November 2020

Preventing Memory Exploits Using Programmable Row Swizzling

N/A

Preventing Memory Exploits Using Programmable Row Swizzling

ABSTRACT

Row hammering is a type of memory exploit where a malicious application repeatedly causes transistors in a row of cells within a memory to rapidly turn on and off in particular patterns. If a row is hammered enough times, it is possible to change data stored in the adjacent rows, leading to malfunctions, shutdowns, or security breaches. This disclosure describes techniques that make it much harder for an attacker to find row-hammer patterns that can compromise a DRAM. Per the techniques, each DRAM has dedicated, programmable row-address swizzling (logical-to-physical address mapping) which changes when the machine reboots. The per-DRAM swizzling makes row-hammer patterns harder to find, and if found, inapplicable when the machine reboots. The per-DRAM swizzle also ensures that a row-hammer pattern found on one DRAM on a DIMM fails on the other DRAMs in the DIMM.

KEYWORDS

- Dynamic random access memory (DRAM)
- Row hammer
- Row swizzling
- Logical-to-physical address map
- Memory attack
- Memory exploit
- Memory controller
- Memory refresh

BACKGROUND

Dynamic random access memory (DRAM) is organized into rows of capacitors that store data and transistors that control access to the data in a row. Row hammering is a type of exploit where a malicious application repeatedly causes the transistors in a row to turn on and off rapidly by accessing data in particular patterns. Every time a row is accessed, a small amount of charge leaks to adjacent rows. If a row is hammered enough times before the adjacent rows are refreshed, it is possible to change the charge enough such that the data stored in the adjacent rows is corrupted. Such adjacent rows that suffer data corruption are called victim rows.

By causing memory errors, row-hammer attacks can at least cause program or computer malfunctions or shutdowns, leading to a denial of service. Row hammers have also been used to escalate the privileges of malicious applications, leading to serious security breaches.

Row hammering is a well-known issue in DRAM. Row hammering is somewhat mitigated by increasing the DRAM refresh rate that restores the charge in DRAM capacitors. By doing so, the affected rows restore their original values before the row hammer can repeat enough times to corrupt data. Another defense against row-hammer attacks is to swizzle (scramble) row addresses such that sequentially numbered logical rows are not physically adjacent to each other. Some memory units are also equipped with error-correcting (or detecting) codes (ECC) that detect or correct the corruption of small numbers of bits.

However, in current DRAMs, which use a fixed swizzling scheme (logical-to-physical address map), the discovery of row-adjacency by an attacker on a memory part on a single machine makes vulnerable all machines that use the same part.

Furthermore, a certain row-hammer attack, known as TRRespass [1], has been proven to bypass the above defenses. The TRRespass attack

- hammers enough rows to overwhelm row-hammer tracking mechanisms in the memory controller;
- picks random row addresses to discover which row addresses are physically adjacent, overcoming row address swizzling in the DRAM;
- times memory accesses, e.g., uses time as a sideband memory error detector, allowing user-mode code to determine when error-correcting codes in the memory controller corrects a memory error (indicating a successful row hammering); and
- once row-hammering patterns are found, uses these patterns to attack machines with similar types of memory controllers and DRAMs.

On client machines, e.g., smartphones, laptops, etc., whose memories typically do not use ECC, the above attack can cause crashes or serious security breaches. On servers, whose memories typically have ECC, the above attack can cause a denial of service attack launched from user space.

DESCRIPTION

This disclosure describes techniques that make it much harder for an attacker to find row-hammering patterns. Per the techniques, each DRAM has dedicated, programmable row-address swizzling, e.g., logical-to-physical address mapping, which changes when the machine reboots. The per-DRAM swizzling makes row-hammer patterns harder to find, and if found, inapplicable when the machine reboots. An attacker is forced to rediscover the row-hammer pattern on each machine on each boot. The per-DRAM swizzle also ensures that a row-hammer pattern found on one DRAM on a dual in-line memory module (DIMM) fails on the other DRAMs in the DIMM, ensuring, for example, that DDR4x4 parts can correct row-hammer errors using standard ECC.

