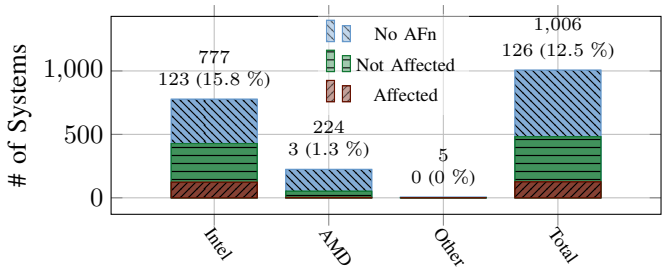


Fig. 7. Datasets affected by Rowhammer grouped by CPU Vendor. The bars of affected, not affected, and AFn failed datasets stack to the total number of datasets with the CPU vendor. The numbers depicted above the bars are the absolute number of datasets on the top and the percentage of affected datasets 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.

TABLE II
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 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.

V. 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.

A. 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 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 III-B. 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 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 II. 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.

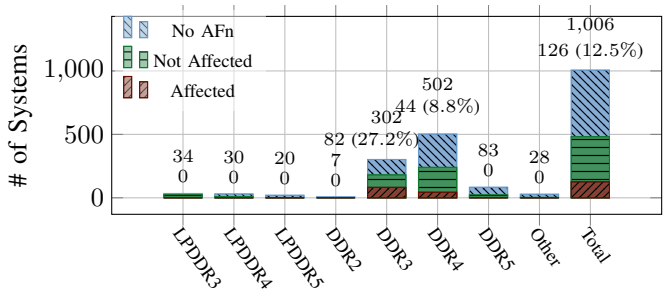


Fig. 8. Datasets affected by Rowhammer grouped by DRAM Generation. The bars of affected, not affected, and No AFn datasets stack to the total number of datasets with the DRAM Generation. The numbers depicted above the bars are the absolute number of datasets on the top and the number and percentage of affected datasets on the bottom.

B. 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) 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 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 (σ = 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 V-A). Focusing only on the 388 DDR4 with Intel CPUs, 43 (11.1 %) are affected. It takes on average 39.9 min (σ = 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 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 III-B.

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]

TABLE III
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

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.

C. 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 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 III-E. Figure 9 provides an overview of these results.

Our results (cf. Table III) 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 (σ = 160 min), the longest time of all vendors. It is followed by DIMMs from Hynix with an average time of 56 min (σ = 50.4 min) until the first bit flip. For DIMMs from Micron, the first bit flip occurred after 1.95 min (σ = 3.79 min). DIMMs where the vendor could not be mapped show the first flip on average after 31.4 min (σ = 44.4 min).

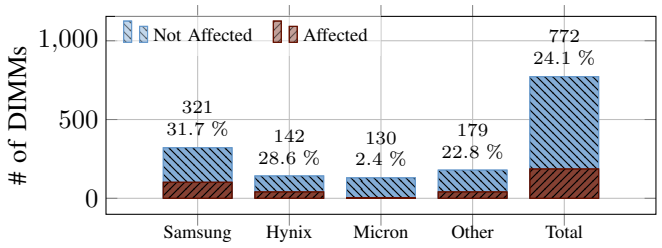


Fig. 9. DIMMs affected by Rowhammer grouped by Vendor. The bars of affected, and not affected systems stack to the total number of DIMMs from that Vendor. The numbers depicted above the bars are the absolute number of DIMMs on the top and the percentage of affected DIMMs on the bottom.

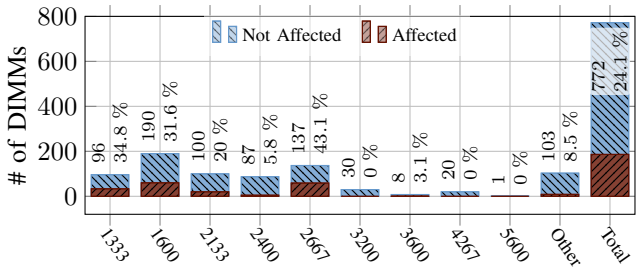


Fig. 10. Susceptibility of DIMMs to Rowhammer grouped by Speed (Frequency). The bars of affected and not affected DIMMs stack to the total number of DIMMs with the DRAM Speed (Frequency). The numbers depicted above the bars are the absolute number of DIMMs on the left side and the percentage of affected DIMMs on the right side.

