Master's Thesis

Reliable Rowhammer Attack and Mitigation Based on Reverse Engineering Memory Address Mapping Algorithms

Saeyoung Oh (오 세 영)

Department of Computer Science and Engineering

Pohang University of Science and Technology

2019


POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY
1986
포항공과대학교

메모리 주소 매핑 알고리즘의 역공학 기반의 신뢰된 로우해머 공격 및 방어 기법

Reliable Rowhammer Attack and Mitigation Based on Reverse Engineering Memory Address Mapping Algorithms

Reliable Rowhammer Attack and Mitigation Based on Reverse Engineering Memory Address Mapping Algorithms

by

Saeyoung Oh
Department of Computer Science and Engineering
Pohang University of Science and Technology

A thesis submitted to the faculty of the Pohang University of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the Computer Science and Engineering

Pohang, Korea
12. 20. 2018
Approved by
Jong Kim
Academic advisor

Reliable Rowhammer Attack and Mitigation Based on Reverse Engineering Memory Address Mapping Algorithms

Saeyoung Oh

The undersigned have examined this thesis and hereby certify that it is worthy of acceptance for a master's degree from POSTECH

12. 20. 2018

Committee Chair Jong Kim

Member Chanik Park

Member Gwangsun Kim

MCSE 오 세 영. Saeyoung Oh
20172171
Reliable Rowhammer Attack and Mitigation Based on Reverse Engineering Memory Address Mapping Algorithms,
메모리 주소 매핑 알고리즘의 역공학 기반의 신뢰된 로우해머 공격 및 방어 기법
Department of Computer Science and Engineering , 2019, 27p, Advisor : Jong Kim. Text in English.

ABSTRACT

Rowhammer attacks intentionally induce bit flips to corrupt victim's data whose integrity must be guaranteed. To perform sophisticated rowhammer attacks, attackers need to repeatedly access the neighboring rows of target data. In DRAM, however, the physical addresses of neighboring rows are not always contiguous even if they are located before or after a target row. Hence, it is important to know the mapping algorithm which maps between physical addresses and physical row indexes not only for an attack but also for protection.

In this thesis, we introduce a method to reverse engineer the exact mapping algorithm and demonstrate that the assumption in previous rowhammer work is faulty. In addition, we introduce a novel and efficient rowhammer method and improve existing mitigations that have a security hole caused by the faulty assumption. Finally, we evaluate the effectiveness of the proposed attack and show that the improved mitigations almost perfectly defend against rowhammer attacks.


POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY
1986
포항공과대학교

Contents

List of Tables

List of Figures

I. Introduction

In the last decades, DRAM has evolved dramatically. Researchers and manufacturers have devoted many efforts to increase the capacity of DRAM memory. As a consequence, the recent DRAM achieves high cell density, and thus, can hold large amounts of data. Despite the benefit, the development also brought side effects such as hardware faults which threaten data integrity.

A rowhammer attack is a representative attack abusing a disturbance error which is one of the high-density DRAM hardware faults. This attack corrupts target data by maximizing the effects of disturbance errors. In specific, the attacks repeatedly access the neighboring rows of the target row that contains the target data. Although rowhammer attacks exploit a hardware fault, the attacks are performed at the software level without direct access to the hardware.

One of the requirements for rowhammer attacks is that attackers need to access the neighboring rows of the target row in the same bank. However, the mapping information between the user-level locations (e.g., virtual addresses or physical addresses) and physical locations on DRAM (e.g., row and bank information) is not available publicly. Therefore, it is not straightforward to access the upper and lower neighboring rows in the same bank.

To find the mapping, previous work has reverse engineered the mapping algorithm between physical addresses and physical DRAM bank indexes [1,2]. Using the mapping algorithm for banks, it is possible to track down bank indexes with physical addresses. However, no method has been proposed to find the exact mapping algorithm for rows, although a part of the mapping algorithm is already revealed [3]. To defend against the rowhammer attacks, a mitigation also has to understand the mapping algorithm since it has to locate the exact neighboring

rows. However, due to the difficulty of knowing row-mapping information, previous work has used a naïve assumption that DRAM rows are arranged identically to the sequence of physical addresses [4, 5]. If the assumption is faulty, these mitigations will not work as the intended way.

In this thesis, we introduce a method to reverse engineer the mapping algorithm for rows. This method exploits the insight that rowhammer induces bit flips only on the neighboring rows. Based on this method, we demonstrate the invalidity of the commonly presumed assumption that physically contiguous rows have also contiguous physical addresses, and propose a novel and precise mapping algorithm-aware rowhammer attack. Also, we show that this invalid assumption causes a security hole in existing mitigation methods, and improve the existing mitigation by using our method to reverse engineer the mapping algorithm. Finally, we evaluate our attack and mitigation methods.

The contributions of our research are as follows:

- We introduce a method to reverse engineer the DRAM row organization and discover the mapping algorithm for rows. To the best of our knowledge, no such method has been used by the previous work to reverse engineer the organization of DRAM modules.
- We propose a novel and precise rowhammer attack with exact mapping algorithm.
- We explain a security hole of existing mitigations and improve one of the mitigations using our exact mapping algorithm.

