Reviewing the Practicability of Rowhammer Attacks on Commodity Hardware in Real World Attack Scenarios

S. Selviyani, ZHAW

***Abstract*—The Rowhammer vulnerability is a hardware exploit in Dynamic Random Access Memory (DRAM) that allows an attacker to trigger bit flips in neighboring memory through rapid access patterns. Vice Motherboard says [1], "It's like forcibly breaking an apartment by repeatedly slamming on the neighbor's door until the vibrations break the desired door". Despite intensive academic investigation since its discovery in 2014, the real-world practicality and prevalence of Rowhammer still remain unknown. This paper is reviewing the practicability of Rowhammer on commodity hardware, using an experimental framework from the FLIPPYR.AM project to analyze device vulnerabilities.**

***Index Terms*—DRAM, Rowhammer, Flippyr.am**

I. Introduction

The ancestor of Rowhammer can be traced back to at least 1970, with the literature from that time discussing the first DRAM IC. However, the first successful attack was introduced in 2014, within the research title of "*Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors*" by Kim et al. It revealed a previously unknown successful attack that exploits the DRAM fault by accessing the DRAM at a highly intensive rate. Kim et al. conclude that they have demonstrated, characterized, and evaluated the phenomena of disturbance errors in DRAM chips. This error occurs when repetitive access to a DRAM row, corrupts data stored in succeeding rows [2]. From their research, Rowhammer was introduced, and the continued exploration of Rowhammer has been going on for ten years now.

In one of the talks from the 32nd Chaos Communication Congress, Daniel Gruss and Clémentine Maurice demonstrated the *Rowhammer.js* exploit that can run the root privileges from web apps. It was one of the successful exploits of Rowhammer at that time [3]. The Rowhammer effect can be abused by causing random bit flips in memory that is part of the attacker's process, which allows for potential security breaches such as data damage, system instability, and unauthorized memory access.

Despite Rowhammer's potential risks, no serious occurrence has occurred that raises concerns. However, the Rowhammer problem has received substantial attention, with over 500 papers published each year, including mitigation publications, demonstrating its importance in the realm of security. The Rowhammer is significant due to its potential impact on reliability and security, since it can induce system crashes and facilitate privilege escalation assaults, making it a critical issue [3]. Here, we focus on reviewing Rowhammer attacks on commodity hardware and participation in the Flippy.ram project to study further the implications of Rowhammer on the DRAM [4].

II. Technical Background

A. Dynamic Random Access Memory (DRAM)

Dynamic Random Access Memory (DRAM) is a type of Random Access Memory where each bit of data is stored as a charge in a capacitor. Each capacitor is paired with an access transistor, and together they form a DRAM cell. These cells are organized in rows and columns. The access transistor is connected to the wordline (which selects the row), and when the wordline is activated, the transistor connects the capacitor to the bitline (which corresponds to the column), allowing data to be read from or written to the cell [5]. A row buffer in DRAM has the temporary task of storing the contents of an entire row after it has been activated, which allows for subsequent reads or writes to that same row. At any given time, only one row can be held in the row buffer.

DRAM can be divided systematically into Channels, Ranks, and Banks 2. The physical connection between the memory controller and the DRAM module is referred to as a *channel*. Inside the channel, the physical memory modules connected to the motherboard are referred to as Dual Inline

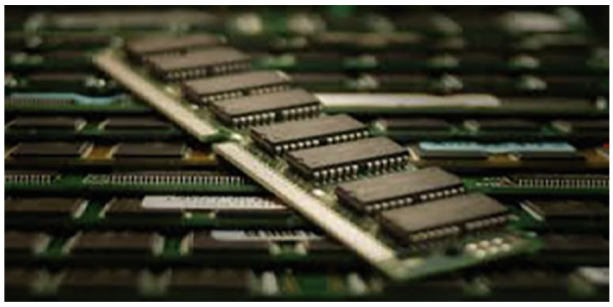


Fig. 1: A typical DRAM module in the computer [5]

Memory Modules (DIMMs), which typically consist of one or two ranks, as shown in the Figure 1. Each rank consists of multiple banks, for instance, DDR3 contains 8 banks, while DDR4 comprises a total of 16 banks organized into 4 bank groups, each containing 4 banks. In contrast, DDR5 features a total of 32 banks organized into 8 bank groups, with 4 banks per group. An illustration of DRAM organization can be seen in Figure 3.

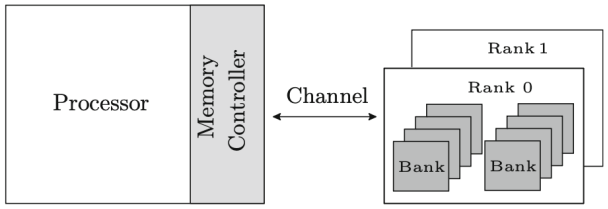


Fig. 2: The DRAM architecture with Channel, Rank, and Banks [5]

The capacitors in each DRAM cell gradually lose their charge over time. The duration for which a cell can reliably hold its charge before requiring a refresh is called *the retention time*, which is guaranteed to be at least 64 ms in the DDR3 DRAM specification. However, the repetition of row activation over a period of time can lead to faster discharge of cells in the adjacent rows.

