Fall 2018

Row hammer exploit in cloud environment

Adithya Venkataraman
Iowa State University

Row hammer exploit in cloud environment

by

Adithya Venkataraman

A report submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Electrical and Computer Engineering

Program of Study Committee:
Akhilesh Tyagi, Major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2018

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

ABSTRACT

The rapid increase in the adoption rate of cloud computing, across numerous businesses, has resulted in extensive use of virtualization tools. Virtualization technology utilizes a software layer (hypervisor) to enable sharing of hardware between multiple tenants that are co-located on the same multi-processor system. This enables the consolidation of servers and user machines into a very small set of physical systems. Physical machines are replaced with virtual machines (VM), running on the same physical system, to achieve better utilization of the hardware. Consequently, cloud users work on and store their data in the same physical machine.

A crucial part of a cloud setup is preventing information leakage between tenants. While the hypervisor enforces software isolation, shared CPU, cache or memory, has the potential to leak sensitive information [4]. This article aims to provide an overview of the security concerns in virtualization technology, particularly in relation to row hammer bug that affects the DRAM chips.

As DRAM process technology scales down in dimension, it becomes increasingly difficult to prevent sub-micron electrical interaction between DRAM cells. This leads to unintentional effects where, activating the same row or same set of rows in DRAM (row hammer) corrupts data in nearby rows. If row hammer is coupled with some resource sharing features that can be enabled in hypervisors, then there is a high likelihood of corrupting a piece of data that belongs to another VM. This article is a survey of some prior works in cloud-based row hammer attacks [6][7] and sheds some light on the exploit mechanics.

CHAPTER 1: INTRODUCTION

The push towards low-cost, high-performance computing results in an increase in hardware and software complexity. The semiconductor industry packs more and more transistors into chips which makes them denser and the components on the chip are getting closer to each other with time. Modern operating systems are loaded with features to support efficient resource management in performance sensitive settings such as the cloud.

The outcomes of the above-mentioned trends include reduced reliability and security. Hardware components are increasingly prone to failures. Smaller cells store smaller charge, resulting in smaller noise margin. As a direct result of this, a significant fraction of the DRAM chips produced in recent years (since 2012) are prone to unintended bit flips. Software features such as memory deduplication serve as side channels for attackers.

This paper explores attack methodologies to breach the software isolation (enforced by the hypervisor) between tenants on a multi-tenant setup. The rest of this report is organized as follows. Chapter 2 talks about some background information about memory hierarchy and DRAM organization, the understanding of which plays a critical role in understanding row hammer bug. Chapter 3 goes on to summarize the row hammer attack, related work and attack methodologies that could result in an increased probability for bit flip observation in a physical DRAM, using DRAM mapping functions. Chapter 4 gets into the details of cloud computing and data deduplication feature present in most popular cloud vendors including Xen, KVM, VMware ESXi. Finally, chapter 5 talks about how this sharing feature could be leveraged to corrupt the data that belongs to another VM (isolation breach).

CHAPTER 2: MEMORY ORGANIZATION

2.1 Memory lookup

Modern operating systems allow programs to work with seemingly contiguous memory space known as the virtual memory. When a program needs access to some data, the CPU requests for that particular virtual address where the data is assumed to be stored by the OS. The virtual address is then translated to a physical address, which is an actual location on the memory.

A data structure, known as page table, holds the translation between virtual page address and physical frame address. These mappings are stored as entries in the page table for each program. This means that if a program wants to access some virtual address then the page table for that program is searched to retrieve the mapping.

Frequently accessed addresses are cached in translation lookaside buffer (TLB), which acts as a cache for page table entries (PTE). If the mapping is not present in TLB then the page table is searched and if its absent in the page table then it is loaded from the disk, following which the PTE and TLB are updated. *Figure 2.1* describes this process.

