Novel Rowhammer Attacks

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

***Abstract*—In current Dual In-line Memory Modules (DIMMs), the density of Dynamic Random-Access Memory (DRAM) cells is so high that frequently accessing rows in memory can lead to disturbance errors in nearby rows. The exploitation of this side-effect is known as rowhammer and allows an attacker to modify the content of memory rows without actually accessing them.**

This effect was not estimated to be security-relevant in real-world scenarios at first. However, in the last years, many real-world exploits were published: Researchers showed that it is possible to escalate privileges on local systems, mobile devices and cloud platforms. Since then, several mitigation techniques have been developed to avoid the exploitation of this side-effect.

In the scope of my bachelor thesis, I introduced two basic attacks that can help to bypass the mitigation of a double refresh rate by significantly increasing the amount of bit flips found in a given time. This mitigation is often used on systems with DDR3 DRAM.

In this paper, those two basic attacks are evaluated in more depth: Several experiments inspect the effects of those attacks individually and in combination. It is shown that the combination of those attacks leads to an increase in the amount of bit flips found in a given time to a hundredfold on systems featuring the mitigation of a double refresh rate.

I. List of publications

Most of the ideas and images in this paper are part of the scientific publication:

- FLIPPER: Rowhammer on Steroids — Exploiting the x86 cmpsb Instruction and Parallelising Hammering [1] submitted to the 26th European Symposium on Research in Computer Security (ESORICS) 2021 [2].

II. Introduction

Current computer systems consist of a lot of different components that build up on each other: Application software builds up on libraries which build up on the operating system which builds up on the hardware. For each of those layers, it is required to trust the layers below: If one of the lower layers fails, the upper layers fail as well.

Typically, those computer systems are shared ones: Multiple processes run on the same Operating System (OS) which is responsible to isolate those processes from each other. One part of this isolation is to avoid one process messing with other processes memory.

In Dynamic Random-Access Memory (DRAM), data is stored in memory cells which consist of capacitors and transistors. Those cells are organized in arrays of rows and columns. When an attacker reads a row in DRAM frequently, this leads to currents floating into and from the cells of the row. Those currents can lead to disturbance errors in nearby rows effectively changing the stored value from a 1 to a 0 or vice versa. Thereby, an attacker is able to flip bits in memory.

The bits that are flipped can be chosen by an attacker in a limited way: If a row is accessed frequently, the bits in the rows before and after that rows are likely to flip. It is not possible for an attacker to select single bits, but only to access memory rows and maybe, one or multiple bits in nearby rows will flip. Because of that limitation, rowhammer was considered to be a rather theoretical problem when it was discovered. However, researchers showed that there are many ways of rowhammer exploitation with significant security implications on real-world systems. Those exploitations techniques made it possible to escalate privileges locally on desktop computers and mobile devices [3]–[5], to execute code on Virtual Machines (VMs) running on the same host than an attackers VM [6], to flip bits from within browsers using JavaScript [7], [8], and to flip bits over the network [9], [10].

The root cause for rowhammer is the high integration density of the Integrated Circuits (ICs) used in DRAM. However, reducing this density to a level that would avoid rowhammer would lead to a massive decrease in memory capacity. For this reason, there are several mitigation techniques that should help to avoid the exploitation of rowhammer without addressing the root problem. In the last years, many of those mitigation techniques were proven to be not as effective as assumed [5], [11].

In my practical thesis [12] bachelor thesis [13], I introduced two basic attacks that can be used to increase the number of bit flips found in a given time significantly. In this paper, the effects of both attacks, individually and in combination, are evaluated. It is shown that combining both attacks can lead to an increase in the number of bit flips found in a given time by a factor of hundredfold.

III. Background

This section describes how the rowhammer exploit works. Because the understanding of rowhammer is based on the understanding of the basic memory architecture, a short overview of the memory architecture is provided before. Afterwards, a short overview of the currently used mitigation techniques is given.

A. Memory Architecture

DRAM cells consisting of capacitors and transistors are organized in an array of rows and columns. In addition to that array of cells, there is the so called *row buffer* which stores exactly one row from the DRAM array. Between the

array and the row buffer, there are components that amplify the signal from the DRAM cells.

Because the capacitors in the DRAM cells leak charge over time, it is required to refresh them periodically. Typically, this is done every 64 ms for a cell. During a refresh, the charge of a cell is read, amplified, and written back to the cell afterwards.

