# A Survey of Rowhammer: Fault Injection by Exploiting Electrical Interactions in DRAM

PHILIP HODGES

Rowhammer is the name that has been given to a relatively recently discovered vulnerability in newer DRAM modules. In order to pack more memory into a smaller space, the density of the memory in these formats leads to the possibility of electrical interference between rows of bits stored in memory. Rowhammer attacks exploit this by executing many read/write operations to induce specific memory access patterns, causing memory faults. Research effort over the last couple of years has shown that the Rowhammer phenomenon is exploitable in practice, allowing programs in user space to gain root access in a variety of scenarios.

I present a survey of Rowhammer, discussing the most significant contributions to the security literature and a broad view of the state of current research.

## 1 INTRODUCTION

A key security property of a computer system is memory isolation: accesses to one memory address should not affect data stored in other memory addresses. However, as newer DRAM modules are built with more and more capacity, the cells are made smaller and are more densely packed. Electrical interference effects between memory cells become a possibility. It turns out that these interference effects can be amplified using specific memory access patterns. This undesirable behaviour has been known for some time, and was originally thought to only be an integrity issue [9], where random errors may occur. Researchers have been able to take advantage of these errors in order to develop powerful exploits. They have termed attacks related to these techniques "Rowhammer" attacks.

Security research into Rowhammer began in 2014, and has since progressed rapidly, with subsequent publications showing an increasing variety and severity of exploits. The Rowhammer phenomenon went from a potential reliability issue to a pervasive, severe, and difficult-to-prevent hardware-based security flaw in only a few years.

I present a survey of the Rowhammer literature, starting from the first publications, up until very recent publications representing the best-known attacks and the possible mitigations in place.

### 1.1 Survey Methodology

To review the literature, I started with the publication that originally introduced Rowhammer to the security community [8]. I then went through all the most cited publications which cited this original paper or subsequent research, working forward. This found the bulk of the core literature on the subject. During a first reading of this literature, I made note of and read other commonly referenced or highly relied-on publications. Due to space constraints, not every publication related to Rowhammer has been included. I included well-written, influential papers, while also discussing enough of a variety of publications to show the various ways Rowhammer can be applied.

Author's address: Philip Hodges.

I have written this survey with a narrative in mind, as opposed to simply an accounting of the most recent research. I believe this is appropriate, as Rowhammer is a relatively new phenomenon, and the research is still in motion. Furthermore, Rowhammer is quite unique among vulnerabilities, being potentially the only example of a hardware fault that is so practically exploitable. Thus, "where is the current research at?" may be less instructive to answer than "how has the research progressed?" The evolution of a fault that was once thought to be benign into a pervasive security issue is of significant interest.

### 1.2 Overview of Survey

Section 2 will cover the original publication that introduced Rowhammer. Section 3 covers the first exploit based on Rowhammer. Sections 4 to 6 cover major subsequent contributions from various authors that improve techniques, find new attack vectors, and tackle previous suggestions for future research made by other authors. Section 7 discusses Rowhammer mitigations. Section 8 covers other situations where researchers have been able to develop Rowhammer exploits.

## 2 ROWHAMMER: THE FIRST ITERATION

The paper "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors" by Kim et al. [8] introduced the idea of Rowhammer to a wide audience. While the phenomenon of disturbance errors in DRAM was known, and repeated row accesses were known to increase the number of errors, Kim et al. were the first to fully study and characterize the behaviour of these errors. Their work was exceptionally well-written, and provided a foundation for future works to build on. In this section, we will go through the main contributions of their paper as a means to provide a full description of how Rowhammer works.

### 2.1 How DRAM Works

A background in how DRAM works is necessary for understanding Rowhammer. For this survey, we will not go into complete detail on exactly how the circuits in DRAM are implemented. This detail is necessary for understanding why Rowhammer errors occur, but not how to cause them.