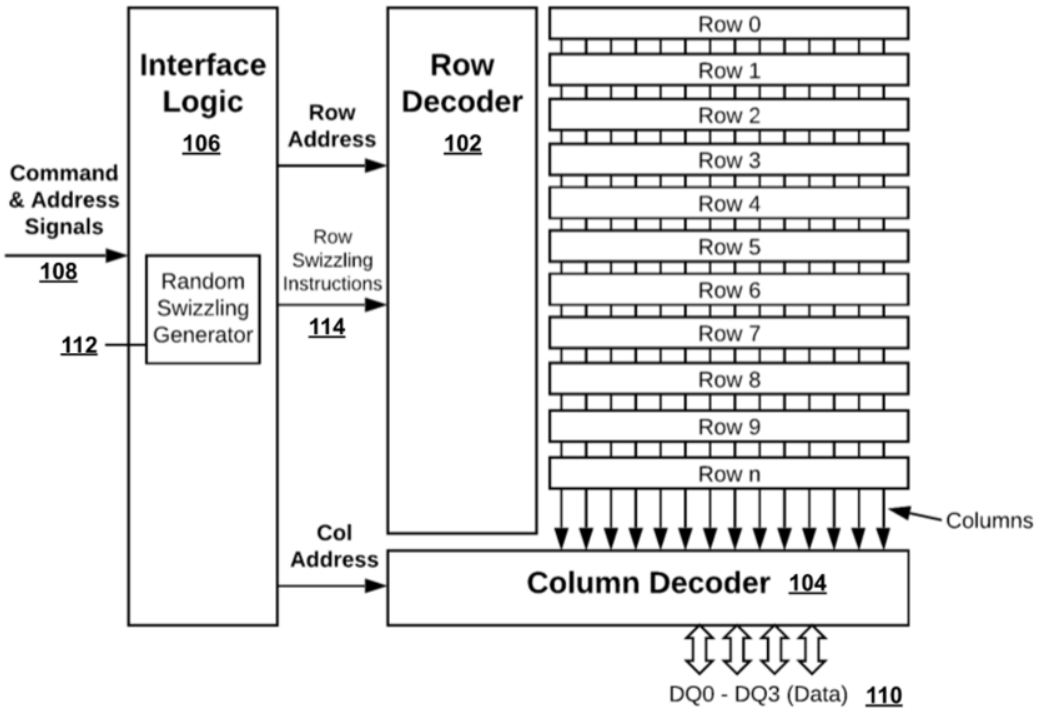

Fig. 1: Preventing row hammer exploits using programmable row swizzling

Fig. 1 illustrates preventing row-hammer exploits using programmable row swizzling, per the techniques of this disclosure. Memory is organized into rows of capacitors (rows 0 through n) that store data. A command and its address signal (108) arriving from the CPU is decoded by an interface logic (106) to generate a row address and a column address. The contents of the requested memory location are accessed by activating the row (using row decoder 102) and the column (using column decoder 104) of the memory location. The contents of the requested memory location are sent on the data bus (110).

A random swizzling generator (112) issues row-swizzling instructions (114) that ensure that sequential logical addresses map to random physical rows; in particular, sequential logical addresses do not map to adjacent physical rows. As mentioned earlier, the row-swizzler is programmable and distinct for each DRAM. For example, physical row 2 and its adjacent physical rows 1 and 3 have different logical row addresses on every DRAM in the DIMM.

The row swizzling can be changed, or reprogrammed, in various ways. For example, a large number of possible row-swizzles, e.g., logical-to-physical address maps, can be pre-stored, and a command to select (change) the logical-to-physical address map be issued when the memory trains. Alternatively, a built-in random number generator can generate a new row swizzling configuration when the DIMM is reset or powered-up.

Row address swizzling can be implemented in a variety of locations, e.g., the row address decoder, the interface logic, or other parts of the DRAM. The row address swizzling can also be implemented outside the DRAM, in the memory controller.