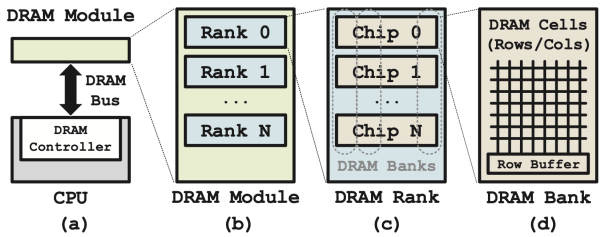


Fig. 3: A typical DRAM organization [6]

DRAM reading works by measuring the charge quantity stored in the capacitors of memory cells. It is divided into three phases [6]:

1) **Activation (ACT)**: When a row within a DRAM bank is activated, the charge stored in its cells is sensed and transferred to the row buffer. The row buffer holds the entire contents of the activated row, enabling faster access to data from that row without requiring activation. Subsequent data reads access the row buffer directly until the row is precharged.
2) **Read (RD)**: The data of the row buffer at a specified row address are sent to the DRAM controller.
3) **Pre-charge (PRE)**: The bank prepares itself for the next access by disconnecting and emptying the active row buffer. All rows in a bank use only one row buffer, and the DRAM controller activates and reads data from a single row at a time within a bank. In addition, write operations works similarly to read operations.
4) **Refresh (REF)**: DRAM cells are vulnerable to electrical leakage, which may lead to data loss if the cells are not accessed at a sufficient frequency.

B. What is Rowhammer

Rowhammer is a computer hardware vulnerability that exploits an unintended side effect in dynamic random-access memory (DRAM), where electrical interactions between memory cells result in charge leakage, potentially altering the contents of adjacent memory rows that were not directly accessed [7]. The breach of isolation among DRAM memory cells comes from the increased cell density in modern DRAM and can be triggered by specially crafted memory access patterns that rapidly activate the same memory rows several times [7].

To exploit the Rowhammer vulnerability, the memory accesses should be [8]:

- **Uncached**. In that means every memory access must bypass the CPU cache to ensure they directly reach DRAM.
- **Fast**. We want to optimize the number of accesses between row refreshes, as we face off against the next row's refresh.
- **Targeted**. We must access two particular rows to achieve the bit flips in the middle row.

The CPU cache is located between the CPU core and the DRAM. Hence, only non-cached accesses directly access to the DRAM. We have two techniques are available: First, flush the cache after accessing the data, or second, prevent the data from

entering the cache in the first place. There are four access methods to ensure that the next access is taken straight from DRAM:

1) `clflush` instruction (x86) proposed by Kim et al. in 2014. [2]
2) `Cache eviction` proposed by Awake et. al in 2016 [9]
3) `Non-temporal accesses` introduced by Qiao et al. in 2016 [10]
4) `Uncached memory` introduced by Veen et al. in 2016 [11]

In the first access method, we begin by accessing the target data, flushing it from the cache using the `clflush` instruction, and then repeatedly reloading and flushing the data in a loop to induce bit flips.

```
hammering:
  mov (X), %eax //Read from addr X
  mov (Y), %ebx //Read from addr Y
  clflush (X) //Flush cache for addr X
  clflush (Y) //Flush cache for addr Y
  jmp hammer //Loop indefinitely
```

For the hammering method to be effective and induce bit flips, addresses X and Y must map to different rows within the same DRAM bank.

In the second access method, by Aweke et al., demonstrate that an attacker may force the cache to invalidate its contents by accessing memory addresses inside the same cache eviction set. Current CPUs typically use highly associative last-level caches (e.g., 8-way or 16-way). Therefore, many memory visits are necessary to evict a cache block, which slows down the Rowhammering process. In this method, the cache replacement policy matters (for instance, current caches typically do not employ Least Recently Used) [9].

In the third access approach by Qiao et al, they perform *non-temporal accesses*. That means, the data is accessed once and not again in the near future. While the data from non-temporal accesses could pollute the cache, the non-temporal accesses instructions are used to bypass the cache and minimize this pollution. Non-temporal store instructions treat the target memory as uncached *"write combining (WC)"* type. All non-temporal access store to a single address are consolidate in a WC buffer, which reduces the write rate to DRAM, making it insufficient for Rowhammering. To ensure each non-temporal store reaches the DRAM chip, the method by Qiao et al. involves executing a cached access to the same address following the non-temporal store [10].

C. Exploitation Techniques of Rowhammer

Rowhammer exploitation techniques are classified into two categories. The first category includes fundamental exploitation techniques like single-sided hammering, double-sided hammering, multi-sided hammering, memory massaging, probabilistic bit flip exploitation, and access pattern optimization. The second category includes advanced exploitation techniques, like browser-based attack, remote exploits, privilege escalation attacks, cross-virtual-machine attacks and Flip Feng Shui attacks. As previously mentioned, we will not explain all of the experimental techniques in this chapter. We will provide an overview of selected items.

The *single-sided attack* focuses on a single row physically close to the target victim row. However, when the memory controller utilizes an open-page policy, it requires accessing two separate rows inside the same bank to clear the contents of the row buffer. This discrepancy arises despite the name *"single-sided attack"*, suggesting the targeting of only a single memory location. To address this issue, a new technique named one-location hammering was introduced [12]. This technique involves an attacker executing a `Flush+Reload loop` exclusively on a single memory address, operating at the maximum possible frequency. This continuous activity effectively reopens the same DRAM row every time the memory controller attempts to close it. Since one-location hammering does not access different rows within the same bank, it can bypass certain existing defense mechanisms designed to detect the original *single-sided attack*. In a nutshell, one-location hammering bypasses defense mechanisms by repeatedly reopening the same DRAM row, rather than accessing multiple rows [12].