*2.1.1 DRAM Organization and Instructions.* DRAM is organized as a collection of grids of individual memory cells. Each memory cell stores a bit, represented by having a capacitor in a charged or discharged state. The rows of the DRAM grid are known as *wordlines*, and the columns are known as *bitlines*. Each grid is known as a *bank*. For each bank, there is a row buffer. A memory access works as follows. Given a specific cell to access (specified by its row and column addresses):

(1) First, `ACTIVATE` the row. This transfers the data in the row to the row buffer. This operation is destructive (i.e. the data in the row is lost), so the data in the row buffer is also copied

back into the row at the same time. All subsequent accesses to this row until the row is deactivated are served by the row buffer.
(2) The desired column can be read from or written (READ/WRITE) to through the row buffer. If many columns need to be accessed, they can all be accessed before the row is deactivated.
(3) The row is deactivated (PRECHARGE), and the row buffer is cleared.

All these operations can be performed individually if given complete access to the DRAM module.

A last important operation is the REFRESH instruction. DRAM cells leak charge, and therefore will lose the value contained in them over time. In order to avoid data loss, the cells need to be refreshed often. Since activating a row also re-writes the data back into that row, a REFRESH command is essentially identical to an ACTIVATE command followed by a PRECHARGE command. In practice, REFRESH commands perform these activations in bulk, refreshing many rows at once.

Modern DRAM cells are guaranteed to have a retention time of 64ms. Within this time frame, enough REFRESH commands are issued to make sure that every row has been refreshed exactly once.

*2.1.2 Disturbance Errors.* The key observation is, as noted by Kim et al. [8], when a wordline is repeatedly activated, some cells in nearby rows will leak charge faster than usual. Critically, some cells may not retain their charge for the whole guaranteed retention period, and therefore will not be refreshed before they exhibit a change in value. Kim et al. give some reasons why this might occur, but here it suffices to say that it is due to various types of electrical interference.

## 2.2 Code for Rowhammering

We now know how DRAM works and the type of accesses that give rise to increased disturbance errors. The code to induce these accessed is then very simple, and the example given by Kim et al. is in Listing 1.

```
1 start:
2         mov(X), %eax
3         mov(Y), %ebx
4         clflush(X)
5         clflush(Y)
6         mfence
7         jmp start
```

Listing 1. Code to Trigger Rowhammer Effect

With suitably chosen addresses X and Y (addresses corresponding to different rows in the same bank), this code will translate to repeated ACTIVATE, READ, and PRECHARGE instructions issued to X and Y in an alternating fashion. Note that both addresses are necessary, since otherwise the row will be activated only once and read from many times, while we want many activations. The clflush instruction evicts the data that has just been read from the cache, so that all the accesses indeed must be directly to the DRAM. Simplified, the mfence instruction serves to ensure that previous memory access instructions are completed before future ones are begun.

This simple code was able to induce thousands of errors in various DRAM modules. It is easy to see that these types of errors could lead to various kinds of data corruption. Kim et al. additionally recognized that a smart attacker might be able go as far as using Rowhammer-induced errors to hijack a system, but leave such a task to future researchers.

## 2.3 Rowhammer Characterizations

Kim et al. provide extensive details on their experimental methodology [8], which we will not reproduce here. They use this experimentation to support several characterizations of the Rowhammer phenomenon, which have been taken as axioms by future researchers, so we will give a recap of their characterizations in this section.

**Errors are Widespread**. Kim et al. concluded that there are likely to be many vulnerable cells (i.e. at least in the thousands) in any recently manufactured DRAM module.

**Access Pattern Matters**. Repeated *activation* of the row, and not just repeated reads, are necessary to cause errors.

**Errors Occur in Physically Adjacent Rows**. While some discrepancies are found (mainly attributed to address remapping), Kim et al. concluded that errors were induced primarily in the two rows that were physically adjacent to the row being activated.

**Data Pattern Matters**. Kim et al. Concluded that the pattern of the data stored in the DRAM during testing (e.g. all zeros, all ones, a checkerboard pattern) made a significant difference in the number of errors induced. This can be partly explained by the fact that discharge of cells happens in only one direction (i.e. charged → discharged).