The combination of DRAM array, row buffer and amplifiers is called *bank*. Typically, those banks are organized in *ranks* and located on *Dual In-line Memory Modules (DIMMs)*. Thereby, one DIMM can contain one or multiple ranks. Those DIMMs are connected to the Central Processing Unit (CPU) with buses called *channels*. There are systems with multiple of those channels which leads to the possibility to access multiple DIMMs at the same time if they are on different channels.

Data from DRAM is always accessed from the row buffer. For that reason, the row containing the requested data has to be load into the row buffer first. After that, the requested data can be returned from the row buffer. Because reading a row from the DRAM array discharges all capacitors in the corresponding DRAM cells, the row has to be restored from the row buffer at a later point in time. There are three different possibilities when data is accessed from DRAM related to the row buffer:

- **row hit**: The requested row is already in the row buffer of the corresponding DRAM bank. So, no access to the DRAM array is required and the data can be returned from the row buffer directly.
- **empty row**: The row buffer of the corresponding DRAM bank is empty. For that reason, the row containing the requested data has to be load into the row buffer. Afterwards, the requested data can be returned from the row buffer.
- **row conflict**: Another than the requested row is in the row buffer of the corresponding DRAM bank. So, the row that is currently in the row buffer has to be restored to the DRAM array before the new row can be load from the array into the row buffer. Afterwards, the requested data can be returned from the row buffer.

B. Rowhammer

When two values that are located at the same DRAM bank but in different rows are accessed alternatingly, this leads to many row conflicts on that bank[1]. If it is assumed that those values are the only ones accessed on that bank, there is always the "wrong" row in the row buffer, so that row has to be restored to the DRAM array before the next row can be load from the DRAM array.

Those accesses lead to currents floating from and into the DRAM cells of the corresponding rows. Due to the high integration density of current DRAM, the cells are so near to each other that those currents can interfere with nearby memory cells. Thereby, charges can leak slowly from and to memory cells that are not even accessed.

This is not a problem if the charge level, and thereby the corresponding voltage, of the capacitor stays within the limits of the logical state (e.g. a full capacitor can discharge as long as its voltage level is still interpreted as *full*, and vice versa). This state has to be maintained until the next refresh cycle: During refresh, the current of all DRAM cells is set to the defined level again (e.g. a full capacitor is completely charged, an empty capacitor is completely discharged)

For that reason, bits are flipping if, and only if, there are so many accesses that the charge of the capacitors changes significantly during one refresh period (by default 64 ms) and is interpreted as the opposite state at the next refresh cycle.

In order to avoid the data being accessed from the CPU cache instead of the DRAM, the data has to be removed from the cache. There are multiple approaches to achieve that: The easiest one is to use the `CLFLUSH` instruction. However, this is only possible on systems providing that (or a similar) instruction. Gruss, Maurice, and Mangard [7] showed that it is possible to remove data from the CPU cache using cache eviction instead of the `CLFLUSH` or a similar instruction.

C. Mitigation techniques

The root cause for rowhammer is the high integration density of current DRAM. However, that high integration density is required to get a reasonable amount of memory for affordable prices. Therefore, the reduction of the integration density would not be a practicable solution.

For that reason, there are different mitigation techniques that are used to avoid the exploitation:

- **Double refresh rate**: In order to flip bits, it is required to leak enough charge from or into a DRAM cell that the charge level is interpreted as the logical opposite. Because the charge is completely restored at refresh, it is required to leak the required amount of charge between two refreshes. One early mitigation was to double the refresh rate by changing the refresh period from 64 ms to 32 ms. This mitigation has the drawback that it has a negative impact on performance and energy-efficiency [14]. Additionally, there are still bit flips possible with that mitigation installed [13]. This will be evaluated in more detail in the scope of this paper.
- **Error Correction Code (ECC)**: The mitigation of using ECC DRAM takes another approach: If a bit flips and this is detected and fixed, that is not a problem. Cojocar *et al.* [15] demonstrated that is is possible to reliably exploit rowhammer even on systems featuring ECC DRAM.
- **pseudo Target Row Refresh (pTRR)**: In contrast to the mitigation that just doubles the refresh rate, pTRR analyzes memory accesses and calculated possible victim rows. If a specific threshold of accesses to nearby rows is exceeded, the possible victim rows are refreshed. The threshold is defined within the Serial Presence Detect (SPD) Electrically Erasable Programmable Read-Only Memory (EEPROM) of the DIMM. pTRR is implemented on the Memory Management Unit (MMU). If no threshold is defined, a double refresh rate is used as fallback.