The *double-sided attack* involves targeting two memory rows wrapping the targeted victim row. This technique can trigger an increased amount of bit flips compared to the single-sided attack. Nevertheless, it demands a more extensive understanding of virtual-to-physical mappings. The two rows to be hammered must be strategically positioned on opposite sides of the target row. Kim et al. indicate that the double-sided attack can be effectively executed to induce successful bit flips via Rowhammer [12]. However, this method requires an in-depth knowledge of virtual-to-physical mappings. See figure 4.

Flip Feng Shui (FFS) is an exploitation vector that allows an attacker to induce bit flips over arbitrary physical memory in a controlled way [13]. FFS relies on a physical memory massaging primitive and a hardware vulnerability to flip bits at specific

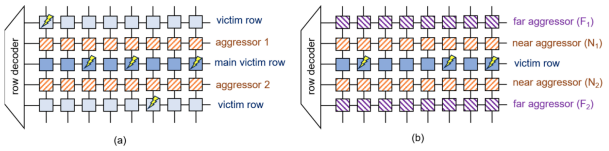


Fig. 4: Illustration of (a) double-sided attack and (b) single sided attack [12]

locations in physical memory. FFS uses memory massaging to map a desired victim's physical memory page into the attacker's own. The attacker needs to probe for matching templates by repeatedly exploiting the hardware vulnerability to surgically target the victim's sensitive data. The end-to-end attack allows the attacker to flip a desired bit in any data in the software stack. FFS needs no software vulnerability and allows the attacker to rewrite arbitrary physical memory on the operating system. FFS transforms a fundamental hardware weakness into a serious software-like vulnerability through memory massaging and exploitation [13].

III. Evaluating the Rowhammer Attack (FLIPPYR.AM Project)

A. Experimental Setup

In this section, we will elaborate on our experimental setup of the Rowhammer Attack, which was inspired by a talk at the 38th Chaos Communication Congress 2024 in Hamburg. The presentation was given by three cybersecurity researchers: Daniel Gruss, Martin Heckel, and Florian Adamsky. The title of the presentation is "*Ten Years of Rowhammer: A Retrospective (and Path to the Future)*". [14]

The researchers in the talk suggested an extensive review and analysis of the available research on Rowhammer, using the scientific method to determine its actual risk and prevalence. An initial review of the existing literature on Rowhammer shows a noticeably limited sample size, mainly consisting of single-sample studies, with most of it focusing on DDR3 and DDR4 memory, while few examine other memory types. As a result, for a better understanding of Rowhammer's prevalence, further research is essential, and a study with an enlarged sample size should be conducted, executing Rowhammer tests on many different devices. Therefore, this research seeks to contribute to this broad study of Rowhammer by evaluating various of our devices to determine how susceptible they are to Rowhammer attacks.

The goals of the study are to determine the actual conditions of real systems and identify which systems are vulnerable to Rowhammer, it requiring a large-scale prevalence observation to identify certain exploits that are more successful on specific systems. To achieve that goal, a testing setup has been developed, consisting of a standardized Linux system that includes a set of tools capable of evaluating and discovering DRAM addressing functions, as well as running Rowhammer tests on the system without running any actual attacks or exploits. The test system developed by the FLIPPYR.AM project is based on Arch Linux and can be booted from a USB drive without modification to the software on the hard drive.



Fig. 5: FLIPPYR.AM USB drive

Here are all the steps of the Flippyr.am experiments that we do in our test devices.

1) First of all, we plug in the Flippyr.am USB drive (like on figure 5) into the device. We also need to disable secure boot of our device since this is a custom Linux system. Otherwise, the test will not run on our device. Then, we restart the systems until they show the boot options. On HP laptops, we can press the `F9` key to show the boot menu option.
2) Second step, in the boot menu, it shows the USB Disk 2.0 option from the USB stick that we plugged in. We selected it. After loading, it will show some options, such as `Rowhammer Test System`, `EFI Shell`, and `Reboot Into Firmware Interface`. We select the `Rowhammer Test System`. After that, we have to wait for the loading of the programs from the USB drive into our memory. It takes several minutes.