**Errors are Repeatable** Many iterations of their experiments yielded the same number of errors within a small margin, so vulnerable cells are reliably vulnerable.

**Vulnerable Cells are Not Weak Cells** Cells vulnerable to Rowhammer are not those that happen to have a lower retention time.

**Not Strongly Affected by Temperature** While retention time of cells is affected quite strongly by temperature, the number of induced Rowhammer errors is not.

These last two points indicate that the reasons for the errors are more complicated than simply a blanket reduction of retention time of cells when exposed to Rowhammering.

## 2.4 Solutions

In this section, we describe the potential solutions put forth by Kim et al. Their provided solutions were well thought-out, as many solutions proposed by future researchers are some implementation of the solutions first described in their paper.

*2.4.1 Potential Solutions.* Kim et al. provide the following 6 solutions, all with significant drawbacks:

(1) Make better chips
(2) Correct errors by using ECC
(3) Increase refresh frequency of cells
(4) Have the manufacturer remap vulnerable cells
(5) Have the consumer remap vulnerable cells
(6) Identify often-used rows and refresh their neighbours

Since there may be many vulnerable cells, solutions 4 and 5 may not work. Solution 1 would be nice, but is a bit optimistic.

Solution 2 is promising, but current ECC memory incurs a 12.5% overhead to allow detection of single bit flips. I will revisit ECC memory later.

Solution 3 provides a good stop-gap, as it will help reduce errors. However, in their analysis, Kim et al. note that to eliminate all errors in some DRAM modules, the refresh frequency would have to be once every 8.2ms, leading to up to a 35% overhead. This is therefore not a workable solution for eliminating errors.

Solution 6 would work well, but there would be much difficulty in trying to keep track of precisely which rows are activated most. Kim et al. tried some alternatives, but the implementation was too expensive for the performance observed. However, a variant on this solution with much less overhead was their main proposal for preventing Rowhammer errors.

*2.4.2 Proposed Solution.* Building on the idea from solution 6, Kim et al. propose *PARA* (*probabilistic adjacent row activation*). As its name suggests, the main idea of PARA is to, whenever a row is activated, refresh one of its adjacent rows with some probability. Kim et al. were able to choose a probability such that errors become virtually nonexistent in adjacent rows. Even further increasing the refresh probability (due to uncertainty about row address remapping and thus adjacency information), their implementation only incurred around a 0.2% overhead.

## 3 PROJECT ZERO: A FIRST EXPLOIT

As a follow-up to the Kim et al. paper, Seaborn and Dullien from Google Project Zero took on their suggested future research task of using Rowhammer-induced errors to hijack a system [12]. Their results were published on the Google Project Zero blog. They provide many new ideas and insights throughout their blog post, as well as experimental results of their own. I give a short recap of their primary contributions here.

### 3.1 Address Selection

Kim et al. avoided extensive discussion on how the addresses of the rows are picked in the code in Listing 1. Seaborn and Dullien emphasize that the critical condition is that the addresses refer to two different rows in the same bank. They propose several ways of finding addresses that satisfy this condition, and introduce the notion of double-sided hammering.

**Using the physical address mapping**. Using knowledge of how a CPU's memory controller maps physical addresses to DRAM cells, as well as some additional information, one can pick different rows that are likely in the same bank. This is the approach taken by Kim et al.

**Random address selection**. A simple approach, and without risk of using a poor strategy, is to randomly select addresses. On a typical system, there could be a 1/16 chance that two randomly chosen addresses are in the same bank, which is a good probability. Increasing the number of addresses being hammered also helps this method.

**Selecting addresses using timing**. Pairs of addresses that are in the same bank will have slower uncached access time than those that aren't, and so can be found with fine-grained timing.

**Double-sided Hammering** Seaborn and Dullien found that hammering the two rows directly adjacent to a target row increased the number of induced errors significantly. This is expected from the characterizations of Kim et al. Called double-sided hammering, this technique is more difficult to implement. It requires, in particular, knowledge of the address offset between adjacent rows. In practice, the desired addresses can be found experimentally, by determining which offsets around a target row maximize the number of induced errors.