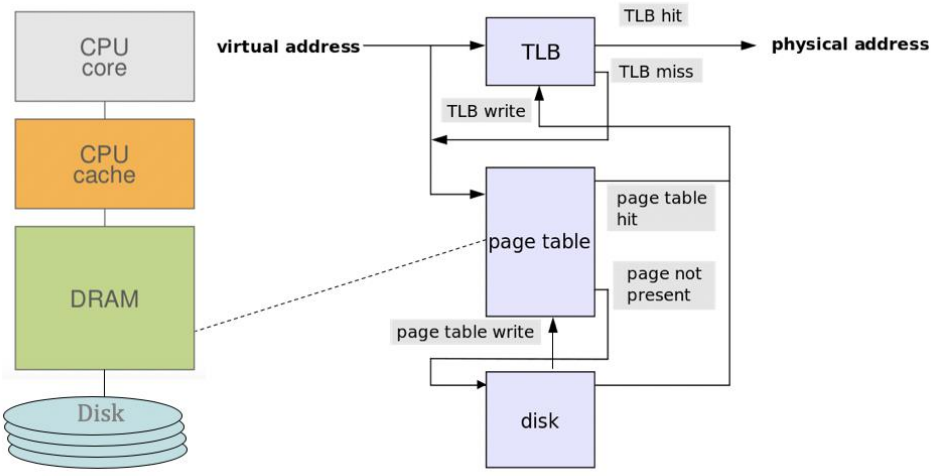
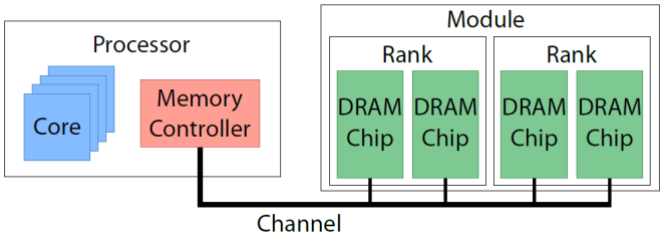


Figure 2.1 *Memory access hierarchy*

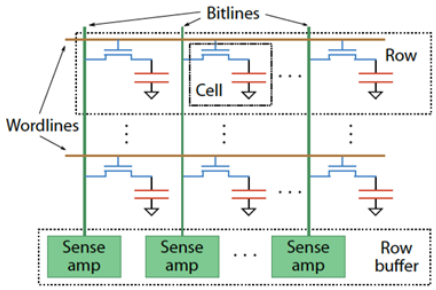
2.2 DRAM organization

DRAM chips are large arrays of memory cells with additional logic for data access (read/write) and refresh circuitry for maintaining data integrity. Memory systems are organized into multiple memory channels, each operated by its own dedicated memory controller. Each channel is divided into ranks (sides of the ram stick), which consist of several DRAM chips that work together to handle misses or refill requests from the processor's last-level cache. Each rank is partitioned into multiple banks. Every bank has a dedicated cache (row buffer) to store the last accessed row in that bank. Ranks and banks support independent transactions which allows parallel accesses to the DRAM chips. *Figure 2.2 (a)* captures the memory hierarchy described in the paragraph.

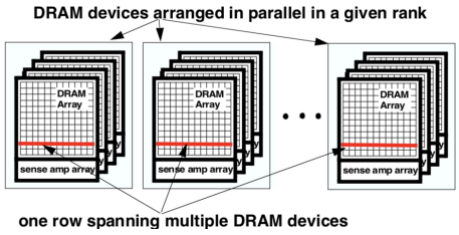
Memory arrays are organized in rows (word-lines) and columns (bit-lines) of memory cells. Memory cells consist of a capacitors that can be charged and discharged to store a 0 or a 1. An access transistor in each cell allows reads and writes to its content. The transistor is controlled through the word-line. When the word-line is activated, the content of all the capacitors on that row are discharged to the bit-lines. Sense amplifier circuitry on each bit-line amplifies the signal and stores the result in the row buffer. Additional circuitry in the memory arrays includes address decoding logic to select rows and columns and internal counters to keep track of refresh cycles. *Figure 2.2 (b)* shows the word-lines, the bit-lines and the sense amplifiers described in the paragraph.