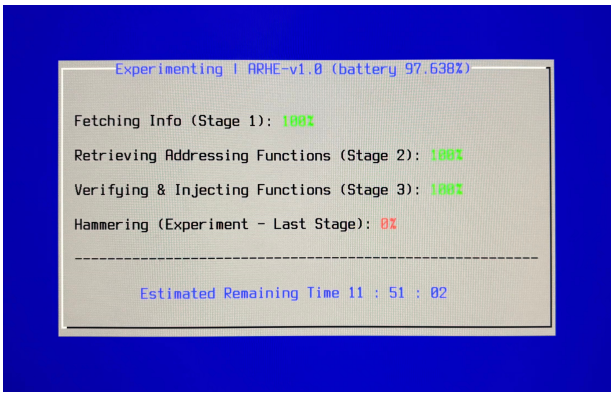


Fig. 6: Preparation and hammering experiments

3) After the loading finished, it shows the disclaimer of the experiment. We can read through all the contents before selecting `Accept`. Next, we need to set up the experiment duration time. Here, we want to do the experiment for 12 hours. After that, we select `OK`. The display will show like this figure 6.
4) As it shows, there are some steps before doing the hammering. The first stage is to get all the information about the memory hardware, the second stage is to retrieve the addressing functions, and the last stage is to verify the injection functions. All of these stages determine the tests' outcome, whether it was skipped, successful, or failed, like we show in the table list on I.
5) We need to wait for a couple of hours until the hammering experiments are finished. We also need to make sure that our device is plugged into their charger.
6) After the experiment is done, it will show the different test results and bit flips like in this figure 7.
7) Next, we select the `Continue` button. It asks us to willingly participate in the Flippyr.am study. We can answer `Yes` or `No`. In here, we want to participate in it, so we select `Yes`. After compressing the result, the display shows the question `Are you willing to upload the result now?`. Here we also want to select `Yes`. In addition, when we upload the result, we need to connect to the Internet. We can configure the Internet connection by entering our WLAN password or connecting an Ethernet cable. Then, it uploads the result to their server now.
8) When the upload is done, it displays a QR code to access our token on their flippyr.am website. We can also use the token to participate in their raffle.
9) Last, the experiment is completed and now we can reboot our device. In the meantime, we can plug out the USB drive from our device.

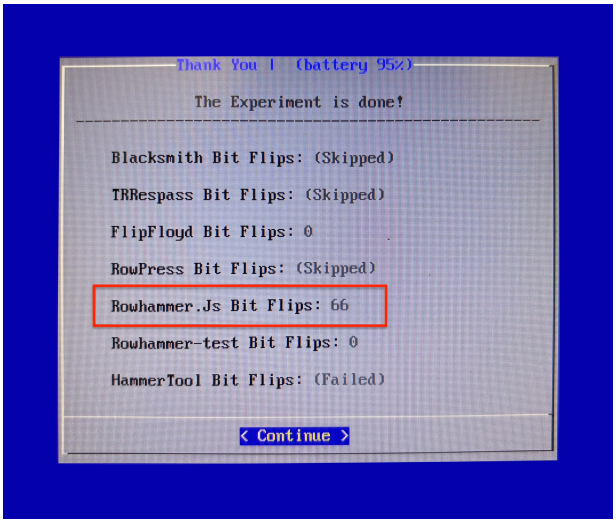


Fig. 7: Results of hammering

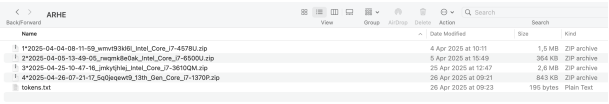


Fig. 8: All the data result from the experiment

B. Result of Experiment

On the USB drive, we can check the `ARHE` directory. This folder contains all the device logs from our tests and tokens that we got from the experiment upload, such as the figure 8.

When we extract one of the compressed reports, it will show all the results of the experiment, together with all the log files that are in plain text format like in figure 9.

In the Table I we list all the tested devices with their device brand, CPU models and memory manufacturers.

IV. Impact on Real-World Devices

A. Unauthorized Memory Access

Rowhammer attacks exploit a weakness in DRAM chips, potentially resulting in data corruption, system crashes, and, most critically, unauthorized memory access. The impact of data corruption is data corruption that can lead to corrupted files,

Device Number	1	2	3	4
Test Date	3 Apr 2025 19:09	5 Apr 2025 13:49	24 Apr 2025 10:47	26 Apr 2025 21:17
Device	MacBook Pro Retina 13″	HP Elitebook 850 G3	ThinkPad W530	ThinkPad X1 Carbon Gen 11
Device Year	2014	2016	2012	2023
CPU	i7-4578U	i7-6500U	i7-3610QM	i7-1370P (13th Gen)
Memory	2×8 GB DDR3	1×8 GB DDR4	1×4 GB DDR3	8×8 GB LPDDR5
Memory Manufacturer	SK Hynix	Samsung	Hynix/Hyundai	Micron Technology
Blacksmith Bit Flips	Skipped	0	Skipped	Skipped
TRRespass Bit Flips	Failed	0	Skipped	Skipped
FlipFloyd Bit Flips	0	0	0	0
RowPress Bit Flips	Skipped	0	Skipped	Skipped
Rowhammer.js Bit Flips	0	0	**66**	Skipped
Rowhammer-test Bit Flips	0	0	0	0
HammerTool Bit Flips	Failed	Failed	Failed	Skipped

TABLE I: List of Tested Device and Results

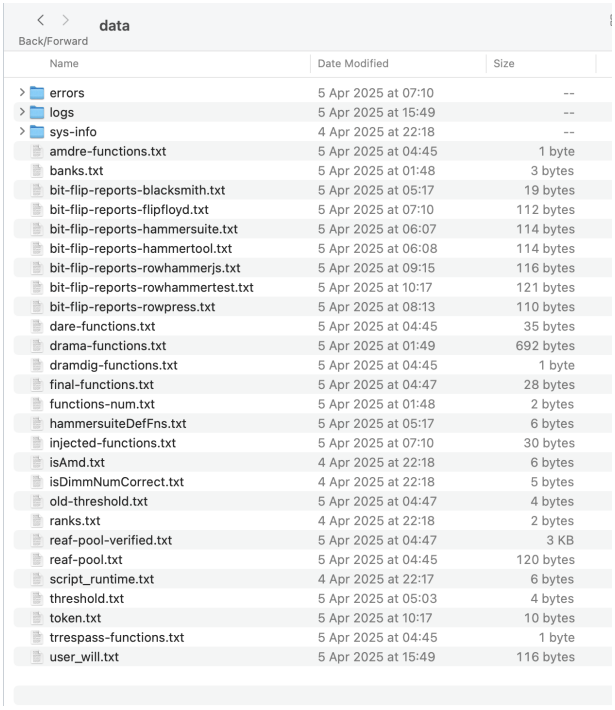


Fig. 9: Result of experiments and logs

application crashes, or even system-level vulnerabilities. The most serious aspect of the Rowhammer attack is illegal memory access by inducing bit flips in metadata such as stored in memory page tables. Rowhammer may allow the attacker to obtain confidential information or run harmful code with escalated privileges. This section will look at previous studies from other researchers as examples of Rowhammer exploitation.