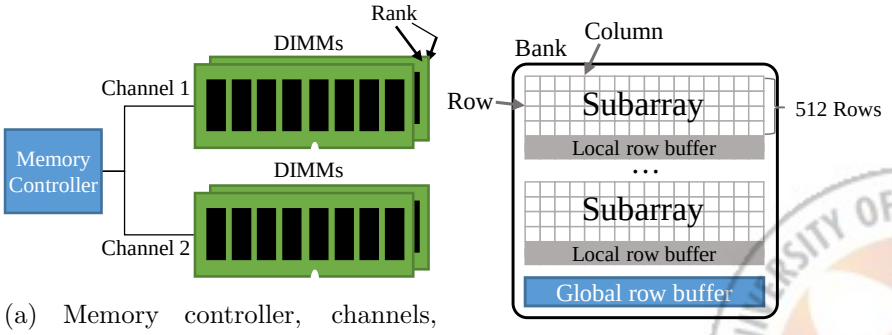
II. Background

In this chapter, we introduce DRAM organization and a rowhammer bug. In addition, we explain the prior work to reverse engineer the DRAM mapping algorithm.

2.1 DRAM Organization

DRAMs are hierarchically organized (Fig. 2.1a). Channels connect a memory controller to Dual Inline Memory Modules (DIMMs). Modern DIMMs can have at most eight ranks which denote sets of DRAM chips. Each rank has multiple banks and banks contain numerous cells that store the voltage representing logical data. All cells are connected horizontally and vertically through wordlines and bitlines. The horizontally and vertically connected cells are respectively called rows and columns (Fig. 2.1b).

The 2D array of cells is subdivided into several subarrays [6]. Each subarray is composed of 512 rows and has regularity, which indicates that the subarray internal organizations are same as each other. All the cells in one subarray are



(a) Memory controller, channels, DIMMs and ranks

(b) Subarrays within a bank

Figure 2.1: DRAM organization.

connected to one local row buffer, and all local row buffers are connected to one global row buffer (Fig. 2.1b). Both local row buffers and global row buffers receive and store the voltage of one accessed row, and amplify the voltage of DRAM data to a recognizable level. These row buffers stores the voltage until the memory controller accesses a new row which is located in the same bank. When the new row is accessed, the original voltage of the row buffer is flushed, and then the row buffer receives and stores the voltage from the new accessed row.

The data on the DRAM cells are volatile because the voltage representing the data is leaked over time, leading to data loss. Therefore, the voltage is maintained by refreshing cells periodically before the voltage drops below the threshold (the indicator to determine whether the data is 0 or 1).

2.2 Rowhammer

Since DRAM chips have been compressed remarkably to increase the capacity within limited space, many hardware faults have emerged [7–9]. One of the hardware faults is a disturbance error which hazards data integrity due to interferences of adjacent cells.

Rowhammer is a method to intentionally induce the disturbance error. Rowhammer is performed by repeatedly and rapidly accessing a DRAM row (called an aggressor row). This process accelerates the effect of the disturbance error, that corrupts the neighboring rows (called target rows or victim rows) of the aggressor row.

Several effective methods of rowhammer attacks have been introduced such as double-sided rowhammer [10], data pattern [9, 11] and so on. These effective methods induce more bit flips, which allows attackers to exploit more bit flips to more precisely corrupt data. One of the effective methods is double-sided rowhammer. Double-sided rowhammer repeatedly accesses both upper and

lower neighboring rows of a target row, while the single-sided rowhammer repeatedly accesses only a single neighboring row to induce bit flips on the target row. To corrupt target data by using double-sided rowhammer (or single-sided rowhammer) exquisitely, it is necessary to know the user-level address (i.e., virtual address) of upper and lower neighboring rows in the same bank. Previous work [12,13] has introduced the methods to detect the virtual addresses that correspond to the physical addresses of potential neighboring rows. However, since these methods are performed with the unproven assumption that contiguous rows have contiguous physical addresses, the potential neighboring rows may not be actual neighboring rows. Therefore, to precisely corrupt the target row, attackers need to know the exact physical addresses corresponding to the neighboring rows in the same bank.

2.3 Mapping Algorithm

To access data on DRAM module, the memory controller locates the data on DRAM by referring to mapping algorithm. The mapping algorithm determines the location of data on DRAM and is composed of several physical address bits. Since it must map the physical address to the hierarchies of DRAM, the mapping algorithm consists of sub-mapping algorithms for channels, modules, ranks, banks, and rows.

The mapping algorithms are required for rowhammer attacks or even rowhammer mitigation methods. Rowhammer attacks must repeatedly access the neighboring rows of victim row. To access the neighboring rows in the same rows with a victim row, attackers must know the logical address of the neighboring rows. Therefore, the attackers have to know the mapping algorithm for rows and banks, which is the relation between physical row/bank indexes and physical addresses. However, the mapping algorithm is not disclosed in detail by manufacturers.

