DRAM Security: When Hardware Faults Become Security Issues

Alirad Malek

Computer Science and Engineering, Chalmers University

Abstract. Thanks to advances in semiconductor industry, DRAM cells have scaled down to feature sizes of a few nanometers, bringing great opportunities and new challenges to the design of future memory systems. On the one hand, we can have higher memory capacity on the same silicon area, on the other hand shrinking size of transistors and capacitors increases variability and affects the reliability of DRAM chips. One of the fault models which is highly associated with technology scaling in DRAMs in disturbance errors. In 2014, an empirical study of off-the-shelf DRAMs revealed that by repeatedly accessing a row within DRAM, it is possible to force disturbance errors in significant numbers of cells in the neighbouring rows [4]. This phenomena, which later became known as *RowHammer* problem, was initially perceived as reliability issue. However, in a short while it became evident that the RowHammer effect could possibly be a major security vulnerability. Over the last couple of years, many researchers have proposed exploiting RowHammer in order to gain kernel privilege on real systems, takeover a virtual machine in shared environment or using a malicious mobile application to gain root access. In this short survey, we will have an overview of RowHammer problem and how it could be exploited as a security vulnerability.

1 Background

Dynamic Random-Access Memory (DRAM) is the prominent choice for main memory in current computers systems, due to its simple structure and relatively cheap fabrication process [3].

As illustrated in Figure1, the basic storage cell in DRAMs consist of one transistor and one capacitor (T1C1 cell). Each DRAM module has multiple devices and each device can have several banks. In a DRAM bank, cells are organized in rows and columns making a cell-array, which means cells within a row share word-line, and cells within a column share a bit-line. At the bottom of each bit-line, there is a sense amplifier circuitry which is connected to a reference voltage. To facilitated the sense amplifiers task, the bit-lines are precharged to half-way voltage. Figure1 is an abstract view of a bank, showing the mechanism of reading a row by activating the word-line.

DRAM cells can not retain the data for a long time due to leakage. Hence, there is a refresh mechanism in the DRAM to mitigate this short retention-time by periodically restoring the data on each cell.

Reads and writes to the DRAM are managed by a *memory controller*, conventionally placed on the processor side[1]. Concisely, a DRAM memory access takes place in three phases: (i)**ACTIVATION** The memory controller sends the address of the requested cache line[2] to the DRAM module. In the DRAM module, the row decoder extracts the row address of the requested data and puts a high voltage on the word-line of the requested row on all devices.

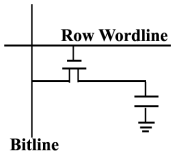

Fig. 1. One-Transistor, One-capacitor DRAM cell. The data is stored as capacitor's charge. A high-voltage on the word-line will drive the data to the bit-line. The bits-line of all cells in one array are connected to the sense amplifier. Source [4] .

Consequently, the data of all cells on that row are sensed by the sense amplifiers at the bottom of the bit-lines and are placed on a row-buffer at the bottom of the cell-array. Effectively, at this point the value of the cells are restored (*i.e.* refreshed) to their original full charge. (ii) **READ/WRITE** The column decoder extracts the requested columns from the address and depending on the type of access, i.e. read or write, some bits on the row buffer are either sent to the output port or are updated with new values.(ii)**PRECHARGE** the access is closed by setting the bit-line voltages to half-way and flushing the row-buffer. The time that each of the steps in these phases take is conventionally called DRAM timing constraints and they vary between different DRAM generations and manufacturers. The described phases and timing constrains make sure that DRAM accesses are stable in two sense: 1- Any read access should not modify any data in the memory. 2- Any write access should only change the requested address.

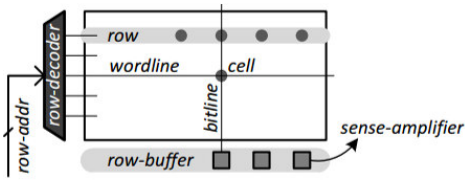

Fig. 2. Structure of a DRAM bank. Source [3]

.

[1] In new emerging 3D-stacked DRAMs the controller is placed on a logic die at the bottom of the stack [6]

[2] Essentially triggering memory access means the requested data doesn't exist on the cache hierarchy.

2 Flipping Bits Through RowHammer

As mentioned before, due to limited retention-time, DRAM cells need frequent refreshing. In DDR3 modules, each row is guaranteed to be refreshed every 64ms. In other words, the data on the cells are untouched in refresh intervals, unless they are accessed. In 2014, Kim et al. showed that by repeatedly accessing a same row within a bank (i.e. hammering a row), some cells in the neighboring rows might encounter a bit-flip, within their refresh interval [4]. This observation can be explained by the fact that toggling the word-line voltage of one row will induce disturbance errors, affecting vulnerable cells in mostly the two immediate neighboring rows, as shown in Figure 3. To show the extent of this problem, Kim et al. conducted an empirical study in which a simple malicious code was used to systematically hammer rows on 129 DRAM modules. The results showed that 110 out of 129 DRAMs encountered at least a single bit-flip and it takes as few as 139k accesses for a hammered row to cause this. In addition, they showed that more recent DRAMs are more susceptible to rowHammer effect, which indicates that the problem is exacerbated in modern DRAM with smaller feature sizes.