[1]There are rowhammer variants that use a different number of values, e.g. one or three, etc.

- **Target Row Refresh (TRR)**: In contrast to pTRR, TRR is implemented on the DIMM rather than the MMU. Frigo *et al.* [5] showed in their paper that the mitigation effect of TRR is not as height as thought before.

IV. Rowhammer Amplification

In my practical thesis [12] and bachelor thesis [13], I introduced two approaches that can be used to increase the amount of bit flips found in a given time. In this section, those approaches are explained briefly. The combination of both approaches is called FLIPPER.

A. Parallising rowhammer attacks

As described in Section III-B, the exploitation of rowhammer requires data which are stored in the same DRAM bank. For that reason, it is possible to execute multiple rowhammer attacks at the same time, if they access data at different banks.

Because those accesses have to be done as fast as possible in order to leak as many charge as possible to other memory cells, it would be obstructive to share a logical CPU core[2] between multiple hammering threads.

For that reason, the number of hammering threads should be equal to the number of idling CPU cores. However, it should not be greater than the number of DRAM banks because, otherwise, multiple threads would access the same bank at the same time, and thereby interfere with each other.

So, the basic idea is to calculate a set of *hammer candidates* for each bank and distribute them to the hammering threads. Thereby, each bank is processed by exactly one thread and multiple banks are processed in parallel. This is implemented in HAMMERTOOL [12], [13]