III. Related Work

3.1 Mapping Algorithm in Prior Work

Mapping algorithm is partially reverse engineered in previous work [1, 2]. The researches reveal the mapping algorithm for banks by using timing method which is related to row buffer. Successive accesses to two rows in the same bank flush the voltage stored in the row buffer. To store the voltage of the subsequent accessed row, the voltage of the prior accessed row is flushed. This additional step (flushing the voltage) makes additional overhead, so successive accesses to two rows which are located in the same bank takes longer time than accesses to two rows which are located in different banks. According to the timing difference, it is possible to determine whether two rows which is successively accessed are located in the same bank or different banks. Therefore, the mapping algorithm for banks can be reverse engineered by using this method.

In contrast, the mapping algorithm for rows cannot be reverse engineered with the method which is used in the work. While it is possible to find which physical address bits compose the mapping algorithm for rows, it is impossible to identify how the composed bits of physical address determine the physical location of rows. This means that it is possible to distinguish whether the row of a certain physical address is different from the rowhammer target row in the same bank, but it is impossible to identify the exact upper and lower rows of the target row.

Without the mapping algorithm for rows, the sophisticated rowhammer attack is difficult because it is difficult to locate the neighboring rows of a victim row. Prior work assumes that the physical contiguous rows have contiguous phys-

ical addresses, but this assumption is not validated experimentally.

3.2 Rowhammer Attack

Rowhammer can be used by corrupting inaccessible data whose data integrity must be guaranteed. The representative rowhammer attack is to corrupt page table entries [2, 10, 12]. By changing the physical addresses of page table entries, the attackers can access the originally inaccessible data and code which are located in the changed physical addresses. [10] also corrupts code to change the code into malicious code.

[14] exploits rowhammer attack to access the inaccessible data in Microsoft Edge browsers. This work corrupts type data and change attacker-controlled double number in to a reference that points to an attacker-controlled object. Especially, this work locates the target data in memory space they want by using deduplication technique.

Also, [13] exploits deduplcation technique to locate target data. This work focuses on the fact that the factorization of the cryptographic procedure is much easier, even if only a few bits of the RSA key are changed by brute-force attacks. Therefore, the Rowhammer attack in this work corrupts the random bits in RSA key and makes the factorization procedure easy to breaks the cryptographic mechanism.

[12] additionally targets ARM and Android. This work analyzes the precise mechanism of the buddy allocator which is used in ARM and Android, and exploits the mechanism to locate the target data. Also, instead of the existing cache bypass method that is unusable in Android and ARM, they access DRAM directly by using DMA buffer management APIs provided by OS.

[15] analyzes most existing mitigations and introduces new methods to bypass the existing mitigations. For example, the attack method in this work

corrupts the sudo binary code located in user space to bypass isolation mechanisms [5]. In addition, this work uses new attack vector such as one-location hammering, memory waylaing, and enclave features.

However, these attack methods need enough exploitable bit flips. [12] states that the attack would be successful if a bit of about $5\% \sim 7\%$ was flipped. Also, [15] mentions that in the shared library *sudoers.so* binary code, only 28 bits flips are exploitable. Therefore, as many bit flips as possible are necessary to make a successful attack.

3.3 Existing Mitigation

Existing mitigation methods can be divided into hardware-based mitigation and software-based mitigation. Hardware-based mitigation includes reducing refresh rate, target row refresh (TRR) and so on. Reducing refresh rate [9] allows refreshes to occur before the row voltage drops below the threshold, thereby preventing corrupting data with the rowhammer attacks. TRR [16] counts the number of row activation and then refresh the potential victim rows when the activation number exceeds the specific threshold.

Software-based mitigation is as follows. Existing static analysis mitigation [17] can recognize the microarchitectural attack in advance at the code level, which may detect rowhammer attacks. There is also a mitigation method based on performance counter [18, 19]. Rowhammer must bypass cache because it has to access directly to DRAM repeatedly. This cache bypass causes cache misses and performance counter based mitigation can detect cache misses, which can detect rowhammer attacks.

Additionally, mitigation based on physical row proximity can defend the rowhammer attacks. This method is based on the fact that the rowhammer attack corrupts only the rows that are adjacent to the aggressor rows. Anvil [4],

for example, detect cache misses by using performance counters and recognizes the potential victim row. After that, Anvil properly refreshes the recognized victim row to prevent disturbance errors. G-CATT [5] recognizes potential rows that may be problematic and prevents the neighbor rows from being accessed by the normal user. Therefore, the attacker cannot use the desired row as an aggressor row and thus cannot corrupt the target data.

The mapping algorithm for rows is required for defense based on physical row proximity. This is because it is necessary to know precisely what neighboring row of target rows is for an arbitrary row. If the mitigation methods do not know the exact location of the neighboring rows, they will not prevent the rowhammer attacks properly, and the attacker may break the integrity of the target data.

IV. Reverse Engineering Mapping Algorithm

While the mapping algorithm for banks is reverse engineered, the mapping algorithm for rows cannot be reverse engineered using the timing method. Therefore, previous work [2,4,5] has naïve assumption that physical rows are arranged identically to physical addresses. According to the prior assumption, two rows are physically contiguous if the physical addresses of these rows are contiguous in the same bank.

Let r represent a row address composed of $\{b_0, b_1, \ldots, b_n\}$, b_i be the i^{th} row bit, where n is the number of bits on a logical row address. The row index function on prior assumption is:

$$LRow(r) = \sum_{i=0}^{n} 2^i b_i, \tag{4.1}$$

where r is a row address. These bits of a row address can be extracted from a physical address by using the previous methods [1,2]. We call this function (4.1) a logical row index function in this thesis. Previous work has not demonstrated that the logical row index function is used in the real architecture.