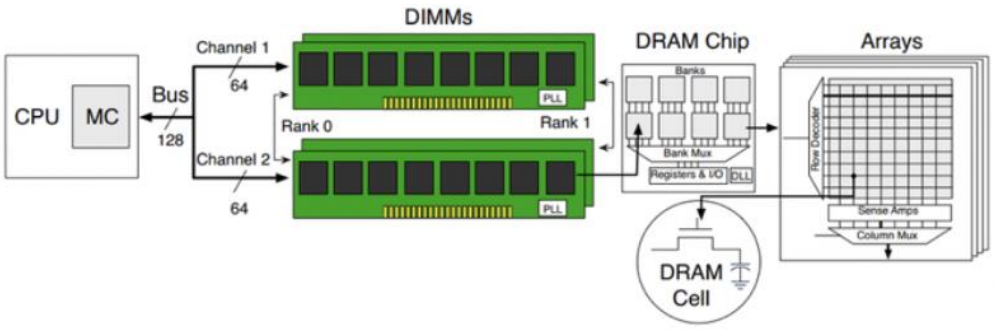
(a) Memory System Hierarchy



(b) DRAM bank structure



(c) DRAM row structure



(d) Memory hierarchy with multiple channels and DIMMs (Module)

Figure 2.2 *DRAM system organization*

CHAPTER 3: ROW HAMMER

3.1 Row hammer Mechanics

Row hammer, as an attack vector, targets the design vulnerabilities present in modern dynamic random-access memory (DRAM) chips. As DRAM memory cell density increases, electrical interaction between neighboring cells also increases, causing leakage of capacitor charges at an accelerated pace and, potentially, data loss.

As discussed in the previous chapter, DRAM chips consist of rows of cells. These cells are periodically refreshed to maintain the charge stored in each cell. When the CPU requests a read/write operation on a particular memory location, memory controller transfers the data to the row-buffer (discharge). After performing the requested operation, memory controller copies the contents of the row-buffer back to the row (recharge). The normal read cycle has been depicted in *Figure 3.1*. Under normal circumstances, such read and write operations do affect the charge stored in the adjacent rows. This can be ensured by making use of clever schemes for virtual to physical memory mapping, to prevent successive accesses to the rows in the same bank (bank thrashing).

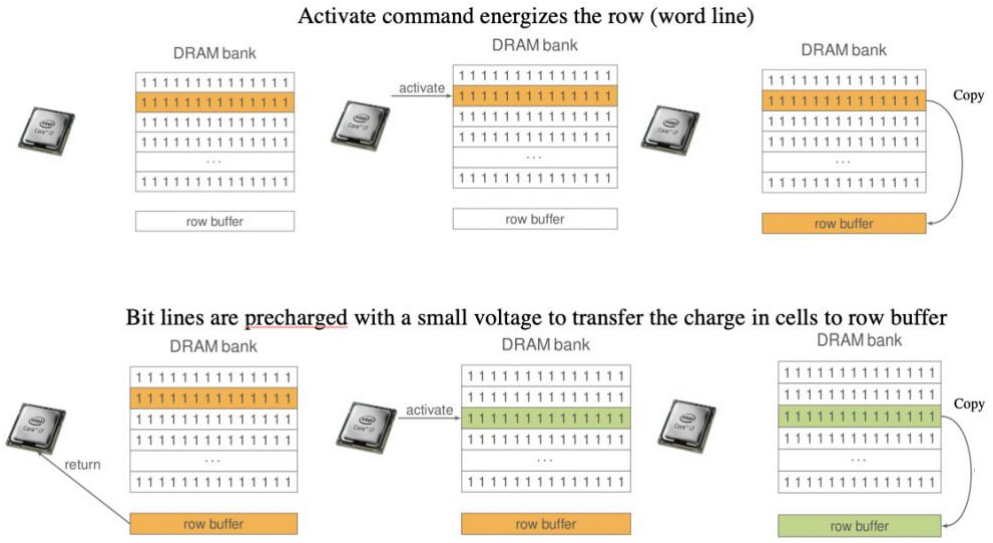


Figure 3.1 *Normal read operation*

Frequent discharge-recharge cycle, known as row hammer, can cause errors that are reflected by an accelerated discharge rate on adjacent row's cells. If the adjacent rows (victim rows) are not refreshed before losing their charge, the acceleration in discharge rate can introduce unintentional bit-flips. *Figure 3.2* explains the exploit visually.