B. Exploiting the `CMPSB` and `REPE` Instructions

As described in [13], the additional execution of the `CMPSB` and `REPE` instructions like they are used in the x86 implementation of the `memcmp()` function in the Linux Kernel depicted in Listing 1 leads to a significant increase in the number of bit flips found at a given memory area.

```
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: Source code of the function `memcmp` from the Linux Kernel 5.11.11 located in arch/x86/boot/string.c.

A short benchmark showed that this implementation is about 2.65 times as fast as the implementation for the ARM microarchitecture which uses two pointer that are incremented and compares the memory at the corresponding location. For the benchmark, I copied both implementations (for the x86 and the ARM microarchitecture) to a user space program and compared 1 000 000 pages with the same content (so the comparison does not exit before comparing all pages). Afterwards, I calculated the factor based on the measured times the comparison took for the different implementations.

Because I don't have access to the internal implementation of the CPU for that instruction, I can only assume that is uses optimized and/or prioritized memory accesses and, therefore, is significantly faster than the implementation on the ARM microarchitecture which does not provide those instructions.

A side effect of that other kind of memory accesses is that it increases the amount of bit flips found when scanning the same memory area significantly. FLIPTOOL[13][3] provides a user space implementation of those instructions.

V. Experimental Evaluation

In Section IV, it was claimed that the amplification primitives of parallel access and additional execution of instructions lead to a significant increase in the number of bit flips found in a given time. In this section, those statements are evaluated with measurements.

A. Experimental Setup and Methods

The experiments described in this paper were done on a ThinkPad X230T and a ThinkPad T540p with the configuration shown in Table I if not noted otherwise.

TABLE I: Hardware specification of our setup.

System	CPU	RAM	Memory capacity
ThinkPad X230T	i5-3320M	`M1`	4 GiB
ThinkPad T540p	i7-4800MQ	`M4`	4 GiB

At time of writing, both systems have the newest available BIOS version installed (2.75 (2019-10-04) on the X230T and 2.38 (2020-05-19) on the T540p). Both BIOS versions contain mitigations for the "risk of security vulnerability related to DRAM Row Hammering" [16], [17]. A short benchmark with HAMMERTOOL showed that both systems have a refresh rate of 32 ms instead of 64 ms, so the mitigation of a double refresh rate is used.

In the scope of this paper, multiple DIMMs were used. In order to distinguish them, they are labeled with references `M1` - `M6`. If not noted otherwise, the DIMM configuration depicted in Table I is used in the experiments.

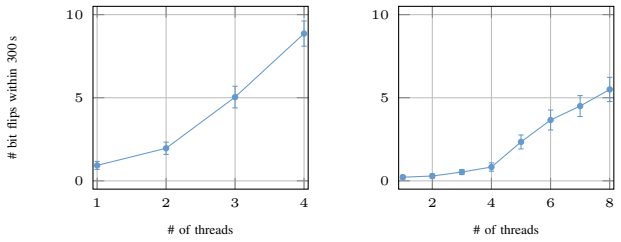
Because HAMMERTOOL does dynamic memory allocation at startup, the physical memory area that is scanned might change at each start. For this reason, most measurements were done 100 times and the average value as well as confidence intervals are depicted in most graphs.

[2]Logical CPU core include hyperthreading cores as well.

[3]formerly known as FLIPPER

B. Parallized Hammering

In Section IV-A, the approach of parallized hammering of multiple DRAM banks was explained. It was assumed that parallizing increases the number of bit flips depending on the number of logical CPU cores used for the execution of the rowhammer attack. Figure 1 depicts the amount of bit flips found within 300 s on the X230T and the T540p. Those measurements were done with a different amount of threads executing the rowhammer attack in parallel.



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

Fig. 1: Number of bit flips found within 300 s depending on the number of threads used by HAMMERTOOL. The average values of 100 measurements are depicted for each number of threads. The error bars show 99 % confidence intervals.

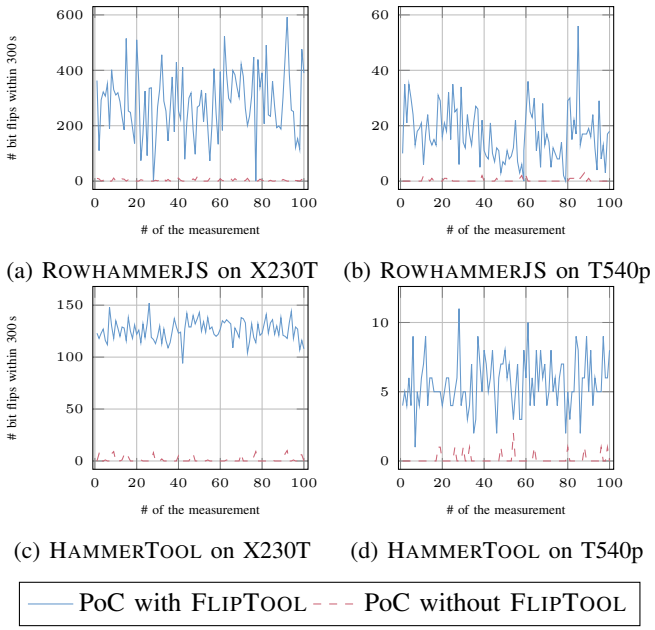
It should be noted that different DIMMs are differently susceptible to rowhammer, so the amount of bit flips found on the X230T is higher than on the T540p although the T540p has more logical CPU cores.