### 3.2 Exploiting Bit Flips

The main goal of Seaborn and Dullien was to give a practical exploit of the Rowhammer-induced errors. They point out that historically, two things have often been true: that bugs that initially appear to only be reliability issues often turn out to be security issues, and that it is well-known among security researchers that single bit flips are exploitable. Thus it is not unexpected that they were successful in providing an exploit in this circumstance. They provide two exploit methods in two different environments: Google Native Client (NaCl) and x86-64 Linux.

*3.2.1 Escaping from the NaCl Sandbox.* The Google Native Client provides a sandbox in which a subset of x86-64 machine code can run. This subset has been validated and determined to be safe. However, bit flips in code corresponding to safe instructions can turn that code into code with unsafe instructions. Seaborn and Dullien provide an example of this.

NaCl uses the instructions in Listing 2 to do indirect jumps. Many different bit flips may be exploitable, and we describe one here. If a bit flip occurs in bit 0 of the register number in `jmp *%rax`, then it will turn into `jmp *%rcx`. This latter command is an unconstrained jump, which allows jumping to any address, as opposed to jumping to only 32-byte aligned addresses, as NaCl usually allows. Since other unsafe commands can be hidden inside safe ones, we can then jump to shell code hidden inside NaCl-validated instructions.

```
andl $~31, %eax
addq %r15, %rax
jmp *%rax   // Indirect jump.
```

Listing 2. NaCl indirect jumps

This can be exploited with Rowhammer. First, fill the code area of memory of NaCl with the indirect jump instructions. Then, Rowhammer this memory using the code in Listing 1 (with random addresses) until a bit flips occurs. If the bit flip is exploitable as above, we can use it to run shell code. Otherwise, continue hammering.

To mitigate this attack in NaCl, the `clflush` instruction has been disabled. Unfortunately, this mitigation is not possible in all scenarios.

*3.2.2 Privilege Escalation in Linux.* Seaborn and Dullien provide an alternate exploit for a normal x86-64 process running on Linux. This exploit is based on flipping bits in page tables, and escalating privilege of the process to gain access to all of physical memory. In particular, if we are able to modify a page table entry (PTE) so that it points to a physical page that contains a page table of our

process, then we will have read-write access to a page table of our own process. Modifying this table will give us access to any physical memory address on the system.

First, memory allocation methods are used to ensure that allocated memory ends up heavily fragmented. Then, at a high-level, the desired bit flip in a page table can be induced (with high probability) as follows (full details are included in the original blog post [12]):

(1) Repeatedly use `mmap()` on a data file to fill memory with page tables, all of which point at the data file.
(2) Populate some of the PTEs by accessing their corresponding pages.
(3) Run `munmap()` on the target page. The kernel will, with high probability, reuse this page for a new page table.

Now, Rowhammer this area of code using the code in Listing 1 until a bit flips occurs. These bit flips are harder to detect than the NaCl exploit, but scanning the mapped region to see if any page tables do not point at the data file is sufficient. If one is found, there is a chance that it is pointing to a page table for the address space of our process.

We can then determine which virtual address our page table points to, and if everything worked as desired, then we have write access to the page table of our own process. As noted above, this gives us write access to all of physical memory.

From here, it is evident a serious attack can be completed. Seaborn and Dullien give a few examples with various advantages and drawbacks: modify an SUID-root executable, modify a library that an SUID executable uses, etc.

It is worth noting that a much more recent paper by Wu et al. [16] is able to successfully rout this page table-based attack with a small modification to the OS memory allocator.

### 3.3 Alternatives to `clflush`

Another important contribution by Seaborn and Dullien was the identification of alternatives to the `clflush` instruction. They suggest several different possibilities, some of which are implemented in future papers. These include normal memory accesses in specific patterns, non-temporal memory accesses, atomic memory accesses, uncacheable pages, and others.