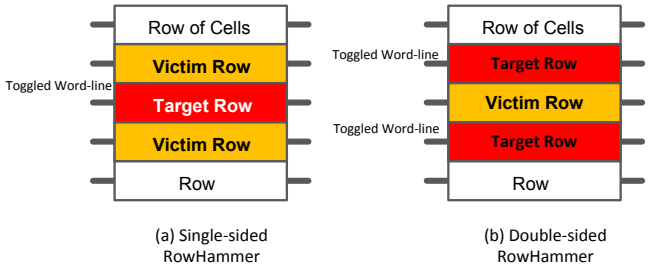

Fig. 3. RowHammer. Toggling the word-line of a target row might induce bit-flips on the victim rows. (a) Single-sided rowHammer. (b) Double-sided rowHammer.

3 Exploiting RowHammer As A Security Vulnerability

As explained, rowHammer is a phenomena in which multiple accesses to a single row in short time, will cause random bits to flips on vulnerable cells of neighboring rows. On its own, rowhammer can be used as a disturbance attack, causing erroneous results or even system crash. However, there are ways to exploit rowHammer for more directed attacks.

Main memory is most often shared between different processes with different permissions, or between virtual machines running on a same system. Moreover, although the operating system (OS) allocates separate virtual pages to each process, the physical location of these virtual pages are not contiguous in the DRAM and consecutive rows in a DRAM bank are used to store pages of different processes. The physical location of memory addresses[3], alongside their permissions[4], are saved in a Page Table (PT) which is stored in the same DRAM

[3] Translation from virtual to physical address.
[4] Per page.

name-space. Based on these facts, there have been many proposal which exploit rowHammer for manipulating the page table and gain kernel privilege [8], take over a server by using JavaScript [2] or taking over a virtual machine(VM) [7] via an attacker controlled VM.

In general, security researchers have considered three main criteria which makes it possible to have rowHammer assisted attacks: (i) the attacker should be able to repeatedly access a row with high enough speed, (ii) the attacker should force the OS to put sensitive data (e.g. page table) on the rows with vulnerable cells, and (iii) the attacker should have information about exact physical mapping of rows to have a selective attack. These criteria are addressed in different ways, based on the target architecture and underlying system.

Fast Access to the row A closer look at the rowHammer effect reveals that there are two conditions which are essential for bit-flips to happen: (1)The target row should be access very fast. (2) The requested address should not be residing on the cache hierarchy, which means every request for the target address should issue a memory access. While the first condition is relatively easy on systems with high performance memory controllers, it is still a challenge to make sure caches are bypassed in every access. Conventional workarounds for this problem is to explicitly flush the cache after each request, accessing addresses with the same cache eviction sets (forcing the cache to evict) or using CPU instructions like *MOVNTI* which trigger non-temporal accesses. In addition to being architecture specific, non-temporal access instructions are just used as a hint for the processor, and are not strictly followed, thus this solution is not as effective as the other two.

Placing sensitive data The most challenging criterion is perhaps tricking or forcing the OS to place the sensitive data on the vulnerable rows inside DRAM space. The state-of-art techniques for this task are mainly based on probabilistic approaches (*e.g.* page table spraying) or rely on architecture specific features like memory duplication and memory management unit (MMU) paravirtualization. Recently a new technique called *Phys Feng Shui* has been presented which relies solely on predictable memory reuse patterns of conventional memory allocators [7]. In a standard memory allocator, when a memory allocation request is being served, the smallest fitting chunk of the memory is priorities to be allocated instead of splitting a bigger size chunk into segments. This feature is originally implemented to reduce internal memory fragmenting. In essence, *Phys Feng Shui* restricts allocator's memory chunk options, forcing it to place the page table in a vulnerable physical location. In addition to being deterministic, this technique can be used to have double-sided rowHammer attack, where the rows above and below a vulnerable row are accessed repeatedly to induce bit-flip in the victim row in the middle.

Physical DRAM addressing Virtual addresses in the OS are translated into physical addresses and stored inside the page table. The physical addresses are

then mapped to DRAM ranks, bank, rows and columns based on the memory system configuration. The processor is oblivious of physical addresses of pages on the DRAM and this information is restricted to the memory system. In order to have a directed attack, which does not randomly crash the system, the attacker needs to know the physical address of each page. One straight workaround would be to use *pagemap* interface to examine the page table and related mappings. However, in recent kernel versions this interface requires administrative privilege. Another way is to increase the granularity of virtual pages to 2MB (instead of usual 4KB) which essentially covers 2MB of contiguous physical addresses.