The graphs for both systems show a lower slope for $0 \leq x < \frac{n}{2}$ than for $\frac{n}{2} < x \leq n$. This can be explained with the behaviour of the scheduler: If a thread runs at a CPU core on full load (which is the case for the hammering threads), the scheduler tries to balance the load between the cores which leads to the threads "jumping" between the cores. This behaviour can be seen in a task manager like *htop*. If there are more than $\frac{n}{2}$ threads, they cannot change the core they are running on at the same time because there are more cores used than free. For this reason, some threads have to stay at the core they are running on which leads to more DRAM accesses in a given time and, thereby, increases the probability of bit flips.

C. Execution of additional Instructions

The approach of additional execution of x86 instructions was explained in Section IV-B. It was assumed that the execution of additional x86 instructions while hammering would lead to a significant increase in the number of bits flipping in the same memory area.

The results of the experiment are depicted in Figure 2. In the scope of this experiment, HAMMERTOOL and ROWHAMMERJS are used as Proof of Concept (PoC). The number of bit flips that occurred within 300 s using the corresponding PoC on the X230T and the T540p is measured 100 times. In order to analyze the effects of FLIPTOOL, the PoCs are started with and without FLIPTOOL. It is important to note that HAMMERTOOL runs in single-threaded mode for this experiment, so the hammering of the banks is *not* parallized.



(a) ROWHAMMERJS on X230T (b) ROWHAMMERJS on T540p

(c) HAMMERTOOL on X230T (d) HAMMERTOOL on T540p

Fig. 2: Number of bit flips measured within 300 s with and without FLIPTOOL running. In the measurements with FLIPTOOL, it was started as a process on the system. The measurements were started after FLIPTOOL was done allocating memory and started comparing. Mind the different scales of the y axes.

It can be noted that there is a significant increase in the number of bit flips when FLIPTOOL is running in addition to the PoCs. ROWHAMMERJS was used in this experiment in order to assure that the effect does not occur due some bug in the implementation of HAMMERTOOL or some side-effects that are implementation-dependent.

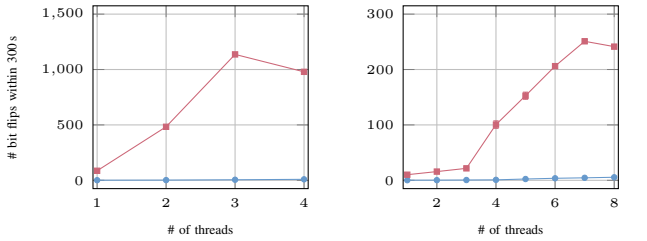
Another significant insight is that the amount of bit flips found with ROWHAMMERJS is about three times the amount of bit flips found with HAMMERTOOL on both systems with and without FLIPTOOL running. This effect can be explained with the implementation details of HAMMERTOOL: Due to the multithreading support (which is not used in this experiment) there is some overhead that leads to less DRAM accesses and, thereby, a lower amount of bit flips found in a given time.

D. Flipper: Combining both Attacks

In the last two sections it was shown that both attack primitives explained before lead to a significant increase in the amount of bit flips. However, now the question is if those attack primitives can be combined: Does the usage of FLIPTOOL increase the amount of bit flips found within 300 s while HAMMERTOOL is used in multithreading mode?

For this reason, the experiment described in Section V-B is repeated with FLIPTOOL running additionally. Figure 3 depicts the results of that experiment. The values from the

measurement without FLIPTOOL were taken from the experiment shown in Section V-B.



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

HAMMERTOOL without FLIPTOOL
HAMMERTOOL with FLIPTOOL

Fig. 3: Number of bit flips found by HAMMERTOOL in 300 s with and without FLIPTOOL. An average value from 100 measurements is depicted for each number of threads. The error bars show 99 % confidence intervals which are so small they are not visible. Mind the different scales of the y axes.

The result shows that the usage of FLIPTOOL increases the number of bit flips significantly. This is also the case when HAMMERTOOL is running with multiple hammer threads. Thereby, the graphs look similar to the graphs shown in Section V-B. However, there are some notable differences:

- The number of bit flips increases until the number of threads is equal to the number of logical CPU cores minus one. It decreases again for a number of threads equal to the number of CPU cores.
- The change of slope happens not at $\frac{n}{2}$ but on $\frac{n}{2} - 1$, so the slope in the range $0 \leq x < \frac{n}{2} - 1$ is less than the slope for $\frac{n}{2} - 1 < x \leq n - 1$

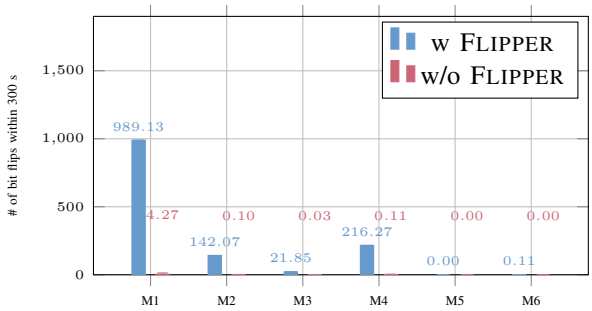