### 3.4 New Proposed Mitigation

Seaborn and Dullien discuss some of the same mitigations as Kim et al. They give some evidence to support the fact that certain newer DRAM modules may be implementing some of the more basic mitigations such as increased refresh rates.

They also propose another new solution: using CPU performance counters to detect hammering. Any method of hammering requires many cache misses, and so monitoring for abnormal amounts of cache misses can detect a Rowhammer attack. Future research is needed, as it is not clear exactly what to do if hammering is detected, and how common false positives will be.

## 4 ROWHAMMER WITH REGULAR MEMORY ACCESSES

One of the next papers to be published on Rowhammer following the Project Zero blog post [12], by Gruss, Maurice, and Manguard [6], advances the research in two significant ways. They develop a framework for studying cache eviction strategies, allowing them to trigger Rowhammer without using `clflush`, and they implement a Rowhammer attack in JavaScript, that runs in a browser.

### 4.1 Cache Eviction Strategies

As noted by both Kim et al. [8] and Seaborn and Dullien [12], the Rowhammer attack would be made more versatile and harder to defend against if the `clflush` instruction was not necessary. Gruss et al. develop a solution to this. They present a formal model for studying memory access patterns that induce cache evictions. They then take an experimental approach to determine the best parameters in their model for memory accesses that maximize cache evictions.

By trying many different memory access patterns, they determine which ones induce the most cache evictions on each machine they study. They call this the offline phase.

During their online phase, while trying to exploit a machine which they may not know details about, they suggest trying the best strategies found in the offline phase. If the target machine matches a previously seen machine, we will find a good strategy with a high eviction rate; if not, we may find an acceptable strategy, or use a “Fall-back Attack" provided by Gruss et al.

These memory access patterns then allow many cache evictions to be forced with little knowledge of the target machine, and with very little privilege.

### 4.2 JavaScript Implementation

Using their new cache eviction strategy, Gruss et al. implement Rowhammer in JavaScript. Seaborn and Dullien speculated that a JavaScript implementation would have to use large typed arrays to fill memory, and this is the approach taken here. Since JavaScript does not provide access to low-level memory, Gruss et al. used previous work in timing attacks to determine the addresses to hammer. They describe how to implement an exploit using their implementation, which is very similar to the page table-based attack by Seaborn and Dullien.

Their implementation of Rowhammer does produce bit flips in practice, although fewer than before, as `clflush` will cause more row activations than tailored memory access patterns. Furthermore, being able to exploit Rowhammer in a browser makes the vulnerability exploitable remotely, which makes it significantly more severe.

## 5 ROWHAMMER WITH NON-TEMPORAL INSTRUCTIONS

In “A New Approach for Rowhammer Attacks" [11], Qiao and Seaborn present another way to avoid the use of `clclush`, by using non-temporal instructions. Briefly, non-temporal instructions are those that indicate to the CPU that caching the data accessed is unlikely to be useful (as it will only be used once, say). As an efficiency measure, the CPU will not cache data from the access. It is easy to see that this is a desirable property for use in Rowhammer implementations.

Qiao and Seaborn identify non-temporal store instructions, `MOVNTI` and `MOVNTDQ`, that can be used in Rowhammer. Since these are store instructions, we now need write access to the addresses that we are hammering.

Qiao and Seaborn give an exploit in NaCl (as in the case of Seaborn and Dullien's attack [12]). They show how to induce data rows and code rows to be interleaved in memory, so that we can hammer the data rows to flip bits in the code rows. Their implementation of this is successful, and they are able to escape the NaCl sandbox.

### 5.1 Bit Flips with `libc` Code

Given that non-temporal instructions can be used to trigger Rowhammer, Qiao and Seaborn explore the possibility of triggering Rowhammer using benign code that already exists on the target system. In particular, they found that the ubiquitous `memcpy` and `memset` functions both use non-temporal instructions when invoked with certain parameters. However, they were unable to get the row activation rate high enough to observe bit flips when repeatedly calling `memset`. They caution, however, that improvements could make this a valid approach, yielding a devastating range of attack opportunities wherever `memcpy` or `memset` are used.