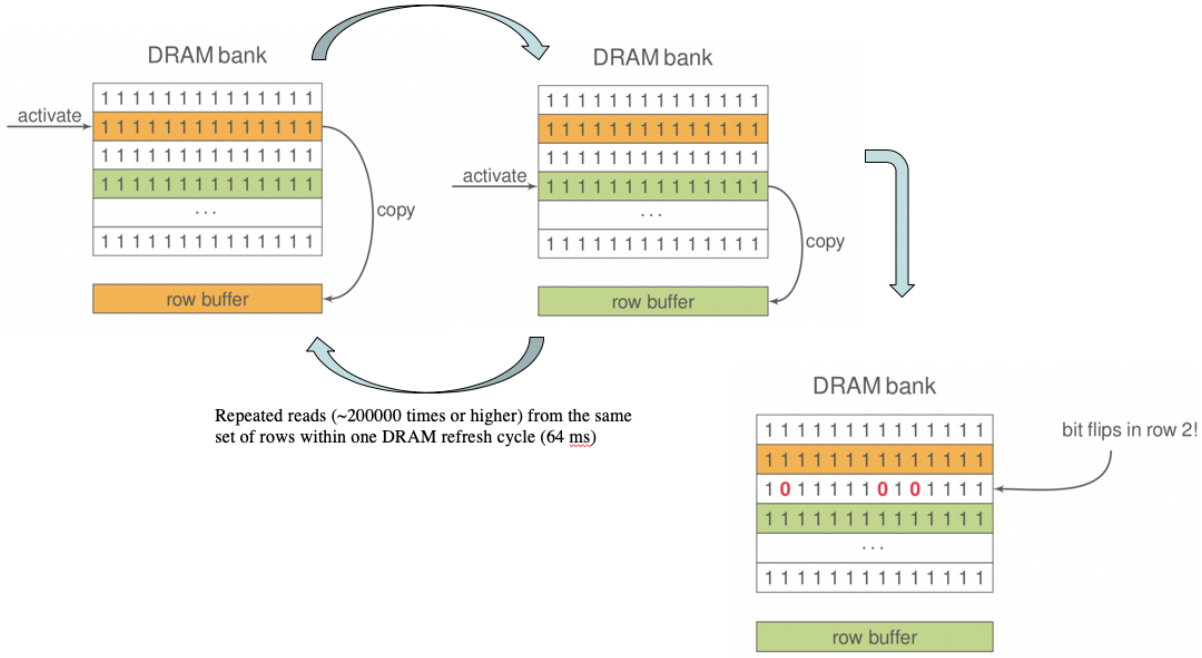


Figure 3.2 *Row hammer Mechanics*

3.2 Cache eviction

One of the fundamental requirements of row hammer is that the repeated reads should happen from the DRAM. In a modern architecture, recently used data is cached. It means that for subsequent operations, the data is fetched from cache rather than the DRAM. Prior work [1][4] make use of the following cache eviction strategies:

- *clflush*
- cache eviction by cache set filling

clflush instruction is provided by intel for cache maintenance and can be used to remove data from cache. clflush instructions, inserted after every read can ensure that the subsequent data accesses always go to DRAM. But clflush is as much a cache attack as a DRAM attack [4]. Also, clflush is language dependent. For example, it is not available in JavaScript, for web-based attacks.

Another technique of removing data from cache is to fill up the cache set with random data until the target data has been evicted [4]. In any memory architecture, various DRAM addresses map to the same cache set. Each cache set has ways, typically 8. Once the data belonging to a particular memory address has been placed in one of the ways, the other remaining ways are used for future accesses involving memory addresses that map to the same cache set, before evicting the data from the previous memory access. The order of eviction depends on the replacement policy used by the CPU.

3.2 Physical memory mapping

Another condition that will significantly boost the chances of seeing a bit-flip is if the memory being accessed belong to different rows but same bank. As discussed earlier, every bank has a row buffer. If row was accessed before, the next access to that row would be happen from the row buffer rather than the row itself. To identify address pairs that follow this condition, clever side channel methods can be used. Figure 3.3 shows the timing observed for various pairs. The most favorable address pair condition has been highlighted. If the address pairs belong to different banks, the next access latency is low because they are cached (row hit). But if the address pairs belong to the same bank (different row), the subsequent access latency is going to be high because the address pairs will keep evicting each other from the row buffer (row conflict). Using this timing difference along with a brute

force search can reveal the physical address to physical location mapping (i.e. the exact channel, rank, bank details), as shown by gruss et al. [4]. By making use of the mapping information, row hammer attacks can be made a more effective and efficient.