The reason for those differences is that the additional execution of FLIPTOOL brings one CPU core to full load. Because of this, there are $n - 1$ cores "free", and not n as in the experiment described in Section V-B.

E. Flipper: Real-World impact

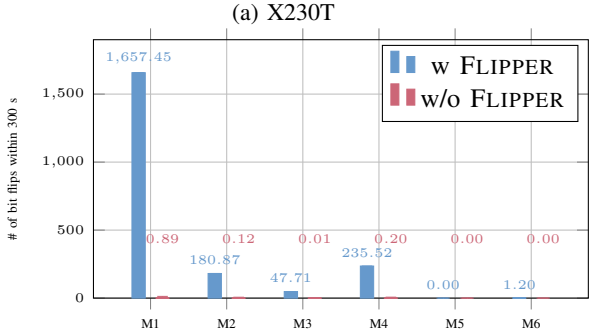
The last experiments demonstrated that there is a significant increase in the amount of bit flips when both attack primitives, the parallel execution of the rowhammer attack and the additional execution of x86 instructions, are used separate as well as in combination.

However, all previous experiments were done on two systems with two different DIMMs. For this reason, is is not possible to compare the systems directly. In this experiment, the amount of bit flips found within 300 s with and without FLIPPER (multithreading and FLIPTOOL) is measured with six different DIMMs on both systems. Figure 4 depicts the result of those measurements.

On the X230T, the maximal factor is 1966 with DIMM `M4`. The minimal factor is 231 with DIMM `M1`. There occurred no bit flips with `M5` either with or without FLIPPER. With `M6`, an



(a) X230T



(b) T540p

Fig. 4: Comparison of the number of bit flips with and without FLIPPER for six different DDR3 DRAM modules. The error bars shows 99 % confidence intervals.

average of 0.11 bit flips were found within 300 s while there were no bit flips in that time without FLIPPER.

On the T540p, the maximal factor is 4771 with DIMM `M3`. The minimal factor is 1177 with DIMM `M4`. Like on the X230T, there occurred not bit flips with `M5` either with or without FLIPPER. With `M6`, an average of 1.2 bit flips were found within 300 s while there were no bit flips in that time without FLIPPER.

In general, the amplification factors when FLIPPER is used tend to be higher on the T540p than on the X230T. This can be explained with the higher amount of CPU cores in the T540p. Thereby, it is possible to scan more memory areas at the same time (and within the 300 s of the measurement).

This experiment demonstrates that the usage of FLIPPER leads to a significant increase in the amount of bit flips found in a given time. As stated before, ROWHAMMERJS finds about three times as much bit flips than HAMMERTOOL does. When FLIPPER is used, the amount of bit flips found in a given time compared to the plain ROWHAMMERJS PoC is in the range of hundredfold.

VI. RELATED WORK

In 2014, Kim *et al.* [14] did an experimental study on DRAM disturbance errors using an Field-Programmable Gate Array (FPGA) based memory controller. The result was that 110 of the 129 tested DDR3 DIMMs were vulnerable to rowhammer.

One year later, Seaborn and Dullien [3] released an exploit that uses rowhammer to gain kernel privileges locally on a

Linux-based system. They developed a tool for automatic rowhammer testing as well [18].

In the same year, Gruss, Maurice, and Mangard [7] introduced an approach to exploit rowhammer in javascript within a browser. Because browsers do not provide the `CLFLUSH` instruction, the exploit works without using that instruction by evicting the corresponding data from the cache. They published their PoC [19] as well. In addition to the eviction-based approach, their PoC supports a "`CLFLUSH` mode" as well which is used in some measurements of this paper.

In 2020, Frigo *et al.* [5] demonstrated that the TRR mitigation on DDR4 systems are not as effective as assumed. In the experiment they published, 13 out of 42 DDR4 DIMMs were vulnerable to their new, TRR aware techniques.

One year later, I wrote my practical thesis [12] and bachelor thesis [13] In the scope of the practical thesis, HAMMERTOOL, which is used as PoC for the evaluation in this paper, was developed. During the bachelor thesis, an evaluation of HAMMERTOOL was already done in terms of time required to reverse-engineer address functions and number of bit flips found in a given time. In addition to that, FLIPTOOL[4] was introduced in the bachelor thesis. However, the experiments described in this paper are new and extend the experiments from that thesis.

In the same year, Ridder *et al.* [8] showed that the approaches to bypass TRR can be used from javascript within browsers as well.

VII. CONCLUSION