## 6 ANOTHER FLIP IN THE WALL: BYPASSING ALL KNOWN DEFENSES

A recent, well-written paper by Gruss et al., titled "Another Flip in the Wall of Rowhammer Defenses" [5], contributes several new techniques, systematically classifies existing defenses, and demonstrates that their new attacks evade all existing defenses, even in combination. This paper is the most recent paper in the core developments of Rowhammer attacks. While there have been more papers published, they tackle more specific threat models. Thus this paper can be considered the state-of-the-art of Rowhammer attacks.

### 6.1 Classifying Defenses

Gruss et al. divide existing Rowhammer defenses into the following five categories:

(D1) Detection through static analysis (similar to NaCl disallowing `clflush` as it can be used for Rowhammer)
(D2) Detection through performance counter analysis (e.g. counting cache misses)
(D3) Detection through analysis of memory access patterns (which would detect the earlier attack of Gruss et al. [6]).
(D4) Prevention by strictly avoiding physical proximity (would prevent exploitable bit flips).
(D5) Prevention by preventing conspicuous memory footprints (preventing the page table-based attack of Seaborn and Dullien [12] and related attacks)

Note that not every type of defense mentioned so far is included in one of these categories. In particular, many suggested hardware-based defenses are not included. Gruss et al. specifically did not include defenses that would be difficult to retroactively add to memory modules that are currently vulnerable, which precludes many hardware-based defenses.

To implement a Rowhammer attack that defeats these countermeasures, Gruss et al. develop several new techniques.

### 6.2 Abusing Intel SGX

To defeat both defense classes D1 and D2, Gruss et al. run their attack inside an SGX enclave. The code in the enclave is only decrypted at runtime, and hence cannot be inspected statically. Also, such code is not monitored by the CPU for performance purposes.

### 6.3 One-Location Hammering

To defeat defense class D3, Gruss et al. use "one-location hammering". Previous hammering methods required addresses corresponding to two different rows in the same bank of memory. If only a single row was used, then the memory controller would simply keep the row buffered, avoiding many activations. Recent changes in memory controller policies will sometimes cause rows to preemptively close early, eliminating the necessity of using two locations. Gruss et al. show that accessing one memory location repeatedly is enough to trigger bit flips. Since defenses in category D3 were based on the assumption of Rowhammer accessing two addresses, they do not work against this attack.

### 6.4 Opcode Flipping

To defeat defense class D4, Gruss et al. introduce opcode flipping. Previous works used attacks based on page tables, which can be defeated by keeping kernel pages and user-accessible pages separate. In contrast, flips in opcodes are not avoided by this, since they are bit flips in user pages. If an attacking process is able to cause a bit flip in the `sudo` binary, for example, it could provide root privileges to the attacking process, by breaking the password-verification logic of `sudo`.

### 6.5 Memory Waylaying

To defeat defense class D5, Gruss et al. use memory waylaying instead of filling memory with e.g. page tables as in previous attacks. Memory waylaying uses only page cache pages, which are considered as available memory, and hence not visible in the system memory utilization. When loading page cache pages, they are loaded to random physical addresses in DRAM. By loading and evicting a page cache page continuously, and using a side-channel technique to detect when it is placed on a specific physical address, it is possible to load the page cache page at a desired location. An attacker can then use an opcode flip as above to gain root privileges.

## 7 DEFENSES

I have discusses many proposed defenses against Rowhammer and how various Rowhammer implementations avoid these defenses. In this section we will recap the research in this direction.

Gruss et al. [5] provided a very good categorization of Rowhammer defenses. Their classification, together with the hardware-based defenses that were suggested early on by Kim et al. [8], encompass all known defenses against Rowhammer. Some specific, more developed implementations of these defenses include ANVIL [2], a performance counter-based defense, ARMOR [4], a hardware-based cache to store frequently accessed rows, and MASCAT [7], a static code analysis tool.