Bank	Row	Latency
Same	Different	High
Same	Same	Low
Different	Different	Low

Figure 3.3 Timing information for address mapping

Typical address map in consumer grade x86 platform

If a system has 4 gigabytes of memory, it can be indexed completely using 32 addressing bits. Which means that the physical address has to be 32 bits long. The physical address bits are further broken down for identifying the exact physical location, as given below:

column index lower bits (0-5)

channel number bit (6)

column index upper bits (7-13)

bank address bits (bits 14,15,16 XOR'd with lower 3 bits of row index bits)

rank number (17)

row index bits (18-32).

This is for a dual channel, dual rank, 8 bank memory structure.

CHAPTER 4: CLOUD COMPUTING

4.1 Virtualization

The concept of virtualization is the foundation upon which modern cloud infrastructures thrive. Virtualization is the creation of virtual versions of some real objects such as hardware and software. It enables creation of a layer of abstraction to mask the complexity of the underlying software or hardware layer. Logical partitions of real objects are made, to create instances of virtual objects. An example of this would be the hard disk on our computers. Each partition of the hard disk in an operating system is the logical copy of original hard disk. An extension of this idea is what is used in cloud computing. Multiple logical computers are created on a single physical server.

Computers of today are underutilized and can be better utilized by running more programs. This was the initial push towards cloud computing and the idea evolved to its present state where multiple users create their own "computers" on a public cloud. Each one of these virtual computers can run different operating systems without interfering with each other. This is made possible by a specialized software called *hypervisor*. The hypervisor is responsible for enforcing isolation between the virtual machines (VMs) that coexist on the same physical hardware. The hypervisor manages any system calls made by the VMs. Hypervisor can be bare metal (type 1), running directly on top of the hardware, or they can run on top of a base OS (type 2). Public cloud providers make use of the type 1 hypervisor for their performance benefits. *Figure 4.1* illustrates these hypervisor types.

In the recent years, hypervisors have been modified to support features that would allow the public cloud providers to maximize profits. These techniques involve resource reduction as their common denominator. By doing so, additional logical computers can be

realized on the same physical rack, without any additional hardware. One such feature is memory deduplication, a feature that is used to reduce memory (DRAM) usage.