Browser-Based Attacks (Rowhammer.js). Reference by Gruss et al. in the research on developing a remote software-induced fault attack in JavaScript, called Rowhammer.js. This attack can trigger hardware malfunctions in DRAM modules without requiring physical access or specialized machine instructions. The Rowhammer attack targets memory modules by rapidly visiting other memory locations, resulting in parasitic effects that change the content of a memory cell without direct access [15]. This indicates a critical threat to system security, including privilege escalation threats. Gruss et al. illustrate a completely automated attack that simply needs a webpage with JavaScript to induce errors in remote hardware. This allows broad access to the systems of visitors to a malicious website and shows that the attack works on standard systems, with known countermeasures proving ineffective against Rowhammer attacks. [15]

The attack evaluation indicates that the `clflush` instruction produces the most significant number of bit flips, and that both native code eviction and JavaScript-based attacks may trigger bit flips. However, the refresh interval must be appropriately configured to cause bit flips, dependent on the specific DIMM for correct selection of the refresh interval. The probability of bit flips in JavaScript is slightly reduced when compared to native code, however, if a computer is affected by native code execution, it is probably also vulnerable to Rowhammer.js. [15] As a result, the vulnerability of a system to Rowhammer-based attacks is dependent on the refresh interval chosen by the DIMM (Dual Inline

Memory Module) and Gruss et al. successfully caused bit flips in the system using the default settings and BIOS version, using a function reverse engineered [15].

Remote Exploits. In discussing remote vulnerabilities, we will reference the research titled "*Throwhammer: Rowhammer Attacks across the Network and Defense*" by Tatar et al. This research, published in 2018 [16], has demonstrated the presence of another exploit related to Rowhammer at the operating system level. This showed a new threat to system security, when an attacker can start and exploit Rowhammer bit flips remotely by sending network packets to a victim machine connected with RDMA (Remote Direct Memory Access)-enabled network devices, which contradicts the common assumption that Rowhammer attacks demand code execution on the victim machine. The research shows the possibility of performing a Rowhammer attack remotely using a standard 10 Gbps network, disproving the original assumption that these attacks, which require flipping bits in DRAM, cannot be conducted across networks.

The threat model involves an attacker sending real traffic to a target server, with the goal of bypassing RDMA's security by modifying bits outside registered DMA areas. A tool for testing this research, titled "*Throwhammer*" was developed to examine this potential vulnerability. It uses RDMA capabilities for efficient packet transmission and contains a server and client process. The research demonstrates that *Throwhammer* may trigger bit flips in a 10 Gbps network, using two machines that have 8-core Haswell i7-4709 CPUs connected with Mellanox ConnectX-4 single-port NICs as their evaluation lab [16]. This research concludes that Tatar et al. showed how an attacker can use remote bit flips in `memcached` to illustrate a remote Rowhammer attack, and they proposed an innovative security method that physically separates the RDMA buffers from the rest of the system [16].

Privilege Escalation. Kurmus et al. from Zürich IBM Research stated in their paper *"From random block corruption to privilege escalation: A filesystem attack vector for rowhammer-like attacks"* that a filesystem attack vector for rowhammer-like attacks, potentially resulting in privilege escalation, is predicated on the research of Cai et al. [17], who suggested that similar attacks could be executed on `MLC NAND` flash. The authors demonstrate that a filesystem-based attack vector can facilitate privilege escalation through random corruption of a strategically chosen block [18].

The attack exploits a vulnerability in the foundational hardware, particularly in *MLC NAND* flash-based SSDs, which can facilitate the corruption of sensitive data. Kurmus et al. illustrate that this issue can facilitate a local privilege escalation attack based on possible assumptions regarding the behavior of the flash device and the utilized filesystem. They also highlight that the attack primitive based on the vulnerabilities of `MLC NAND` flash is characterized by coarse granularity corruption, that the attacker can corrupt only one block of data [18]. In contrast, Rowhammer allows the attacker to flip an individual bit. Nevertheless, their research illustrates that this less potent attack primitive is adequate for executing a local privilege escalation attack. Kurmus et al. also present specific instances of hardware-based attacks, including voltage probing, silicon chip layer scraping, coolant spray and cold boot attacks, and timing side-channel attacks. It also highlights known system integrity attacks, such as the Rowhammer and DRAMMER attacks, both of which exploit the Rowhammer memory write primitive to execute privilege escalation on Android [18].