The goal of this chapter is to find the real mapping algorithm. We can express the mapping algorithm as:

$$PRow(r) = \sum_{i=0}^{n} 2^i F_i(r) \tag{4.2}$$

We call this function (4.2) a physical row index function in this thesis and the goal of this chapter is to find the $F_i(r)$ that is assumed to consist of physical address bits.

Before explaining our method in detail, we assume that 512 contiguous logical

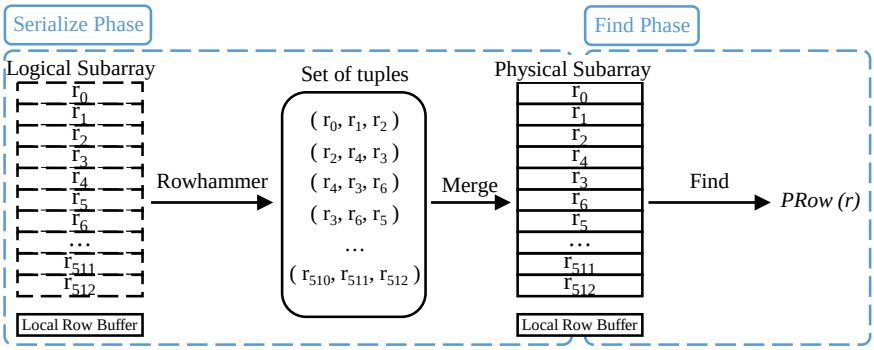


Figure 4.1: Overall procedure of reverse engineering.

rows constitute one physical subarray and the mapping algorithm of one DRAM subarray also works for all DRAM subarrays. These assumptions seem reasonable due to the regularity of DRAM subarrays [8]. Therefore, we only focus on the lower 9 bits of a row address because each DRAM subarray consists of 512 ($= 2^9$) rows.

We reverse engineer the mapping algorithm in two steps. First, we arrange DRAM rows in sequential order by using rowhammer. We refer to this step as row serialization. Second, we manually find the mapping algorithm with the serialized rows, that is finding the function $F_i(r)$. The overall procedure of reverse engineering is shown in Fig. 4.1.

4.1 Row Serialization

In this chapter, we describe how to serialize rows of the single subarray in physical order to find the exact mapping algorithm. The key idea of row serialization is that *rowhammer induces bit flips only on the neighboring rows of an aggressor row*. We can identify the neighboring rows of aggressor rows by detecting the rows that have been corrupted by the rowhammer.

For convenience, we first define the following primitives:

- r_n is a row address which is composed of $\{b_0, b_1, b_2, ..., b_i\}$, where i is the

number of bits on row address r_n.

- $sr_{n,m}$ is a row that is sandwiched between rows r_n and r_m.
- $Dhammer(r_n, r_m)$ returns the set of the rows that have been corrupted by double-sided rowhammer when r_n and r_m are aggressor rows.
- $Shammer(r)$ returns the set of the rows that have been corrupted by single-sided rowhammer when r is an aggressor row.

We perform double-sided rowhammer with all the combinations of two rows within a single DRAM subarray to find a row that is sandwiched between two rows. A single DRAM subarray consists of 512 rows, so the number of cases is at most $\binom{512}{2}$ = 130,816. Thus, we argue that the time required to perform the rowhammer with two selected rows is feasible. We check which rows have been corrupted by double-sided rowhammer on the selected rows r_n and r_m. By this process, we can find the set of $Dhammer(r_n, r_m)$, which is related to $sr_{n,m}$. However, the set of $Dhammer(r_n, r_m)$ may contain rows that are adjacent to the other sides of the aggressor rows in addition to the sandwiched rows due to the effects of single-sided rowhammer. Therefore, we remove the effect of single-sided rowhammer by excluding the sets of $Shammer(r_n)$ and $Shammer(r_m)$ from $Dhammer(r_n, r_m)$ by performing additional single-sided rowhammer on r_m and r_n.

Ideally, $Dhammer(r_n, r_m) - Shammer(r_n) - Shammer(r_m)$ must have only one row $sr_{n,m}$ because only one sandwiched row can be between two aggressor rows. Therefore, except for two boundary rows (the 0th row and the 512th row) which are not affected by double-sided rowhammer, 510 tuples $(r_n, sr_{n,m}, r_m)$ can be obtained within a single DRAM subarray. This tuple indicates a sequence of physically contiguous rows. After obtaining the tuple, we can infer overall row arrangements due to the subarray regularity.

With the tuples, we can reconstruct a subarray by using the following rules:

- If two tuples, $(r_{n_1}, sr_{n_1,m_1}, r_{m_1})$ and $(r_{n_2}, sr_{n_2,m_2}, r_{m_2})$ exist such that $sr_{n_1,m_1} = r_{n_2}$ and $r_{m_1} = sr_{n_2,m_2}$, then the two tuples can be merged into $(r_{n_1}, r_{n_2}, r_{m_1}, r_{m_2})$.
- If two tuples, $(r_{n_3}, sr_{n_3,m_3}, r_{m_3})$ and $(r_{n_4}, sr_{n_4,m_4}, r_{m_4})$ exist such that $r_{m_3} = r_{n_4}$, then the two tuples can be merged into $(r_{n_3}, sr_{n_3,m_3}, r_{m_3}, sr_{n_4,m_4}, r_{m_4})$.

