June 2020

Preventing Row Hammer Attacks by Dynamic Indirection of Row Addresses

N/A

Preventing Row Hammer Attacks by Dynamic Indirection of Row Addresses

ABSTRACT

Row hammer in dynamic random access memories (DRAM) is an effect by which repeatedly activating a row of the DRAM causes bits in nearby rows to flip. Because OS and program variables can be stored adjacent to each other in the DRAM, a malicious program can repeatedly activate DRAM rows to flip nearby bits that store important OS states (e.g., program privileges). In this manner, an attacker can gain unauthorized, privileged access to a computer. This disclosure describes techniques that use a combination of indirection and randomization to make it difficult for an attacker to hammer DRAM rows. Per the techniques, the relationship between memory addresses and physical rows is made random and dynamic, such that the physical relationship between the rows is difficult, if not impossible, to discover and exploit.

KEYWORDS

- Dynamic random access memory (DRAM)
- Row hammer attack
- Logical-to-physical table
- Row buffer

BACKGROUND

A dynamic random access memory (DRAM) includes banks of memory cells organized into rows and columns. Before a row can be read or modified, it must be activated, which is when its contents are copied into the row buffer. Only one row of a bank can be activated at a given point in time. To activate another row, the bank is pre-charged, readying it to activate a different row.

Activating a row repeatedly can cause bits in nearby rows to flip. This is known as the row hammer effect. It is not uncommon for program-related instructions or variables to be stored adjacent to operating-system instructions or variables. A malicious program can repeatedly activate DRAM rows that store its own variables, with the effect that nearby rows that store important OS states (e.g., user or program privileges) flip. An attacker can thus gain unauthorized, privileged access to a computer. As DRAMs become increasingly dense, physics dictates that the row-hammer effect will intensify, making computers more vulnerable.

Rows repeatedly accessed (hammered) by an attacker are known as aggressor rows. Rows that experience bit flips are known as victim rows. Some current techniques of hardening against row hammer attacks refresh victim rows; however, these do not defend against bit flips that occur at row distances greater than unity. Other techniques that attempt to harden against row hammer attacks entail a static DRAM remapping scheme that distributes row hammer errors over multiple words. An error pattern thus distributed over a large group of words can be recovered by an error-correcting code. The focus of such techniques, however, is data integrity rather than a security. To the extent that such techniques at all achieve secure computing, they amount to security-by-obscurity.

Operating system based techniques to harden against row hammer attacks include the use of hardware performance counters to detect row hammer events. Once detected, such attacks are foiled by copying data and moving page table entries to move the page that is being hammered. Alternatively, once a row hammer is detected, the rows can simply be refreshed.

Some mitigation techniques count, and bound, row activations without tracking per-row bits. Row hammer attacks are mitigated by the bound on the number of row activations within one refresh window. The enforcement mechanism is the refreshing of the adjacent (victim) rows.

These techniques are typically implemented outside the DRAM, e.g., in the register clock driver, which can be expensive in terms of space.

A probabilistic approach to mitigating row hammer is to refresh neighboring rows with some probability upon each activation. The expectation is that any row hammering attempt is likely to result in refreshes to neighboring (potential victim) rows, due to the large number of activations carried out during a row hammer attack. Again, the mechanism of enforcement is the refreshing of victim row(s). This approach becomes more complex as the row hammer distances increase. For example, on highly dense DRAM, it is possible to cause bit flips two rows away from an aggressor row. In such a scenario the number of refreshes can potentially grow unsustainably. Being typically implemented in the memory controller, probabilistic approaches are limited by the double data rate (DDR) protocol, which bounds the number of refreshes that can actually happen. Additionally, the probabilistic nature means that strong guarantees about preventing bit flips cannot be provided. Probabilistic approaches can work if the memory controller has complete knowledge of row-adjacency inside the DRAM chip and the probability value is set close enough to unity. However, the transmission of row-adjacency information from DRAM to memory controller leads to power and performance overheads that may be unsustainable. Setting the probability value too close to unity can trigger a lot of false positives, whereby false victim rows are frequently and unnecessarily refreshed, leading to unsustainable energy consumption.

Thus, existing techniques utilized by DRAM and memory controller vendors to prevent row hammer attacks are limited in the ability to mitigate such attacks.