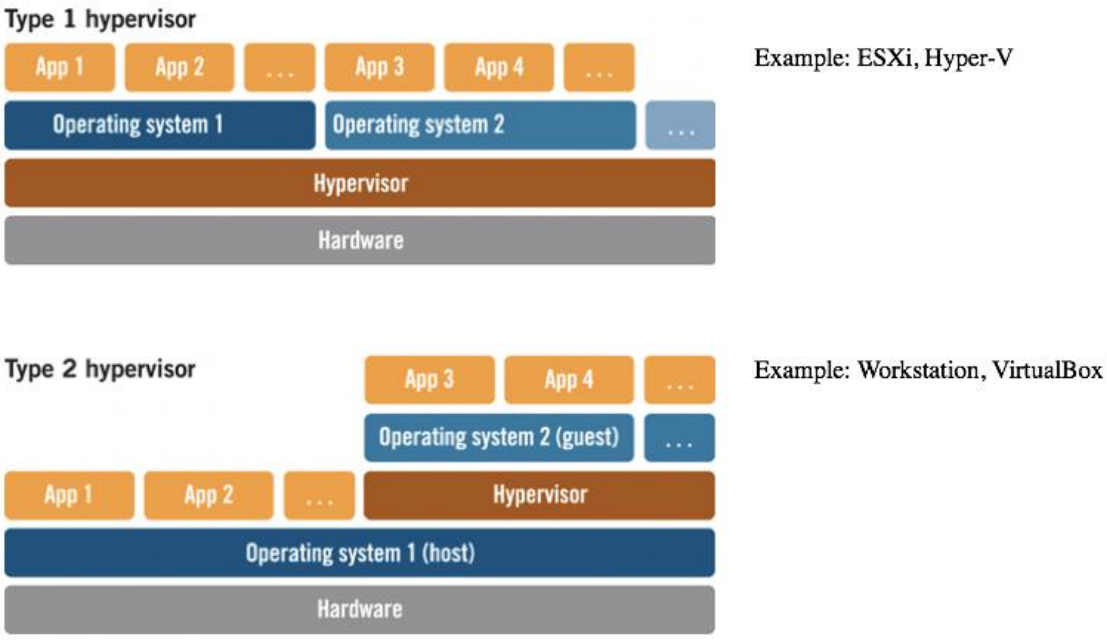


Figure 4.1 *Hypervisor types.*

4.2 Memory deduplication

Memory deduplication is a memory sharing feature implemented in popular open-source and commercial hypervisors. The list includes Linux KVM, VMWare ESXi and Xen. The idea is that if multiple memory pages have the same content, then the hypervisor could maintain one copy for these pages and point the PTEs of those programs to that common physical memory location. If one of the shared (deduplicated) pages is modified by any one of the VMs that is sharing the page, a copy-on-write (CoW) is issued. In other words, a copy of the page made before performing the write operation. The write operation takes effect on the

copy that belongs to the VM that issues the write operation. This allows for significant memory space savings when there are many identical pages across VMs.

Although the technique has space saving benefits, it comes with its share of unexpected security vulnerabilities. In a virtualized environment where both an attacker and a victim virtual machine might be co-hosted, the attacker can gather memory information from the victim. Because of an extra copy operation, the access times incurred for performing write operation to a deduplicated page and a non-deduplicated page are different. Based on this difference in access times, the attacker can determine whether a given page is present in the memory of a co-hosted VM by loading the page into its own memory space and waiting until the contents are merged into a single memory location. A subsequent write to that page takes longer time than a write operation to a normal page. This time difference serves as a side channel that reveals the contents of the victim VM, resulting in information leakage.

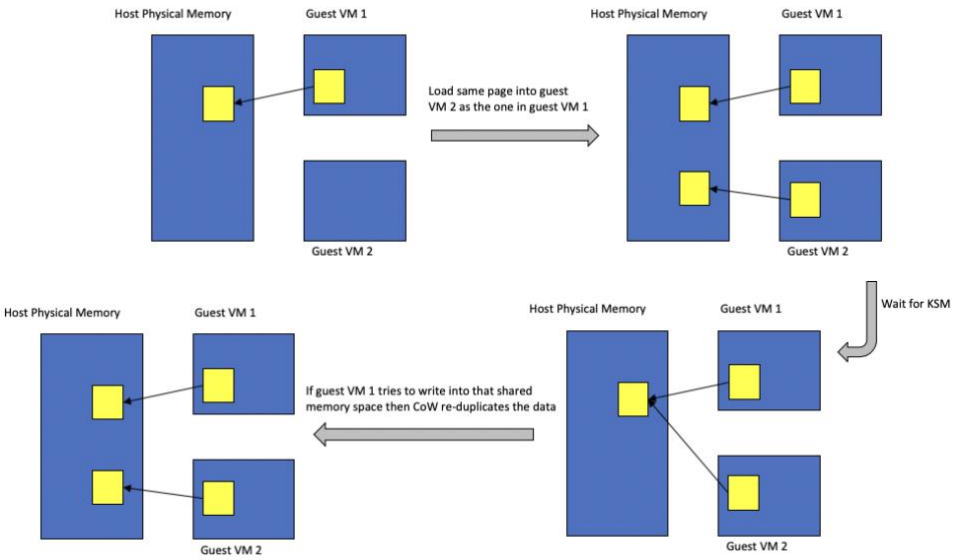


Figure 4.2 *Kernel same-page merging*

CHAPTER 5: ROW HAMMER IN MULTI-TENANT CLOUD SETTING

5.1 Address translation in VM:

On a machine running Linux natively, the virtual to physical address translation is available in the */proc/self/pagemap* file. In a virtualized setup, the guest OS is provided with a pseudo address translation. Which means that the page table available to the guest OS only generates a pseudo physical address. These addresses are handled by the hypervisor. The hypervisor has a shadow page table which contains the mapping for pseudo address (guest physical address) to the actual physical address. Thus, the guest OS does not have access to these addresses, which renders the row targeted row hammer attack difficult to mount.