As Gruss et al. showed, all of the defenses they categorized, even when used together, are inadequate. Among hardware-based defenses, there are a few options. PARA, as introduced by Kim et al. [8], and Target Row Refresh (TRR) (detailed as a recommended feature in the LPDDR4 standard [1]), are two options that appear to be the

most effective defenses. This is expected, as they simply put limits on how many row activations can happen before the neighbours of a row will be refreshed. A remaining concern is what overhead these methods would incur, but Kim et al. showed that PARA is very effective with very low overhead.

The existence of effective hardware-based prevention mechanisms is not sufficient. Modules that do not already implement these solutions will always be vulnerable, and hence software-based solutions are strongly desired. DDR3 modules do not have hardware-based countermeasures, and while some countermeasures were included in the DDR4 specification, they are only recommended, and not required. Thus vast numbers of machines will be weak to Rowhammer attacks for many years if an effective software-based solution is not found.

### 7.1 ECC Memory

Memory based on error-correcting codes, or ECC memory, is another hardware-based defense that many have suggested may prevent Rowhammer bit flips. However, many researchers [3, 5, 6, 8, 11, 14] have pointed out that current ECC DRAM modules would not defend against all Rowhammer attacks, as they would not be able to correct multi-bit flips.

Cojocar et al. in particular do an in-depth study of Rowhammer attacks in the presence of ECC memory [3]. They first use techniques to reverse engineer ECC implementations in commodity systems, and then make improvements to existing Rowhammer attacks to demonstrate that ECC memory can still be vulnerable to Rowhammer in practice.

## 8 OTHER ROWHAMMER APPLICATIONS

In this section we briefly mention a few other situations where researchers have managed to implement effective Rowhammer attacks, which further illustrate the severity of the vulnerability. These works all include improvements to various parts of the Rowhammer attacks, and so are interesting case studies in their own right.

### 8.1 Mobile Devices

There are two papers that address Rowhammer attacks on mobile devices [14, 15], answering the pondering of earlier researchers [6, 8] on whether the ARM architecture would be vulnerable in the same way as x86-64.

The first, coined Drammer [14], developed a deterministic Rowhammer attack, relying on predictable memory reuse patterns. This leads them to an android root exploit that uses no software vulnerabilities or user permissions.

The second paper [15] provided more attacks on the latest Android OS of the time, and then suggested a countermeasure called GuardION, that is able to prevent the type of exploit used in their attacks and in Drammer. It is entirely possible that other exploit methods on Android are found.

### 8.2 Rowhammer in Networks

There are two papers that address Rowhammer over a network [10, 13]; that is, in a situation where the attacker simply sends packets to the target without controlling any code on the target system. The research in these two papers was conducted concurrently and independently.

The first, coined Throwhammer [13], describes a method for sending packets through a network with high enough throughput to trigger the Rowhammer effect. The authors then show how to exploit the resulting bit flips.

The second, coined Nethammer [10], contains very similar contributions. They show how to use Nethammer to exploit systems that use uncached memory or flush instructions when dealing with network requests. Notably, their experiments show that TRR on the target DRAM does not always eliminate all bit flips. This throws some doubt into the assumption that Rowhammer bit flips occur in rows that are physically adjacent to the hammered row, as was assumed by Kim et al. [8].

### 8.3 Rowhammer in the Cloud

Rowhammer has also been shown to be practical in cross-VM situations [17]. Specifically, new techniques have allowed the breaking of Xen paravirtualized memory isolation, leading to exploits in public clouds where this technology is used.

## 9 CONCLUSION

Rowhammer has exposed the most significant hardware-based security vulnerability in recent memory. I have shown the evolution of the Rowhammer bug from inception to current research. The research literature has covered from the details of Rowhammer behaviour in hardware to the various different kinds of high-level exploits that can be executed using Rowhammer-induced bit flips.

The research is still very much in motion, with many research suggestions in published papers not yet attempted. Improvements to Rowhammer techniques will continue to make Rowhammer a more and more severe attack.