DESCRIPTION

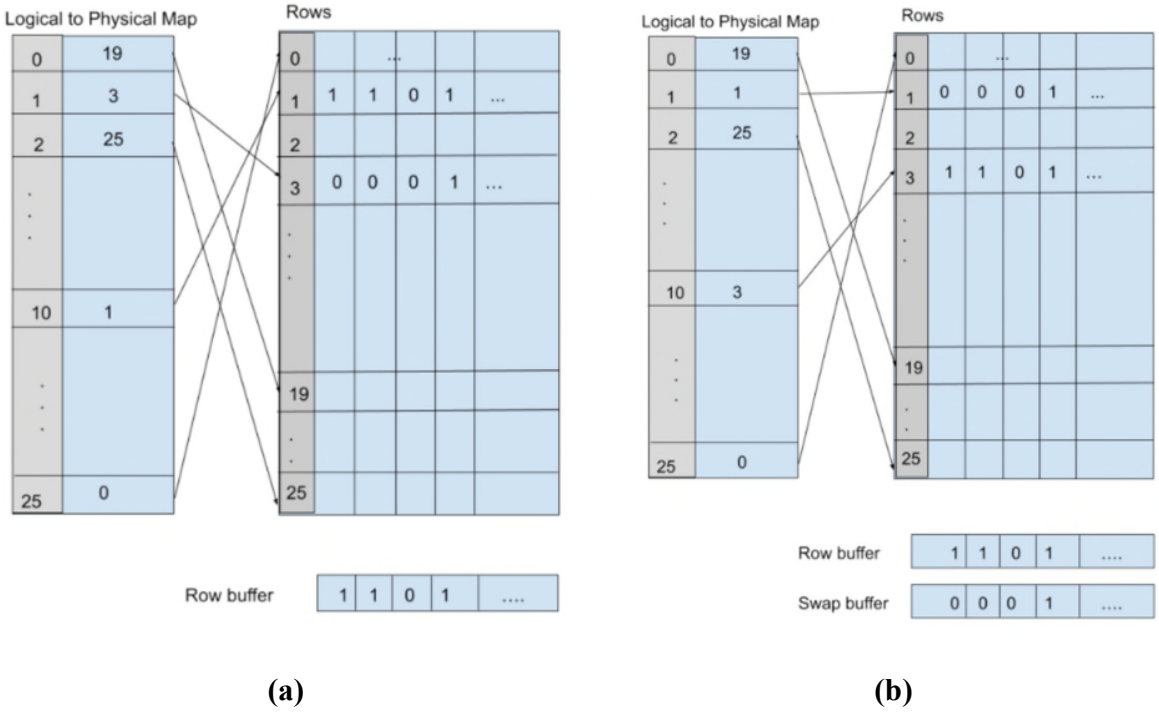
To successfully execute a row hammer attack, an attacker needs to deduce information about neighboring DRAM rows. This information is usually static, and once figured out, enables the attacker to repeatedly access certain rows to cause bit-flips in a nearby row. This disclosure describes techniques that use a combination of indirection and randomization to make it difficult for an attacker to hammer rows. Per the techniques, the relationship between memory addresses and physical rows is made random and dynamic, such that the physical relationship between the rows is difficult, if not impossible, to discover and exploit.

In addition to the row buffer, the techniques augment a memory bank with another buffer known as the swap buffer, which has the same size as the row buffer. Loading the swap buffer has a cost similar to that of row activation. An indirection table, known as the logical-to-physical (L2P) table, maps an incoming logical row to a physical row in the bank. For example, if a bank has 64K rows, then the L2P table can comprise 64K two-byte entries. Entries of the L2P table can be initialized as a permutation of the physical rows. The `k` th row of the L2P table is referred to as `L2P[k]`. The L2P table can be stored in a static RAM (SRAM) block coupled to the DRAM module. Further, the mappings specified by the L2P table are confined to the DRAM module, with no method provided to surface the mappings.

When a row is activated in the idle state, rather than using the raw offset to read a row from the bank, the row number is looked up in the L2P table and the corresponding physical row is activated. In particular, when a logical row `src` is pre-charged, the following take place:

- The logical row number `src` is looked up in the L2P table to arrive at a physical row number `src'`.
- A logical row index, `dest`, is randomly generated.

- The logical row number `dest` is looked up in the L2P table to arrive at physical row number `dest'`.
- The contents of the row `dest'` are loaded into the swap buffer.
- The current contents of the row buffer are moved into the row `dest'`.
- The contents of the swap buffer are moved into the row `src'`.
- `L2P[src]` and `L2P[dest]` are swapped.



(a) **(b)**

Fig. 1: The L2P map and the physical layout of bits in the DRAM: (a) Before relocation of logical row 10; (b) After the relocation of logical row 10.

Effectively, physical rows are swapped for two logical rows. This involves swapping the content between the physical rows and updating the L2P to reflect this change, as illustrated in Fig. 1. Fig. 1(a) illustrates the L2P map and the physical layout of bits in the DRAM prior to relocation of logical row 10, which is presently mapped to physical row 1. Fig. 1(b) illustrates the L2P map and the physical layout of bits in the DRAM after the swapping of logical row 10

with a randomly selected row, such that logical row 10 now maps to physical row 3. Thus, the contents of physical rows number 1 and 3 have been interchanged.

