Flipper: Rowhammer on Steroids

Martin Heckel[1,2] [0009−0006−9739−0587] and
Florian Adamsky[1] [0009−0002−6642−6904]

[1] Hof University of Applied Sciences, Hof, Germany
[2] 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 Although DDR3 memory has been superseded by DDR4 and DDR5 memory technologies, it is still widely used in devices that do not require frequent replacement, such as projectors, smart displays, servers,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 [21].

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, 5, 6, 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[3].

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.

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

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. 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 [21]. 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. [5] 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 1.1 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].

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[4] 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

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

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

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. [5] 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 [20]. Previous work shows that parallel hammering can have positive [23, 13, 7, 19] and negative [28] impacts on the number of bit flips occurring.

This is why we implemented a multithreading mode in our Rowhammer proof-of-concept (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 $\text{n\_threads} = min(\text{n\_logical\_cpus}, \text{n\_memory\_banks})$ 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[5] 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 rate mitigation. Details of this measurement can be found in Section 4.2. All systems run Arch Linux. Table 1 shows the hardware specification of our setup.

Table 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 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

[5] 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.

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 [5] 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.

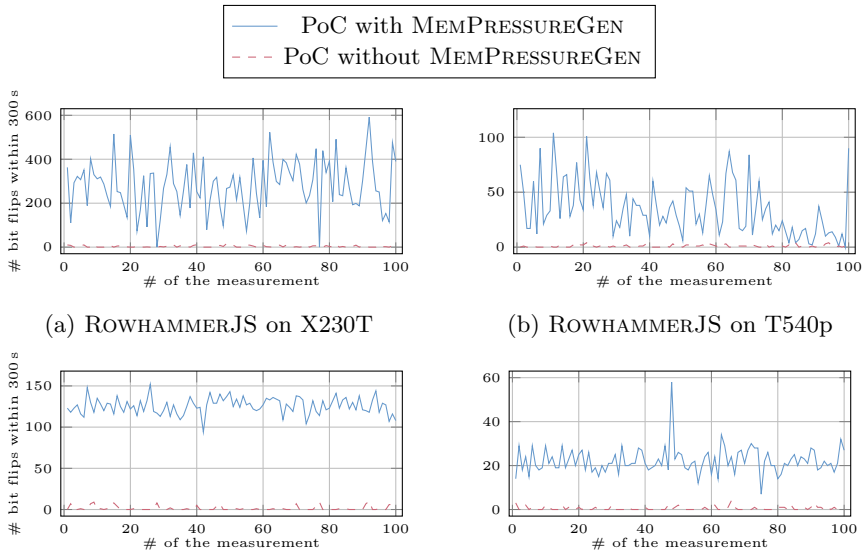
We also tried to use the implementation of hammertime introduced by Tatar et al. [36] as the authors state that their implementation is able to find orders of magnitude more bit flips on their test system. However, we were unable to identify any bit flips using hammertime on our test systems. Since this 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 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.

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.



(a) RowhammerJS on X230T

(b) RowhammerJS on T540p

(c) HammerTool on X230T

(d) HammerTool on T540p

Fig. 1: Number of bit flips measured within 300 s with and without MemPressureGen running in parallel.

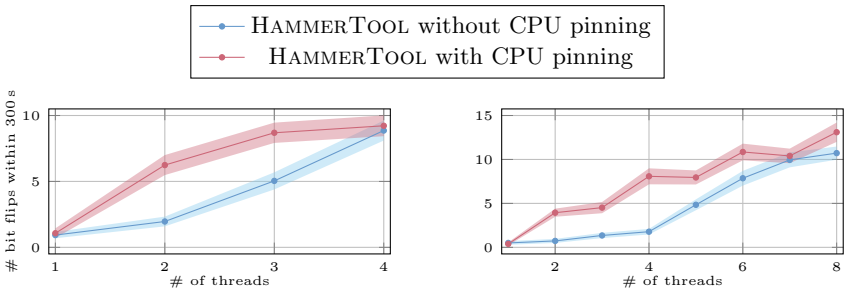
Without MemPressureGen running in parallel, RowhammerJS finds approximately 0.91 bit flips. In the same time, there are approximately 35.09 bit flips found when MemPressureGen is running. So, the usage of MemPressureGen increases the amount of bit flips by a factor of 38.56. The usage of MemPressureGen 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 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

Fig. 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 CPUs 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[6]. 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 CPUs 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.

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

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 2b). This effect does not occur on the X230T (see Figure 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 → L1 bandwidth of 32 bytes per cycle while Haswell has a L2 → L1 bandwidth of 64 bytes per cycle [32]. However, for both microarchitectures, the L2→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 → L1 and L3 → L2. So, running two hyperthreads saturates the bandwidth of the sytem.