Cloud and Cross-Virtual-Machine Attacks. The paper titled *"Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers"* by Cojocar et al. presents an extensive approach for assessing the vulnerability of cloud servers to Rowhammer attacks, which could compromise the security of cloud providers and their customers by exploiting memory disturbance errors in DRAM [6]. Cojocar et al. observe that cloud providers lack a systematic approach for determining the vulnerability of the DRAM in their servers to Rowhammer attacks. Furthermore, previous proof-of-concept attacks used varied methodologies without thoroughly evaluating their effectiveness, making it difficult to distinguish between Rowhammer-safe memory and a sub-optimal testing methodology. They also developed an approach that involves constructing worst-case testing conditions for DRAM and applying it to three categories of servers from a major cloud provider, showing that none of the CPU instruction sequences used in previous research to execute Rowhammer attacks create worst-case DRAM testing conditions. [6]

Cloud and Cross-Virtual-Machine Attacks emphasized the importance of mitigating the risks associated with Rowhammer attacks, which are easily executable and scalable, affecting all varieties of DRAM, including DDR3, DDR4, and even ECC-equipped DRAM. These attacks may harm security-critical bits, such as operating system page table entries, browser isolation sandboxes, and encryption

keys. Cojocar et al. used a conventional Rowhammer instruction sequence, similar to those used by other authors in their Rowhammer studies, as illustrated in this figure 10. This sequence includes instructions, `clflush` instructions, and a global memory barrier, similar to sequences found in other Rowhammer research.

```
loop:
  movzx rax, BYTE PTR [rcx]
  movzx rax, BYTE PTR [rdx]
  clflush BYTE PTR [rcx]
  clflush BYTE PTR [rdx]
  mfence
  jmp loop
```

Fig. 10: Typical Rowhammer instruction sequence [6]

In conclusion, based on their paper, they conclude that Rowhammer constitutes an important possible threat to cloud providers, while the full scope of the actual threat itself still remains unclear.

Mobile Platform Attacks. Another example of Rowhammer exploitation is attacks on mobile platforms. This paper will reference the research titled "*DRAMMER: Deterministic Rowhammer Attacks on Mobile Platforms*" by Victor et al. [11]. On this mobile platform attack, they introduce a deterministic Rowhammer attack named "*DRAMMER*", capable of compromising a system without dependence on specialized memory management functionalities. Victor et al. claim that Rowhammer uses the predictable memory reuse patterns of conventional physical memory allocators to execute a deterministic attack, consequently making it possible on commodity mobile platforms, such as Android/ARM, to potentially experience a Rowhammer attack. [11]

In their paper, they also demonstrate the practicability of their attack by implementing *DRAMMER* on Android/ARM and showing the first Rowhammer-based Android root exploit that relies on no software vulnerability and requires no user permissions. Conversely, the traditional x86-based Rowhammer exploitation techniques are found to be ineffective on mobile platforms, and Victor et al. also address the resulting challenges towards practical mobile Rowhammer attacks [11]. The research concludes that they effectively illustrate the feasibility of effective deterministic Rowhammer attacks capable of gaining an attacker root privileges on a system, using only the always-on features provided by common operating systems [11]. Similarly, they also demonstrate an execution of the *DRAMMER* attack on the Android/ARM platform. Victor et al. stated in their investigation that deterministic Rowhammer attacks pose an important risk to billions of mobile users, and their research is the first attempt to prove that Rowhammer is possible and continually exploitable on systems outside x86 with a restricted software feature set. [11]

B. Attacks on Cryptography

John D. Cook, on his website [19], clarifies that encryption can be hacked by a straightforward method by making public keys vulnerable to factorization via Rowhammer attacks on DRAM. He explains the well-known principle that the security of RSA encryption is dependent on the difficulty of factoring the product of two substantial prime numbers, allowing for the release of the product with little fear of its factorization, especially in the case of big primes, such as those comprising 2048 or more bits [19].

The Rowhammer attack weakens security by causing bit flips in DRAM memory, which could change the public key. This approach does not involve traditional encryption breaches. Instead, it secretly modifies encryption to a vulnerable system.

The impact of bit flipping can be illustrated by the products of two prime numbers, $p = 251$ and $q = 643$, leading to the product $n = 161393$. By modifying a single bit, $n = 161393$ is transformed into $m = 161392$. While n is difficult to factor, m is easily factorable due to its smaller prime factors:

$$n = 161393 = 251 \cdot 643$$
$$m = 161392 = 2^4 \cdot 7 \cdot 11 \cdot 131$$

John D. Cook generated two 100-bit random prime numbers in Mathematica Wolfram Alpha as an extensive example. Let's say:

$$p = 1078376712338123201911958185123 \text{ and}$$
$$q = 1126171711601272883728179081277$$

An extensive example presented above demonstrates that generating two 100-bit random primes in Wolfram Mathematica and modifying the least significant bit of their product significantly facilitates factorization of n, decreasing the factoring time from almost 100 seconds to one-third of a second.

Further experiments that John D. Cook also provides, shows that flipping any bit of the product of two large primes can make the factoring problem easier, with the median time of factoring being 0.4 seconds when one bit is flipped, and the difficulty depending on the number and the bit flipped. Like shows in the graphic on figure 11.