TABLE IV
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

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 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.

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].

D. 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 IV and Figure 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 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 III-E. 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 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 IV-D, 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.

VI. Limitations

In this section, we discuss limitations of our results.

Most addressing function revese-engineering tools failed on many systems as described in Section IV-A. 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 IV-B, 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 III-B, 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 V-B, 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.

VII. 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 A 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.

[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).

[3] Sarani Bhattacharya and Debdeep Mukhopadhyay. "Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis". In: *CHES*. 2016.

[4] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *S&P*. 2016.

[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.

[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.

[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).

[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.

[9] Lukas Gerlach, Fabian Thomas, Robert Pietsch, and Michael Schwarz. "A Rowhammer Reproduction Study Using the Blacksmith Fuzzer". In: *ESORICS*. 2023.

[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.

[11] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: *DIMVA*. 2016.

[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).

[13] Martin Heckel and Florian Adamsky. "Flipper: Rowhammer on Steroids". In: *uASC*. 2025.

[14] Martin Heckel and Florian Adamsky. "Reverse-Engineering Bank Addressing Functions on AMD CPUs". In: *DRAMSec Workshop*. 2023.

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

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

[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.

[18] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. "SGX-Bomb: Locking Down the Processor via Rowhammer Attack". In: *SysTEX*. 2017.

[19] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. "BLACKSMITH: Rowhammering in the Frequency Domain". In: *S&P*. 2021.

[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.

[21] JEDEC Solid State Technology Association. *DDR3 SDRAM STANDARD*. 2012. URL.

[22] JEDEC Solid State Technology Association. *DDR4 SDRAM Standard*. 2021. URL.

[23] JEDEC Solid State Technology Association. *DDR5 SDRAM Standard*. 2024. URL.

[24] Sangwoo Ji, Youngjoo Ko, Saeyoung Oh, and Jong Kim. "Pinpoint Rowhammer: Suppressing Unwanted Bit Flips on Rowhammer Attacks". In: *AsiaCCS*. 2019.

[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.

[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.

[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.

[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.

[29] Kirill A. Shutemov. *Pagemap: Do Not Leak Physical Addresses to Non-Privileged Userspace*. 2015. URL.

[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.

[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.

[32] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. "RAMBleed: Reading Bits in Memory Without Accessing Them". In: *S&P*. 2020.

[33] Chris S. Lin, Joyce Qu, and Gururaj Saileshwar. "GPUHammer: Rowhammer Attacks on GPU Memories are Practical". In: *USENIX Security*. 2025.

[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.

[35] 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.

[36] Michele Marazzi and Kaveh Razavi. "RISC-H: Rowhammer Attacks on RISC-V". In: *DRAMSec Workshop*. 2024.

[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.

[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).

[39] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks". In: *USENIX Security*. 2016.

[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.

[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.

[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.

[43] Mark Seaborn. *Exploiting the DRAM rowhammer bug to gain kernel privileges*. 2015. URL.

[44] Mark Seaborn and Thomas Dullien. *Test DRAM for bit flips caused by the rowhammer problem*. 2015. URL.

[45] Andrei Tatar, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. "Defeating Software Mitigations Against Rowhammer: A Surgical Precision Hammer". In: *RAID*. 2018.

[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.

[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.

[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.

[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.

[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.

[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.

[52] Zhenkai Zhang, Zihao Zhan, Daniel Balasubramanian, Xenofon Koutsoukos, and Gabor Karsai. “Triggering Rowhammer Hardware Faults on ARM: A Revisit”. In: *ASHES Workshop*. 2018.

Appendix A
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.