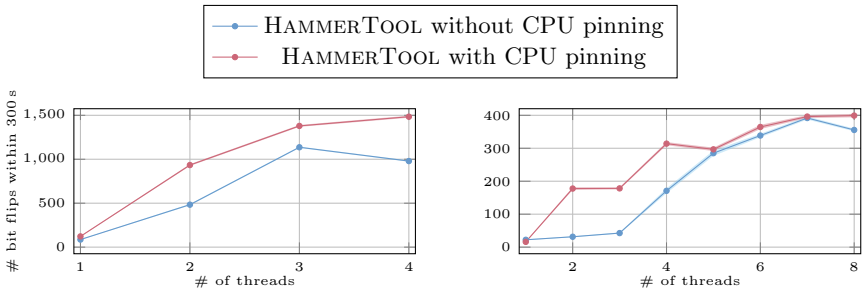
In contrast to that, on Haswell, two hyperthreads can perform one 32 byte load each in one cycle resulting in 64 bytes being loaded. The L2 → L1 bandwidth is 64 bytes per cycle as well, so this saturates the L2 → L1 bandwidth as well. However, the L3 → L2 bandwidth is only 32 bytes per cycle and thereby 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 3 shows the amount of bit flips found with CMP_{IST} running at the same time.

The result of the experiment from Figure 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



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

Fig. 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.

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.

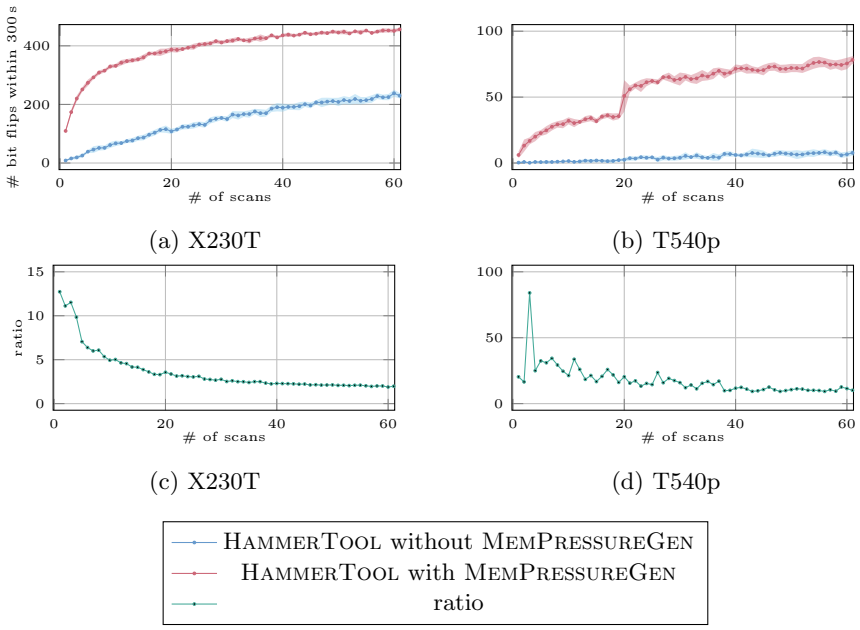
4.7 Comparison of Attacks based on Unique Bit Flips

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 4 depicts the amount of unique bit flips.



(a) X230T (b) T540p

(c) X230T (d) T540p

Fig. 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.

The measurements on both systems depicted in Figures 4a and 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 4c and 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 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 5 depicts the results of the measurement.

When MemPressureGen is used (see Figure 5a and Figure 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 5a) and the T540p (see Figure 5b).

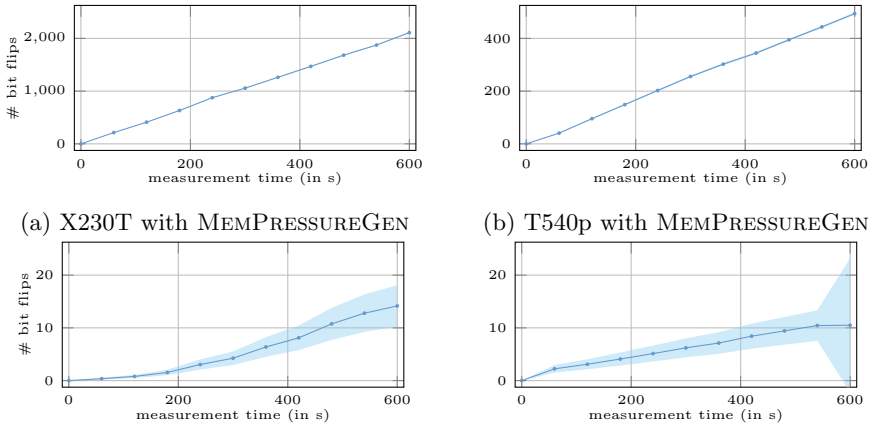
In contrast to that, when MemPressureGen is not used (see Figure 5c and Figure 5d), the number of bit flips found in a given time does not seem to be linearly correlated to that time. However, when looking at the error bars, it is likely that they are linear as well and the difference from the linear 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.

Based on the results shown in Figure 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,



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

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

Fig. 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.