Using this method, we can ideally merge all of the tuples into one sequentially ordered tuple, that is, the physical sequence of 512 rows within a single DRAM subarray.

The success of row serialization depends on the number of corrupted rows after performing double-sided rowhammer. However, some rows might not be corrupted. To resolve this problem, we focus on the subarray regularity, which means that every subarray is organized identically. Due to this regularity, undiscovered tuples in one subarray can be obtained from another subarray.

4.2 Finding Mapping Algorithm

Next, we use the serialized rows to find the mapping algorithm of physical row indexes $PRow(r_n)$ (or $F_i(r)$). As we mentioned, the input of this function (4.2) is a row address and the output is a physical row index. We manually find the function $PRow(r)$ by exploiting the insight that the discovered tuple $(r_n, sr_{n,m}, r_m)$ implies $PRow(r_n)+1 = PRow(sr_{n,m})$ and $PRow(r_m)-1 = PRow(sr_{n,m})$, which means that the rows of the tuple are sequentially contiguous.

4.3 Row Address Mapping Schemes

We tested six DDR3 modules of three major DRAM manufacturers with this method to reverse engineer the mapping algorithm (Table 4.1). By using this method, we discovered two mapping algorithm schemes, which are address

mirroring and row twisting. As a result, we demonstrate that the prior assumption is faulty.

4.3.1 Address Mirroring.

The first scheme is observed only on one rank[1] of two-rank DRAM modules for all of the experimented manufacturers. The other rank is subject to either the logical row index function or row twisting. The physical row index function is (we mark the components which are different from the logical row index function in bold.):

i	0	1	2	3	4	5	6	7	8
$F_i(r)$	b_0	b_1	b_2	$\mathbf{b_4}$	$\mathbf{b_3}$	$\mathbf{b_6}$	$\mathbf{b_5}$	$\mathbf{b_8}$	$\mathbf{b_7}$

JEDEC [3] documents this scheme, called address mirroring that is deployed to increase throughput by cross-wiring some wires on one rank. We can identify that our discovered physical row index is address mirroring because the index of address mirroring in the JEDEC standard is exactly identical with ours. The sameness implies that our method to reverse engineer is valid to find the exact physical address.

4.3.2 Row Twisting.

The second scheme is observed only on DRAMs from A manufacturer, and we call the scheme row twisting. The physical row index function is (we mark the components which are different from the logical row index function in bold.):

[1]In fact, it is impossible to distinguish the rank bits and bank bits from the existing reverse engineering method. However, since we can track the difference of mapping algorithm for each rank, we can infer which bit of the bank bits is a rank bit.

Table 4.1: Address mirroring and row twisting on DDR3 modules of three major manufacturers.

Tag	Manufacturer	Model	# of ranks	Mirroring	Twisting
A	SAMSUNG	M378B5273DH0-CK0	2	✓	✓
B		M378B5273DH0-CH9	2	✓	✓
C		M378B5173QH0-CK0	1		✓
D	HYNIX	HMT351U6CFR8C-PB	2	✓	
E		HMT351U6CFR8C-H9	2	✓	
F	MICRON	MT16JTF51264AZ-1G6M1*	2	✓	

*In this case, there are not enough bit flips to reverse engineer the mapping algorithm. However, since some bit flips are induced in rowhammer experiments with only address mirroring, we infer that this module is affected only by address mirroring.

i	0	1	2	3	4	5	6	7	8
$F_i(r)$	b_0	$\mathbf{b_1} \oplus \mathbf{b_3}$	$\mathbf{b_2} \oplus \mathbf{b_3}$	b_3	b_4	b_5	b_6	b_7	b_8

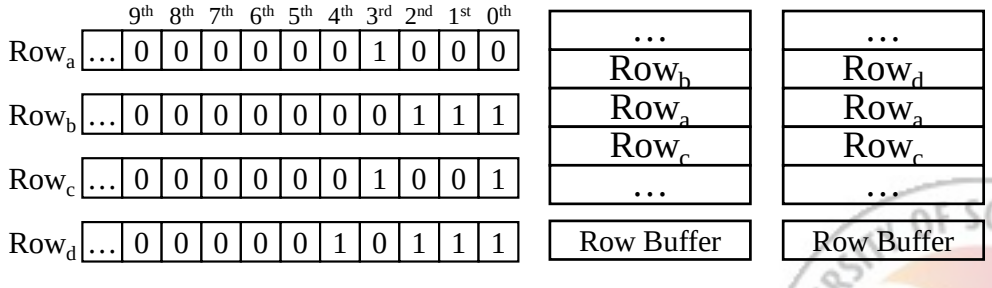
According to our experimental result, the row twisting function is applied after address mirroring. For example, when a module is affected by both row twisting and address mirroring, the bit swap of b_3 and b_4 is first performed (due to address mirroring) and the swapped b_3 (originally b_4) is XORed with b_2 and b_1 later (due to row twisting). Therefore, we infer that row twisting is an internal mechanism of DRAM chip unlike address mirroring which is a mechanism at a circuit level. We also infer that row twisting is somehow related to twisted wordline schemes [20, 21].