In this paper, the two basic attacks of FLIPPER (parallized hammering and additional execition of x86 instructions) were evaluated. It was shown that both of them, when executed separately, lead to a significant increase in the number of bit flips. Furthermore, a combination of both basic attacks results in an increase in the amount of bit flips found in a given time by about hundredfold.

Because both systems used for evaluation feature the mitigation of a double refresh rate, the increase in the amount of bit flips found in a given time can be used to bypass the mitigation that reduces the probability of bit flips.

REFERENCES

[1] M. Heckel and F. Adamsky, "Flipper: Rowhammer on steroids — exploiting the x86 cmpsb instruction and parallelising hammering," in *26th European Symposium on Research in Computer Security (ESORICS) 2021*, (submitted), 2021.

[2] "26th european symposium on research in computer security (esorics) 2021." (2021), [Online]. Available: https://esorics2021.athene-center.de/ (visited on 06/02/2021).

[3] M. Seaborn and T. Dullien. "Exploiting the DRAM rowhammer bug to gain kernel privileges." (2015), [Online]. Available: https://www.cs.umd.edu/class/fall2019/cmsc818O/papers/rowhammer-kernel.pdf (visited on 11/16/2020).

[4] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *Proceedings of the 23rd ACM Conference on Computer and Communication Security (CCS)*, Oct. 2016. [Online]. Available: https://vvdveen.com/publications/drammer.pdf.

[5] P. Frigo, E. Vannacci, H. Hassan, V. van der Veen, O. Mutlu, C. Giuffrida, H. Bos, and K. Razavi, "TRRespass: Exploiting the Many Sides of Target Row Refresh," in *Proceedings of the IEEE Security & Privacy*, May 2020. [Online]. Available: https://download.vusec.net/papers/trrespass_sp20.pdf.

[6] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a Needle in the Software Stack," in *Proceedings of the 25th USENIX Security Symposium*, Jun. 2016. [Online]. Available: https://download.vusec.net/papers/flip-feng-shui_sec16.pdf.

[7] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript," in *Proceedings of the 13th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2016.

[8] F. de Ridder, P. Frigo, E. Vannacci, H. Bos, C. Giuffrida, and K. Razavi, "SMASH: Synchronized Many-sided Rowhammer Attacks From JavaScript," in *Proceedings of the 30th USENIX Security Symposium*, Aug. 2021. [Online]. Available: https://download.vusec.net/papers/smash_sec21.pdf.

[9] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer Attacks over the Network and Defenses," in *Proceedings of the 27th USENIX Security Symposium*, Pwnie Award Nomination for Most Innovative Research, Jul. 2018. [Online]. Available: https://download.vusec.net/papers/throwhammer_atc18.pdf.

[10] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice, and D. Gruss, "Nethammer: Inducing rowhammer faults through network requests," Sep. 2020. [Online]. Available: https://hal.inria.fr/hal-01872588/document.

[11] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another Flip in the Wall of Rowhammer Defenses," *CoRR*, vol. abs/1710.00551, 2017. arXiv: 1710.00551. [Online]. Available: http://arxiv.org/abs/1710.00551.

[12] M. Heckel, "HAMMERKIT: Toolset for memory inspection and automatic rowhammer detection," 2021.

[13] ——, "RAEAX: Rowhammer Amplification by Execution of Additional X86 instructions," 2021.

[4] formerly known as FLIPPER

[14] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory without Accessing Them: An Experimental Study of DRAM Disturbance Errors," *ACM SIGARCH Computuer Architure News*, vol. 42, no. 3, pp. 361–372, Jun. 2014, ISSN: 0163-5964. DOI: 10.1145/2678373.2665726.

[15] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks," in *S&P*, Best Practical Paper Award, Pwnie Award Nomination for Most Innovative Research, May 2019. [Online]. Available: https://download.vusec.net/papers/eccploit_sp19.pdf.

[16] "BIOS update utility README (X230T)." (2015), [Online]. Available: https://download.lenovo.com/pccbbs/mobiles/gcuj22us.txt (visited on 11/16/2020).

[17] "BIOS update utility README (T540p)." (2015), [Online]. Available: https://download.lenovo.com/pccbbs/mobiles/gmuj16us.txt (visited on 11/16/2020).

[18] M. Seaborn and T. Dullien. "Program for testing for the DRAM "rowhammer" problem." `commit c1d2bd9`, Google. (2015), [Online]. Available: https://github.com/google/rowhammer-test (visited on 12/04/2020).

[19] "Program for testing for the DRAM 'rowhammer' problem using eviction." `commit 62c9f31`, IAIK. (2017), [Online]. Available: https://github.com/IAIK/rowhammerjs (visited on 12/04/2020).