Swizzling command

When the memory powers upon booting, the memory controller can send a command to the DRAM to instantiate a new row-address swizzling configuration for the duration of the boot. This enables the BIOS or unified extensible firmware interface (UEFI) to ensure that the row addresses are swizzled differently every boot while protecting the swizzling information from a potential attacker. The swizzling command herein described can be incorporated into DRAM standards such as JEDEC. Per DRAM addressing (PDA) can be used to configure each DRAM on the DIMM with a distinct row-swizzling configuration.

Random number generator in the DRAM

As mentioned earlier, a random number generator in the DRAM can be used to configure row swizzling when the DRAM is reset, powered up, or issued a specific command. In contrast to the previously described command, which only instructs the DRAM to change its logical-to-physical address map, the random number based command establishes the logical-to-physical address map.

Partial row swizzling

In situations where it is impractical to swizzle all the row addresses, the low order bits of the row number can be swizzled such that sequentially numbered logical rows are not physically adjacent. Alternatively, the DRAM can skip a row order bit for swizzling using any combination of low and high order bits. The effectiveness of row swizzling, e.g., the robustness to row-hammer attacks, increases with the number of rows that are swizzled, or when many low order bits of the row numbers are swizzled.

Debuggability

Memory parts that fail are often returned to the vendor to undergo failure analysis. An understanding of how row addresses are swizzled can be used to improve the quality of memory. The BIOS/UEFI can send the DRAM a command to enable logging of the swizzling configuration such that the log can later be used for failure analysis. Knowing how the row address command is used to swizzle row addresses, a failure analyst can translate from the row address of the memory controller to the physical row address inside the DRAM.

Runtime re-swizzling

It is possible to re-swizzle row addresses at runtime, rather than boot time, as follows. The CPU copies the contents of the DRAM to scratchpad memory, issues the row-swizzling command, and copies the contents back from scratchpad to DRAM. If the DRAM has spare rows, the CPU can quiesce the operating system to pause memory accesses, send a command to re-swizzle the row addresses, and trigger a state machine inside the DRAM to go through each row, copying data from the old row address to the new row address. The memory can be re-swizzled a bank-at-a-time and in such a case, the scratchpad would need to be at least the size of the bank.

Low-memory row-swizzling command at runtime

It is also possible to re-swizzle memory using less scratchpad memory by re-swizzling memory a physical or logical row-at-a-time, in swizzle order. When the swizzling logic receives a re-swizzle command, the re-swizzling logic walks through the logical or physical rows, swapping the data from the old location to the next location. In this scheme, the swizzling logic copies the next row in the order to a scratchpad, moves the previous row to the next row's location, continuing this until all of the rows have been copied to their new locations. This only requires a scratchpad that has space for two rows and requires the swizzling pattern that the re-swizzling logic could walk, such as a pseudo-random pattern from something like a linear feedback shift register. The memory can be quiesced while the rows are moved to their new location or the re-swizzling logic can allow memory accesses during the re-swizzling by keeping track of where it was in the copying sequence and making sure accesses to memory happen in the old row location for rows that had not been moved and in the new location for moved rows.

Scope of row swizzling

To forestall increases in die space or cost caused by the swizzling logic, the DRAM can have row address swizzling logic that is shared by all the banks in the DRAM. Alternatively, if die space is not an issue, the row address swizzling logic can be implemented per bank-group or per bank.

CONCLUSION

This disclosure describes techniques that make it much harder for an attacker to find row-hammer patterns that can compromise a DRAM. Per the techniques, each DRAM has dedicated, programmable row-address swizzling, e.g., logical-to-physical address mapping, which changes when the machine reboots. The per-DRAM swizzling makes row-hammer patterns harder to

find, and if found, inapplicable when the machine reboots. The per-DRAM swizzle also ensures that a row-hammer pattern found on one DRAM on a DIMM fails on the other DRAMs in the DIMM.

REFERENCES

[1] 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." *arXiv preprint arXiv:2004.01807* (2020).

[2] CVE-2020-10255 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-10255