V. Mapping Algorithm-Aware Rowhammer

If attackers know the exact mapping algorithm for rows, the attackers can perform more effective rowhammer attack by inducing more exploitable bit flips. In addition, attackers can exquisitely perform rowhammer attacks because the attackers can locate the exact upper/lower neighboring rows of the target row. We refer to this method as mapping algorithm-aware rowhammer.

We give an example of rows that are vulnerable to rowhammer attacks (Fig. 5.1). For simplicity, we assume that the DRAM module is affected by only address mirroring. When the faulty assumption is applied, the upper row is Row_b and the lower row is Row_c with respect to Row_a (Fig. 5.1b). However, the upper row of Row_a in address mirroring is actually Row_d, not Row_b (Fig. 5.1c). The logical row indexes of Row_a, Row_b, Row_c and Row_d are 8, 7, 9 and 23, but their physical row indexes are 16, 7, 17 and 15, respectively.

We evaluate the effectiveness of mapping algorithm-aware rowhammer. To evaluate the effectiveness, we measured the number of bit flips that were caused by double-sided rowhammer. We exploited three kinds of double-sided rowhammer:



(a) Vulnerable rows (b) Logical (c) Physical

Figure 5.1: Vulnerable rows on address mirroring. The upper row in logical index (b) and the upper row in physical index (c) is different about Row_a.

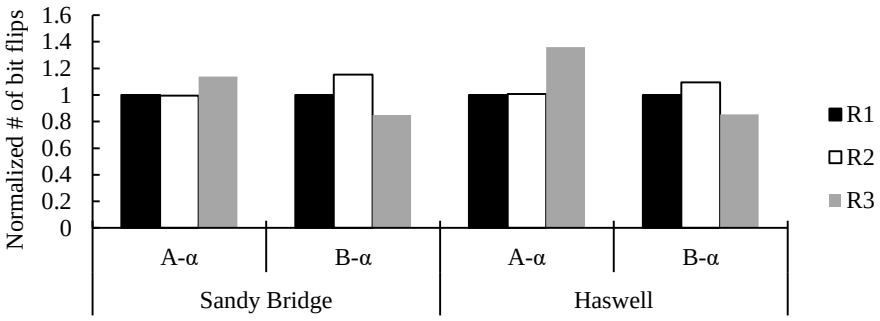


Figure 5.2: Normalized number of bit flips. The number of bit flips for each case is normalized to R1.

(i) R1 is performed with the assumption that physically contiguous rows have contiguous physical addresses.

(ii) R2 is performed with the assumption that row indexes are affected only by address mirroring.

(iii) R3 is performed with the assumption that row indexes are affected by both address mirroring and row twisting.

We performed these three types of rowhammer on the memory space that is different from the subarray used in the experiment of Table 4.1. We tested on two modules: one was inferred to be affected only by address mirroring (module D); the other was inferred to be affected by both address mirroring and row twisting (module A) by our proposed method. We measured the number of bit flips on two processors, Sandy Bridge (i7-2600) and Haswell (i5-4460).

Although the prior assumption is faulty, R1 showed considerable bit flips (Fig. 5.2). The abnormal result is because some physical rows are sequential like logical rows, or single-sided rowhammer has shown an effect even if the attack uses double-sided rowhammer. However, module A shows the largest number of bit flips by R3 (25% more bit flips than R1 on average), and module D shows the largest number of bit flips by R2 (12% more bit flips than R1 on average). This

experimental result shows that the proposed attack method, mapping algorithm-aware rowhammer, is more efficient than the conventional method regardless of processor types.

Also, this experimental result validates that our discovered mapping algorithm is correct.

VI. Improved Mitigation

In this chapter, we explain existing mitigations and how the mitigations can be attacked by rowhammer attacks. Then, we improve Anvil, one of the existing mitigation methods, and evaluate its effectiveness as a proof-of-concept.

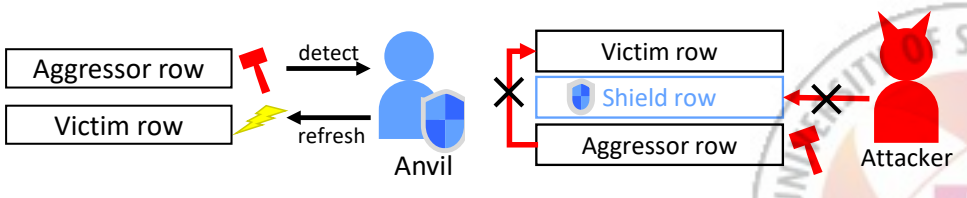
6.1 Existing Mitigations and a Security Hole

We study two representative mitigations, Anvil [4] and G-CATT [5], and explain how they are affected negatively by the faulty assumption using the aforementioned example of the vulnerable row (Fig. 5.1).

6.1.1 Anvil.

Anvil (Fig. 6.1a) defends rowhammer attacks by refreshing potential victim rows when the signs of rowhammer attacks are detected. The potential victim rows are the neighboring rows of the detected aggressor rows and thus Anvil refreshes the presumed neighboring rows.