4 Deterministic RowHammer attack on mobile platform

As mentioned before, most of the proposed rowHammer assisted attacks on the systems are probabilistic. In addition, these attacks were mainly targeting x86 architecture, where many of the described solutions in previous section are available. However, rowHammer assisted attacks are shown to be possible in other systems like mobile phones. Drammer proposed a deterministic attack which exploits rowHammer to gain root privilege through an unprivileged Android application on ARM-based mobile devices [9]. Drammer capitalize on existence of Direct memory access (DMA) buffers in mobiles devices. DMA allows certain hardware modules to directly access the memory module and bypass the processor, in order to enhance memory access speed. Effectively this means cache hierarchy will not be an issue anymore (criterion 1). Also, DMA needs to work with physical addresses which nullifies the need for having information about physical DRAM addressing (criterion 2). As for the sensitive data placement, Drammer employs the same technique as *Phys Feng Shui*, thus offering a deterministic, double-sided rowHammer attack.

5 Counter Measurements Against RowHammer

Kim et al. has proposed several ways to protect DRAM against rowHammer problem [4]. Since rowHammer is in nature a disturbance error, any technique which reduces the probability of error will improve the security. In general, these proposals can be divided into two categories: 1- Short term solutions on the existing DRAM modules, 2- technique that require modifications either in the manufacturing process or memory subsystems.

Since rowHammer only takes effects in refresh-intervals, perhaps the most direct solution would be to increase the DRAM refresh rate. Although very effective, this solution has high performance and energy overhead [5]. Another recently proposed solution is ANVIL which is a software aided technique that exploits hardware performance counters to detect rowHammer attacks and selectively refresh the victim rows [1].

In the long term, the best solution for rowHammer problem is to make sure that future DRAM cells are more reliable and are less vulnerable to rowHammer. Considering the ever shrinking transistor size, and high density demand, this

will be a challenging task if not impossible. Another proposed solution is to locate vulnerable cells through exhaustive testing and remap error prone cells. However, such a testing mechanism also comes at a high cost and increases time-to-market. Vulnerable cells can also be retired at the system-level which incurs high performance overhead of detection and area cost of bookkeeping potential victim and target rows. Another promising solution is to use error correcting codes (ECCs) to correct any bit-flip caused by a rowHammer attack. Unfortunately, ECC imposes high storage capacity and requires modification on both DRAM modules[5] and memory controller.

As explained above, most of the solutions to the rowHammer problem impose high overheads on the systems. Kim et al. proposed a new technique, *i.e.* PARA, which is somehow similar to ANVIL in the way it selectively increases refresh rate of some rows [1]. But PARA is stateless, which means does not require row accesse counters. In this technique, when a row is accessed, with a low probability[6] its neighbors are refreshed. Hence, if a row is accessed multiples times, *i.e.* is being hammered, with a high probability its neighboring rows will be refreshed before disturbance causes a bit-flip [5].

References

1. Aweke, Z.B., Yitbarek, S.F., Qiao, R., Das, R., Hicks, M., Oren, Y., Austin, T.: Anvil: Software-based protection against next-generation rowhammer attacks. ACM SIGPLAN Notices 51(4), 743–755 (2016)
2. Gruss, D., Maurice, C., Mangard, S.: Rowhammer. js: A remote software-induced fault attack in javascript. In: Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 300–321. Springer (2016)
3. Jacob, B., Ng, S., Wang, D.: Memory systems: cache, DRAM, disk. Morgan Kaufmann (2010)
4. Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J.H., Lee, D., Wilkerson, C., Lai, K., Mutlu, O.: Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In: ACM SIGARCH Computer Architecture News. vol. 42, pp. 361–372. IEEE Press (2014)
5. Mutlu, O.: The rowhammer problem and other issues we may face as memory becomes denser. arXiv preprint arXiv:1703.00626 (2017)
6. Pawlowski, J.T.: Hybrid memory cube (hmc). In: Hot Chips 23 Symposium (HCS), 2011 IEEE. pp. 1–24. IEEE (2011)
7. Razavi, K., Gras, B., Bosman, E., Preneel, B., Giuffrida, C., Bos, H.: Flip feng shui: Hammering a needle in the software stack. In: Proceedings of the 25th USENIX Security Symposium (2016)
8. Seaborn, M., Dullien, T.: Exploiting the dram rowhammer bug to gain kernel privileges. Black Hat (2015)
9. van der Veen, V., Fratantonio, Y., Lindorfer, M., Gruss, D., Maurice, C., Vigna, G., Bos, H., Razavi, K., Giuffrida, C.: Drammer: Deterministic rowhammer attacks on mobile platforms. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1675–1689. ACM (2016)

[5] Require additional chip and extended channel width.

[6] statically adjustable.