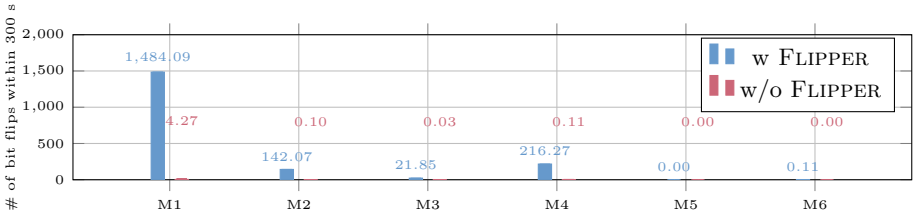
as shown in Section 4.4, Flipper effectively yields 837.5 times more bit flips in the same time frame. This is an improvement over the state of the art by almost 3 orders of magnitude.

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.

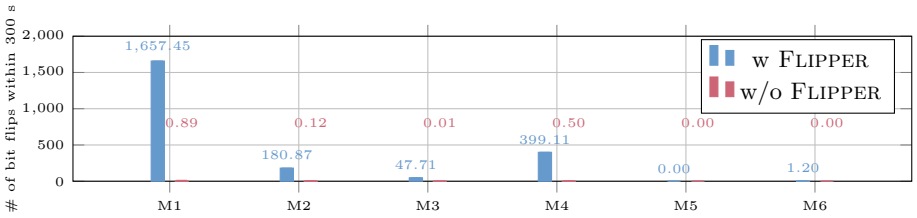
5 Discussion on the Security Impact

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 Page Table Entry (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. [6] 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].



(a) X230T



(b) T540p

Fig. 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.

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. [21] 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. [5] 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. [21] 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 [5] 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`.

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

[3] Erik Bosman et al. "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector". In: *S&P*. 2016.

[4] Pietro Frigo et al. "TRRespass: Exploiting the Many Sides of Target Row Refresh". In: *S&P*. 2020.

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

[6] Daniel Gruss et al. "Another Flip in the Wall of Rowhammer Defenses". In: *S&P*. 2018.

[7] Wei He et al. "WhistleBlower: A System-level Empirical Study on RowHammer". In: *IEEE Transactions on Computers* (2023).

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

[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.

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

[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/.

[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.

[13] Yeongjin Jang et al. "SGX-Bomb: Locking Down the Processor via Rowhammer Attack". In: *SysTEX*. 2017. DOI: 10.1145/3152701.3152709.

[14] Patrick Jattke et al. "BLACKSMITH: Rowhammering in the Frequency Domain". In: *S&P*. Nov. 2021.

[15] Patrick Jattke et al. "ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms". In: *USENIX Security*. 2024.

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

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

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

[19] Ingab Kang et al. "SledgeHammer: Amplifying Rowhammer via Bank-level Parallelism". In: *USENIX Security*. 2024.

[20] Yoongu Kim et al. "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM". In: *ISCA*. 2012. DOI: 10.1109/ISCA.2012.6237032.

[21] Yoongu Kim et al. “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors”. In: *ISCA*. 2014. DOI: 10.1109/ISCA.2014.6853210.

[22] Andreas Kogler et al. “Half-Double: Hammering From the Next Row Over”. In: *USENIX Security*. 2022.

[23] Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. 2016. URL: http://www.thirdio.com/rowhammer.pdf.

[24] Moritz Lipp et al. “Nethammer: Inducing Rowhammer Faults through Network Requests”. In: *SILM Workshop*. 2020.

[25] Haocong Luo et al. “RowPress: Amplifying Read Disturbance in Modern DRAM Chips”. In: *ISCA*. 2023.

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

[27] Peter Pessl et al. “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: *USENIX Security*. 2016.

[28] Rui Qiao and Mark Seaborn. “A New Approach for Rowhammer Attacks”. In: *HOST*. 2016.

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

[30] Kaveh Razavi et al. “Flip Feng Shui: Hammering a Needle in the Software Stack”. In: *USENIX Security*. 2016.

[31] Finn de Ridder et al. “SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript”. In: *USENIX Security*. 2021.

[32] Subhash Saini et al. “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.

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

[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.

[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.

[36] Andrei Tatar et al. “Defeating software mitigations against rowhammer: a surgical precision hammer”. In: *RAID*. 2018.

[37] Andrei Tatar et al. “Throwhammer: Rowhammer Attacks over the Network and Defenses”. In: *USENIX Security*. July 2018.

[38] Victor van der Veen et al. “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: *ACM CCS*. 2016.

[39] Minghua Wang et al. “Dramdig: A Knowledge-assisted Tool to Uncover-DRAM Address Mapping”. In: *Design Automation Conference (DAC)*. 2020.

[40] Yuan Xiao et al. “One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation”. In: *USENIX Security*. 2016.

[41] Zhenkai Zhang et al. “Triggering Rowhammer Hardware Faults on ARM: A Revisit”. In: *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*. 2018.