However, according to the faulty assumption of Anvil, Anvil regards Row_b and Row_c as potential victim rows when Row_a is the aggressor row (Fig. 5.1b). Therefore, if attackers repeatedly access the aggressor row Row_a, bit flips may occur on Row_d, which was not refreshed by Anvil. As a result, Anvil can detect



(a) Anvil (b) G-CATT

Figure 6.1: Two representative mitigations, Anvil and G-CATT.

rowhammer attacks but cannot refresh all the victim rows.

6.1.2 G-CATT.

G-CATT (Fig. 6.1b) defends rowhammer attacks by putting a row between rows of different security domains (e.g., kernel and user) to physically separate the rows. Since rowhammer attacks only corrupt rows that are adjacent to aggressor rows, attackers cannot corrupt victim memory on G-CATT.

However, according to the faulty assumption of G-CATT, G-CATT expects that Row_a and Row_d are already separated (Fig. 5.1b). Therefore, if an attacker process is allocated in Row_d, Row_a is vulnerable to rowhammer attacks.

6.2 Method to Improve Existing Mitigations

To perfectly defend against rowhammer attacks, the existing mitigations must be improved with the proper physical row indexes. There are two methods to improve the existing mitigations with our discovered mapping algorithm.

The first method regards all possible neighboring rows of each and every case of the mapping algorithm schemes as the actual neighboring rows. This method takes additional runtime overhead to manage additional rows but our method to find the mapping algorithm is not required. The second method finds the mapping algorithm by using our proposed method before performing the existing mitigations. This method is only possible when the victim's DRAM modules are vulnerable enough to be able to be reverse engineered and initial time to find the mapping algorithm. However, this method does not need an additional runtime overhead compared to the first method.

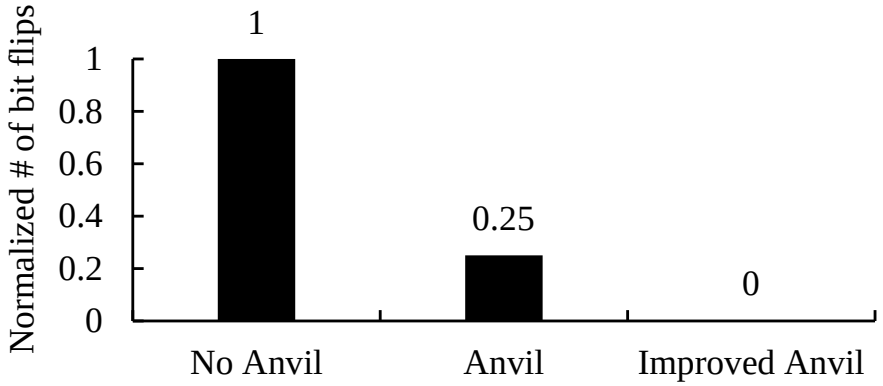


Figure 6.2: Normalized number of bit flips with module A on Sandy Bridge.

6.3 Evaluation

We measured the bit flips by rowhammer on module A to evaluate the effectiveness of the improved mitigation that considers the exact mapping algorithm schemes. First, to show the security hole of existing mitigations, we performed single-sided rowhammer on Anvil with only the mapping algorithm for banks. Next, we improved Anvil into two cases: one is improved by considering only address mirroring, and the other is improved by considering both address mirroring and row twisting. We perform single-sided rowhammer again on those two improved Anvil methods (Fig. 6.2).

Anvil and Anvil with only mirroring might refresh actual victim rows because some refreshed rows are actual victim rows in spite of the faulty information about the neighboring rows. However, not all potential victim rows are actual victim rows, and thus Anvil and Anvil with only mirroring cannot refresh all of the actual victim rows. Fully improved Anvil, which is modified by using our mapping algorithm, shows no bit flips. This result shows that the fully improved mitigation can effectively defend against rowhammer attacks. In addition, we believe that G-CATT also properly prevents rowhammer attacks if the exact mapping algorithm is applied to G-CATT.

VII. Discussion

In this chapter, we discuss limitations of our method to reverse engineer mapping algorithm and future works to solve the limitations.

7.1 Limitations

One limitation of our work is that enough bit flips are required to reverse engineer the mapping algorithm for rows. Our proposed method reverse engineer the mapping algorithm for rows by inducing the bit flips caused by double sided rowhammer. If our proposed method cannot induce a certain DRAM module which we want to reverse engineer, it is difficult to analyze the relation between physical addresses and physical row indexes.

7.2 Future Works

To induce enough bit flips for detecting mapping algorithms, we can use several methods. First method is to consider the data pattern of aggressor rows and victim rows. Prior work [9, 11] shows certain data pattern can induce more bit flips by using rowhammer methods. Second method is increasing the refresh interval. If the refresh interval is increased, rowhammer methods can drop the data voltage below the threshold (the indicator to determine whether the data is 0 or 1) during the longer period. Therefore, rowhammer methods induce more bit flips in the system whose the refresh interval is longer.[1]

[1]Refresh interval can be increased at the system level, so this method is proper for the improved mitigation, not for the attackers at the user level.