As of now, Rowhammer attacks are very much ahead of the defenses. Many of the most recent papers demonstrate attacks that work even in the presence of all known software-based defenses combined. Hardware-based defenses are effective, but cannot be retrofitted to the vast amount of vulnerable memory already in commodity systems.

Finding a widely effective software-based solution to detect and/or prevent Rowhammer is the most important open problem in the area. If this is not possible, then defenses specifically tailored to deter the various kinds of Rowhammer exploits should be studied. In any case, new Rowhammer-based attacks should continue to be studied, so that researchers can help find new ways to prevent the emerging threats, before these attacks are able to be used for malicious purposes.

## REFERENCES

[1] Association, J. S. S. T. Low power double data rate 4. http://www.jedec.org/standards-documents/docs/jesd209-4b, 2017.
[2] Aweke, Z. B., Yitbarek, S. F., Qiao, R., Das, R., Hicks, M., Oren, Y., and Austin, T. Anvil: Software-based protection against next-generation rowhammer attacks. *SIGOPS Oper. Syst. Rev. 50*, 2 (Mar. 2016), 743–755.
[3] Cojocar, L., Razavi, K., Giuffrida, C., and Bos, H. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. pp. 55–71.
[4] Ghasempour, M., Lujan, M., and Garside, J. Armor: A run-time memory hot-row detector. http://apt.cs.manchester.ac.uk/projects/ARMOR/RowHammer/, 2015.

[5] Gruss, D., Lipp, M., Schwarz, M., Genkin, D., Juffinger, J., O'Connell, S., Schoechl, W., and Yarom, Y. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy (SP)* (May 2018), pp. 245–261.

[6] Gruss, D., Maurice, C., and Mangard, S. Rowhammer.js: A remote software-induced fault attack in javascript. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721* (Berlin, Heidelberg, 2016), DIMVA 2016, Springer-Verlag, pp. 300–321.

[7] Irazoqui, G., Eisenbarth, T., and Sunar, B. Mascat: Preventing microarchitectural attacks before distribution. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2018), CODASPY '18, ACM, pp. 377–388.

[8] Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J. H., Lee, D., Wilkerson, C., Lai, K., and Mutlu, O. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *SIGARCH Comput. Archit. News 42*, 3 (June 2014), 361–372.

[9] Lenovo. Row hammer privilege escalation. https://support.lenovo.com/us/en/product_security/row_hammer, July 2016.

[10] Lipp, M., Aga, M. T., Schwarz, M., Gruss, D., Maurice, C., Raab, L., and Lamster, L. Nethammer: Inducing rowhammer faults through network requests, 2018.

[11] Qiao, R., and Seaborn, M. A new approach for rowhammer attacks. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (May 2016), pp. 161–166.

[12] Seaborn, M., and Dullien, T. Exploiting the dram rowhammer bug to gain kernel privileges. https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, Mar 2015.

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

[14] van der Veen, V., Fratantonio, Y., Lindorfer, M., Gruss, D., Maurice, C., Vigna, G., Bos, H., Razavi, K., and Giuffrida, C. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2016), CCS '16, ACM, pp. 1675–1689.

[15] van der Veen, V., Lindorfer, M., Fratantonio, Y., Padmanabha Pillai, H., Vigna, G., Kruegel, C., Bos, H., and Razavi, K. Guardion: Practical mitigation of dma-based rowhammer attacks on arm. In *Detection of Intrusions and Malware, and Vulnerability Assessment* (Cham, 2018), C. Giuffrida, S. Bardin, and G. Blanc, Eds., Springer International Publishing, pp. 92–113.

[16] Wu, X.-C., Sherwood, T., Chong, F. T., and Li, Y. Protecting page tables from rowhammer attacks using monotonic pointers in dram true-cells. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2019), ASPLOS '19, ACM, pp. 645–657.

[17] Xiao, Y., Zhang, X., Zhang, Y., and Teodorescu, R. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association, pp. 19–35.