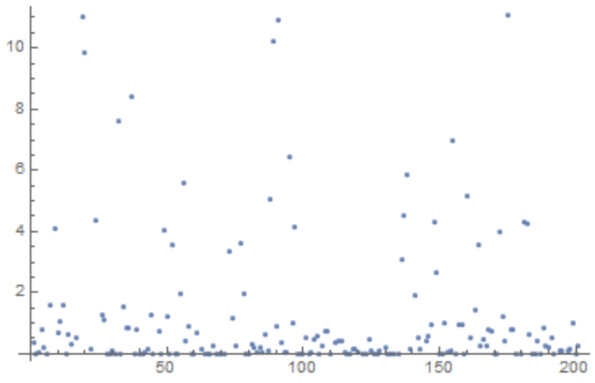


Fig. 11: The graphic illustrates around 80% of the data that John Cook tested. [19]

Original	Flipped
117.563	3.828
108.672	4.875
99.641	0.422
103.031	0.000
99.188	0.000
102.453	0.234
79.594	0.094
91.031	0.875
64.313	0.000
95.719	6.500
131.125	0.375
97.219	0.000
98.828	0.203

Fig. 12: The results table from John Cook's experiment. [19]

The result of the experiments are verified by generating a sequence of pairs of random 100-bit primes, factoring their product, and then factoring the product with a randomly chosen bit flipped, which consistently shows that flipping a bit makes the factorization easier. John D. Cook also noted that bit flipping can occur unintentionally, such as when memory is hit by cosmic ray. Like shows in the table on figure 12 [19].

V. Mitigation on Real-World Devices

Conducting Rowhammer attacks can be difficult to perform because of the requirement for high-resolution timers and system variability. However, previous research indicates that bit flips can be consistently reproduced. Therefore, it becomes possible to exploit them.

Increasing the refresh rate could help prevent the Rowhammer attack. Typical memory refresh rates are 64 milliseconds (in DDR3), but they may be increased. However, this results in increased power consumption and decreased memory performance. Even so, with double refresh rate mitigation, attacks remain possible [14].

There are several techniques proposed by previous research to mitigate the Rowhammer attack. Unfortunately, recent attacks and advances in technology continue to weaken these solutions. We will look at several previous solutions mentioned in [20]:

1) **Increased Refresh Rate:** Traditional systems operate with a refresh rate of 64 ms. To mitigate Rowhammer, the system's refresh rate may be increased by 2× to 4×, enabling refresh intervals between 32 ms or 16 ms. Moreover, a refresh rate of 2× to 4× may mitigate the risk of Rowhammer, but it does not ensure its complete elimination.
2) **Refresh Neighbor Rows:** Rowhammer can be mitigated by refreshing the victim rows through access and pre-charging. This can be achieved either by probabilistically refreshing the two neighboring rows (Probabilistic Row Activation or PRA) on obtaining a specific row, or by monitoring the access frequency for each row (Counter-Based Row Activation or CRA) and refreshing the neighboring rows once the access count reaches a threshold that has been set. Although these approaches show little performance overhead, their key assumption is that the Memory Controller (MC) knows information of the mapping or rows within the DRAM chip, and, most importantly, that the Rowhammer threshold is determined. Unfortunately, the Rowhammer threshold shows substantial variation across bits, and choosing a conservative Rowhammer threshold might not guarantee effective Rowhammer protection, since specific DIMMs might have a lower Rowhammer threshold. Moreover, the Rowhammer threshold might vary considerably between various technological generations and over time.
3) **Targeted Row Refresh (TRR)**: DRAM manufacturers have integrated TRR into DDR4 memory, which "refreshes" the next rows of frequently visited data. The details regard-

ing the tracking of frequent rows within the DRAM are typically not disclosed to the public. Nevertheless, attacks have succeeded in inducing Rowhammer despite the presence of TRR.

4) **Error Correcting Code (ECC)**: Typical ECC codes have been designed to address natural errors such as alpha particle impacts or chip malfunctions, mainly to improve the correction capacity. Rowhammer errors, which are not naturally occurring but can be planned by adversaries, are not a reliable issue but a security threat. To protect against these weaknesses, ECC codes must offer strong detection capabilities for arbitrary data corruption while maintaining a sufficient level of corrective capability. A Message Authentication Code (MAC) can help effectively to detect various failures. With the help of *Integrity Protected Memories* can be constructed when using current ECC codes. It helps for identifying Rowhammer failures, and mitigating the security risk with minimal impact on area, power, and performance [20]. This MAC approach may protect against adversarial memory corruption, such as Rowhammer attacks.

These are the proposed solutions to reduce Rowhammer attacks. However, not all the exploits mentioned above are prevented by these mitigations. For instance, the *Throwhammer attack* demonstrates that present mitigations, such as ECC memory and Targeted Row Refresh, may be insufficient for protecting against these threats [16]. To mitigate the remote Rowhammer *Throwhammer attack*, the literature proposes an innovative memory allocation strategy that isolates DMA buffers with guard zones, able to absorb any bit flips induced by an attacker, and introduces a new allocator named *ALIS* that securely allocates these buffers in addition to their guard zones [16].

The other mitigation strategies for the research titled *"From Random Block Corruption to Privilege Escalation: A File System Attack Vector for Rowhammer-like Attacks"* are distinct from the techniques mentioned earlier [18]. The implementation of disk encryption, such as `dm-crypt`, would fundamentally obstruct the attack, making it very unlikely for the attacker to gain control over the data stored on the disk. Additionally, ensuring metadata integrity would mitigate the file system attack vector. However, other attack vectors may still be vulnerable if possible application-level exploits are identified in the future [18].

The last mitigation for the research titled *"Drammer: Deterministic Rowhammer Attacks on Mobile Platforms"* by Victor et al. stated that hardware-based countermeasures, such as ECC memory and increased DRAM refresh rates, have limits against these attacks, as well as increased power consumption and performance loss, making them less suited for mobile devices [11]. Additionally, alternative motivations such as *Target Row Refresh* (TRR) and *Probabilistic Adjacent Row Activation* (PARA) require support from the DRAM chip or the memory controller, and their efficacy against the *Drammer attack* on mobile platforms requires further investigation. The research proposes that restricting userland interface access to DMA-capable memory could work as an effective countermeasure against the *Drammer attack* on mobile platforms, considering the critical role of Direct Memory Access in the deterministic Rowhammer attack on mobile devices [11].