A row hammer attack entails repeated activation of one or more aggressor rows. Even if the attacker were to find the original mapping of the rows necessary to mount a row hammer attack, with the implementation of row swapping described herein, the mapping changes as soon as the rows are pre-charged (or after a certain number of activation/precharge cycles have elapsed, as further described below). Further, as explained before, the mappings of the L2P are confined to the DRAM module, with no method to extract them. The shifting, randomized mapping of the L2P table and its confinement to the DRAM module render repeated hammering of the same set of rows nearly impossible.

Copying of data within a DRAM can be done efficiently using primitives, e.g., as described in [1]. The use of such primitives for copying is efficient because the data stays within the chip or device, and simply passes through internal buffers on its way to another location in the same bank.

If the L2P table is very large, rather than storing it in a coupled SRAM, it can be stored within the DRAM module that it indexes. In such a case, to protect the L2P table from row hammer attacks, a randomized index of metadata rows can be maintained in a first-level L2P table in SRAM. The first-level L2P table in SRAM is a directory (base+offset physical address to physical row in DRAM) to a second-level L2P table maintained in DRAM. The second-level L2P table contains the actual mapping, e.g., DRAM logic row to physical row mapping. In this manner, implementations are provided that straddle the DRAM module and memory controller boundary, and which hierarchically address rows in the DRAM through randomized mapping.

Since true random number generators are slow when used directly, a true random number generator can be used to seed a pseudorandom number generator at some reasonable granularity. To reduce the energy and time needed to relocate rows, a counter can count the number of activations since the last move. The counter can comprise relatively few bits and the logical row can be moved when the counter overflows. The counter can be designed to overflow at a point well before the number of row hammers necessary to flip neighboring bits. This number is dictated by the physics of the DRAM technology used.

Some systems-on-chips use more than one row address bits in the hash functions for DRAM bank addresses. In such DRAMs, swapping row addresses may result in the moving of data between two banks, which can be beneficial from a row hammer mitigation standpoint, as the aggressor row is now placed in any of several, e.g., eight, banks. In such a case the memory controller is configured to keep track of the rows that crossed bank boundaries and accordingly, issue commands to the correct bank.

The row swap can be limited within the same bank for simplicity, and for energy and time conservation purposes. Limiting the swapping of rows to within a single bank abstracts away implementation details from the memory controller, which can send the same addresses and let indirection be performed inside the DRAM. As mentioned before, extending this mechanism to enable swapping across banks, e.g., using copying primitives as described in [1], requires the active participation of the memory controller. To access a particular row, the memory controller queries the L2P to identify the correct bank at which the particular row currently resides.

To reduce memory footprint, rather than having bits built into the L2P table, a separate table of smaller size can be maintained as a LRU cache. Entries of this table are used to choose

the rows to randomize (rather than the rows to refresh). The LRU cache table includes fields such as the valid bit, the row address, the activation count, a notion of a timestamp (or age), etc. If the LRU cache is implemented as part of the bank logic, the row number would be smaller, making the representation more compact. Alternatively, the LRU cache can be implemented in the memory controller, with a somewhat larger footprint, but with the ability to share space between multiple chips and banks. In this case, the protocol between the memory controller and the DRAM is modified to include a command to request row randomization.

As opposed to techniques that harden against row hammer attacks by refreshing victim rows, the techniques described herein use indirection and dynamic randomization of row mappings to make it harder to consistently hammer aggressor rows. Risk due to bit flips that happen at row-distances greater than unity is thereby well mitigated. Per the techniques, an attacker cannot choose or predict the physical row of a DRAM that is to be activated.

CONCLUSION

This disclosure describes techniques that use a combination of indirection and randomization to make it difficult for an attacker to hammer rows of a dynamic random access memory. Per the techniques, the relationship between memory addresses and physical rows is made random and dynamic, such that the physical relationship between the rows is difficult, if not impossible, to discover and exploit.

REFERENCES

[1] Seshadri, Vivek, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo et al. “RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization.” In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 185-197. 2013.

[2] Kim, Moonsoo, Jungwoo Choi, Hyun Kim, and Hyuk-Jae Lee. “An Effective DRAM Address Remapping for Mitigating Rowhammer Errors.” *IEEE Transactions on Computers* 68, no. 10 (2019): 1428-1441.

[3] Lee, Eojin, Ingab Kang, Sukhan Lee, G. Edward Suh, and Jung Ho Ahn. “TWiCe: preventing row-hammering by exploiting time window counters.” In *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 385-396. 2019.

[4] Kim, Yoongu, 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 *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 361-372. IEEE, 2014.

[5] Frigo, Pietro, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. “TRRespass: Exploiting the Many Sides of Target Row Refresh.” presented at the Forty-First IEEE Symposium on Security and Privacy, May 15-18, 2020, preprint available online at https://arxiv.org/abs/2004.01807.

[6] Aweke, Zelalem Birhanu, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. "ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks." In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 743-755. 2016.