The translation process described above is slow because of the double page walk. Newer para-virtualization technology allows guest OS to store the actual physical addresses to speed things up. As a result, targeted row hammer attacks can be mounted in a para-virtualized setup.

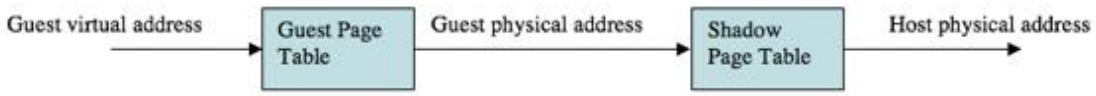


Figure 5.1 *Address translation in VM*

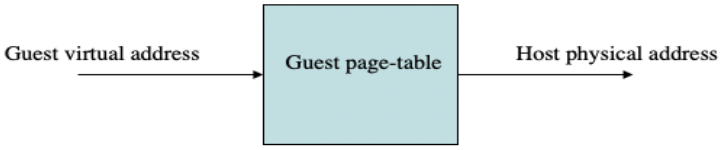


Figure 5.2 *Address translation in Para-Virtualized VM*

To mount an effective row-hammer attack, selection of aggressor rows belonging to the same bank is essential. In order to mount an attack in a para-virtualized setup (no shadow page table):

- Look for bit flips in your VM's (attacker VM) address space.
- If found, release the memory and load the target page into that location.
- Force the victim VM to load the page into its memory.
- Wait for KSM feature to merge the page into your (attacker) physical address and update the PTEs.
- Hammer the adjacent rows to get a flip.
- CoW does not work here because row hammer is not a traditional write operation (does not go through CPU).

CONCLUSION

The attack scheme discussed in this paper allows an attacker to compromise co-hosted cloud VMs with relatively low effort level. Newer chips from intel and AMD support hardware virtualization (intel-VT-x and AMD-V). The translation from guest physical address to host physical address is done in the hardware. In this case, a brute force single sided row hammer is the alternate attack scheme for a cross VM attack, which makes it relatively ineffective. Therefore, it is essential that the cloud vendors make a switch from para-virtualized setup to hardware-virtualized setup in order to mitigate the malicious effects of row hammer in a public cloud.

REFERENCES

1. SEABORN, M., AND DULLIEN, T. Test DRAM for bit flips caused by the row hammer problem. https://github.com/ google/row hammer-test, 2015. Retrieved on July 27, 2015.
2. SEABORN, M. How physical addresses map to rows and banks in DRAM. http://lackingrhoticity.blogspot.com/ 2015/05/how-physical-addresses-map-to-rows-andbanks.html, May 2015. Retrieved on July 20, 2015.
3. XIAO, J., XU, Z., HUANG, H., AND WANG, H. Security implications of memory deduplication in a virtualized environment. In Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'13) (June 2013), Ieee, pp. 1–12.
4. P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks," in USENIX Security Symposium, 2016.
5. Gruss, D., Maurice, C., Mangard, S.: Row hammer.js: a remote software-induced fault attack in JavaScript. In: Caballero, J., Zurutuza, U., Rodríguez, R.J. (eds.) DIMVA 2016. LNCS, vol. 9721, pp. 300–321. Springer, Cham (2016).
6. XIAO, Y., ZHANG, X., ZHANG, Y., AND TEODORESCU, M.- R. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In 25th USENIX Security Symposium (2016).
7. K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos. Flip Feng Shui: Hammering a Needle in the Software Stack. SEC'16.
8. J. Ahn et. al., "Revisiting Hardware-Assisted Page Walks for Virtualized Systems", ISCA'12.
9. R. Bhargava et. al., "Accelerating Two- Dimentional Page Walks for Virtualized Systems", ASPLOS'08.
10. RowHammer Discussion Group. https://groups.google.com/forum/#!forum/rowhammer-discuss.
11. Y. Kim. Flipping Bits in Memory Without Accessing Them. http://users.ece.cmu.edu/~omutlu/pub/ dram-row-hammer_kim_talk_isca14.pptx.
12. 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 S&P'18.
13. SEABORN, M., AND DULLIEN, T. Exploiting the DRAM Rowhammer Bug to Gain Kernel Privileges. BHUS'15
14. O. Mutlu, "The RowHammer problem and other issues we may face as memory becomes denser," in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017.