VI. Future Direction and Conclusion

DDR5 memory is considered more secure, using an innovative mitigation approach known as PRAC. The Per Row Activation Counting (PRAC) framework for DDR5 enables precise enumeration of DRAM row activations. PRAC uses an Alert Back-Off (ABO) protocol to tell the memory controller to execute Rowhammer mitigation requests, solving the issues related to Targeted Refresh and guaranteeing that each victim row is refreshed in the set time limit.

Rowhammer attack testing is more often based on Intel CPUs. This is not because AMD CPUs are resistant to Rowhammer, but because of historical research and knowledge gaps. The first Rowhammer research and subsequent work were focused heavily on Intel's architecture and less attention to AMD architecture. However, newer investigations, such as the Zen Hammer study, have demonstrated that bit flips can also occur on AMD CPUs and with DDR5 by modifying techniques and aligning things differently [21]. The Zen Hammer investigation indicated that between ten tested systems, just one DIMM had a few bit flips, which may not accurately reflect the true risk [21]. For our tests, we had also selected a computer based on AMD Phenom II 1090T. Unfortunately, the FLIPPY.RAM software was not able to perform any tests on this machine.

The Rowhammer experiment performed in the flippyr.am project is interesting. [4] However, we discovered that the Rowhammer attack is practically challenging. Even so, it is not impossible that Rowhammer does not appear in our systems. We identified one of our old devices as vulnerable to

Rowhammer.js bit flips. Moreover, Rowhammer.js is still possible but rare, and it can take a long time to induce a bit flip by browsing a web page, with most computers not vulnerable because of non-default configurations.

In conclusion, although there may be no known attempts to exploit Rowhammer in real-world scenarios, or such attempts may be difficult to detect, if these exploits do exist, they might potentially be identified by an analysis of the assembly code for high-precision timers.

REFERENCES

[1] A. Jean-Pharuns, "Rowhammer.js is the most ingenious hack i've ever seen." [Online]. Available: https://www.vice.com/en/article/rowhammerjs-is-the-most-ingenious-hack-ive-ever-seen/

[2] 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 Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[3] C. Maurice and D. Gruss, "Rowhammer.js: Root privileges for web apps?" [Online]. Available: https://media.ccc.de/v/32c3-7197-rowhammer_js_root_privileges_for_web_apps

[4] Gruss, Heckel, and Adamsky, "Flippyr.am project," 2024. [Online]. Available: https://flippyr.am/

[5] S. Bhattacharya and D. Mukhopadhyay, "Advanced fault attacks in software: Exploiting the rowhammer bug," in *Fault Tolerant Architectures for Cryptography and Hardware Security*. Springer, 2018, pp. 111–135.

[6] L. Cojocar, J. Kim, M. Patel, L. Tsai, S. Saroiu, A. Wolman, and O. Mutlu, "Are we susceptible to rowhammer? an end-to-end methodology for cloud providers," in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 712–728.

[7] Wikipedia contributors, "Row hammer — Wikipedia, the free encyclopedia," 2025, [Online; accessed 16-May-2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Row_hammer&oldid=1290050048

[8] Hammertux, "Github - hammertux/rowhammering: Github page on rowhammer." [Online]. Available: https://github.com/hammertux/rowhammering

[9] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, Y. Oren, and T. Austin, "Anvil: Software-based protection against next-generation rowhammer attacks," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 743–755, 2016.

[10] R. Qiao and M. Seaborn, "A new approach for rowhammer attacks," IEEE, pp. 161–166, 2016.

[11] 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 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 1675–1689.

[12] D. Kim, H. Park, I. Yeo, Y. K. Lee, Y. Kim, H.-M. Lee, and K.-W. Kwon, "Rowhammer attacks in dynamic random-access memory and defense methods," *Sensors*, vol. 24, no. 2, p. 592, 2024.

[13] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1–18.

[14] D. Gruss, M. Heckel, and F. Adamsky, "Ten years of rowhammer: A retrospect (and path to the future)," Dec. 2024. [Online]. Available: https://media.ccc.de/b/congress/2024

[15] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 2016, pp. 300–321.

[16] A. Tatar, R. K. Konoth, E. Athanasopoulos, C. Giuffrida, H. Bos, and K. Razavi, "Throwhammer: Rowhammer attacks over the network and defenses," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 213–226.

[17] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in mlc nand flash memory programming: Experimental analysis, exploits, and mitigation techniques," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 49–60.

[18] A. Kurmus, N. Ioannou, M. Neugschwandtner, N. Papandreou, and T. Parnell, "From random block corruption to privilege escalation: A filesystem attack vector for rowhammer-like attacks," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[19] J. D. Cook, "Making public keys factorable with rowhammer," Jun. 2019. [Online]. Available: https://www.johndcook.com/blog/2019/06/21/rowhammer/

[20] M. Qureshi, "Rethinking ecc in the era of row-hammer," in *DRAMSec*, 2021.

[21] C. S. Group, "Zenhammer: Rowhammer attacks on amd zen-based platforms." [Online]. Available: https://comsec.ethz.ch/research/dram/zenhammer/