VIII. Conclusion

We introduced a method to reverse engineer the mapping algorithm for rows and revealed an exact mapping algorithm for rows. This method uses the feature that rowhammer induces bit flips only on the neighboring rows of aggressor rows. Using this method, we can infer the exact row arrangement. As a result, we demonstrate that previous work has faulty assumption that physically contiguous rows have also contiguous physical addresses.

Based on the exact physical row index, we can induce bit flips more efficiently than the conventional rowhammer method which does not consider the exact physical row index. Note that more bit flips make the rowhammer attack more successful because there are more candidates of exploitable bit flips. Also, we have shown that it is possible to make bit flips in the system with rowhammer mitigation methods if they were used with the faulty assumption. Finally, we have improved the existing mitigation and showed that the improved mitigation perfectly protects against rowhammer attacks.

요 약 문

컴퓨터 시스템에서 memory로 사용되는 하드웨어인 DRAM은 전기적 방법을 이용해 정보를 저장한다. 이런 전기적 방법은 매우 민감하여 가까운 cell의 자극에 의해 값이 변화할 수도 있다. 이러한 하드웨어의 오류를 인위적으로 발생시키는 로우해머 공격 방법은 피해자의 데이터 무결성을 헤친다. 이전 연구에서 많은 로우해머 공격 방법과 그에 대응하는 방어 기법들이 소개됐지만, 실제 DRAM의 정확한 구조가 밝혀지진 않았으며, naive한 가정을 가지고 실험을 진행하였다. 본 논문은 DRAM의 정확한 구조를 reverse engineer하고 이것이 기존 공격방법과 방어기법에 어떠한 영향을 줄 수 있는지, 또 어떻게 기존 방법들이 발전할 수 있는지 소개하고 이를 수치화했다.

References

[1] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. Drama: Exploiting dram addressing for cross-cpu attacks. In *USENIX Security Symposium*, pages 565–581, 2016.

[2] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *USENIX Security Symposium*, pages 19–35, 2016.

[3] JEDEC. *DDR3 SDRAM Unbuffered DIMM Design Specification*, 2013. Rev. 1.06.

[4] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. Anvil: Software-based protection against next-generation rowhammer attacks. *ACM SIGPLAN Notices*, 51(4):743–755, 2016.

[5] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. Can't touch this: Software-only mitigation against rowhammer attacks targeting kernel memory. In *Proceedings of the 26th USENIX Security Symposium (Security). Vancouver, BC, Canada*, 2017.

[6] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A case for exploiting subarray-level parallelism (salp) in dram. *ACM SIGARCH Computer Architecture News*, 40(3):368–379, 2012.

[7] Robert Baumann. The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction. In *Electron Devices Meeting, 2002. IEDM'02. International*, pages 329–332. IEEE, 2002.

[8] Samira Khan, Donghyuk Lee, and Onur Mutlu. Parbor: An efficient system-level technique to detect data-dependent failures in dram. In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pages 239–250. IEEE, 2016.

[9] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 361–372. IEEE Press, 2014.

[10] Mark Seaborn and Thomas Dullien. *Exploiting the DRAM rowhammer bug to gain kernel privileges*, 2015.

[11] Mark Lanteigne. *How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware*. Third I/O Incorporated, March 2016.

[12] Victor Van Der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1675–1689. ACM, 2016.

[13] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *USENIX Security Symposium*, pages 1–18, 2016.

[14] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE symposium on security and privacy (SP)*, pages 987–1004. IEEE, 2016.

[15] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. *arXiv preprint arXiv:1710.00551*, 2017.

[16] JEDEC Solid State Technology Association. *Low Power Double Data Rate 4*, 2017.

[17] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Mascat: Stopping microarchitectural attacks before execution. *IACR Cryptology ePrint Archive*, 2016:1196, 2016.

[18] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing*, 49:1162–1174, 2016.

[19] Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 118–140. Springer, 2016.

[20] Dong-Sun Min and Dietrich W Langer. Twisted line techniques for multi-gigabit dynamic random access memories, March 7 2000. US Patent 6,034,879.

[21] Dong-Sun Min, Dong-Il Seo, Jehwan You, Sooin Cho, Daeje Chin, and YE Park. Wordline coupling noise reduction techniques for scaled drams. In *VLSI Circuits, 1990. Digest of Technical Papers., 1990 Symposium on*, pages 81–82. IEEE, 1990.

Acknowledgements

석사 과정 동안 올바른 연구 방향을 가르쳐 주신 김종 교수님께 감사드립니다. 교수님의 좋은 지도로 석사 과정을 잘 마무리 할 수 있었습니다. 또한 연구적으로나 정서적으로나 많은 도움을 준 연구실 동료들에게도 감사드립니다. 선후배님들의 지원이 있었기 때문에 많은 어려움을 이겨낸 것 같습니다. 마지막으로 힘들때나 기쁠때나 항상 응원해준 가족들에게 감사한 마음을 전합니다.

Curriculum Vitae

Name : Saeyoung Oh

Education

2012. 3. – 2017. 2. Department of Computer Science and Engineering, Pohang University of Science and Technology (B.S.)

2017. 3. – 2019. 2. Department of Computer Science and Engineering, Pohang University of Science and Technology (M.S.)


POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY
1986
포항공과대학교