GHAMMER: A GPGPU Rowhammering Attack

Georgios Anagnopoulos
FORTH
ganagno@ics.forth.gr

Sotiris Ioannidis
FORTH
sotiris@ics.forth.gr

Preserving the security and privacy has always been a fundamental procedure in the development of computing systems. The research community continually advances the state-of-the-art proposing new attacks and defences to mitigate them. In this work, we focus on side-channel attacks. These attacks exploit information that originate from the implementation of a system, rather than locating weaknesses in the implemented system itself. Meta-information, such as time, power and radiation, are used for timing, power-monitoring, and electromagnetic attacks, which eventually lead to data leakage (e.g. passwords, cryptographic keys). A vastly known software-initiated fault attack is the so-called Rowhammer [1]. Rowhammer is a side effect in Dynamic Random-Access Memory (DRAM) that causes a charge leak to memory cells, which can eventually alter the contents of adjacent memory cells that were not even addressed. This phenomenon occurs due to the high density in cells within the memory module and can be triggered by access patterns that excessively activate the same memory rows numerous times. *Sequential*, *Strided*, and *Nearest neighbour* are some instances of memory access patterns. The first occurence of the term Rowhammer was stated in Yoongu Kim's paper [2], that leaded the way for similar kind of attacks. Then, GoogleProjectZero used that effect for privilege escalation [3]; running Native Client code to escape the NaCL sandbox, acquiring the ability to call the hypervisor's syscalls directly. Recently, GLitch [4] was created, a GPU WebGL rowhammering project exploiting Android phones via remote Javascript execution. In our project, we examine the feasibility of the Rowhammer attack in GPGPU hardware architectures and provide insight regarding our results.

GHammer's architecture consists of two major parts as shown in Figure 1: (i) CPU serial code that allocates the host memory and for every set of pointers launches a kernel, (ii) GPU kernels that hammer the set of pointers for a specific number of iterations. It is based on the double-sided version of Rowhammer, which means that we access row A and row C and check for potential bitflips in row B. We have implemented four simple techniques: (i) Sequential Hammer, (ii) Strided Hammer, (iii) Flushing Hammer and (iv) Fenced Hammer. In *Sequential Hammer*, all threads access the same pointers concurrently in a linear sequence. In *Strided Hammer*, threads access the locations based on their thread index (using the runtime environmental variables provided by the CUDA API). In *Flushing Hammer*, half threads hammer the target pointers and the rest read a

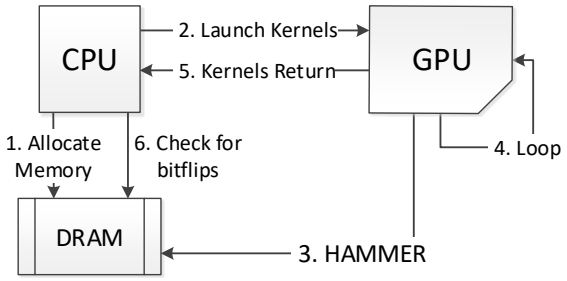

Figure 1. GHammer Process

pre-allocated dummy memory fragment. This results in filling the L2 device cache with dummy data replacing the sensitive ones, so as to "re-hammer" the target that is not cached but fetched from the psysical memory directly. In *Fenced Hammer*, kernels are synchronized with the host using `__threadfence_system()` at every hammer iteration, achieving the desired memory consistency. Our measurements show that still no bitflip errors occur from the device to the host physical memory, even though using state-of-the-art hardware. We believe that this is due to two main reasons. Firstly, Nvidia's current architecture prohibits cache manipulation; although `nvcc` offers cache load and store operators, there are just hints and do not enforce the compiler to include them. Secondly, the preferred data rate for the attack to be achieved is not yet clear. In our system, we were able to produce a CPU-based Rowhammer so our DRAM modules were susceptible to this kind of exploit. We believe that the number of memory accesses between the host and the discrete GPGPU is capped to the PCI Express bandwidth, thus resulting in a slowdown. Upcoming technologies such as Nvidia's *NVlink* [5], might offer us the ability to successfully produce the exploit taking advantage of the 10X higher bandwidth compared to the currently used PCIe Gen 3. Moreover, as future work we plan to use the integrated GPU, found within the CPU die, in order to trigger the exploit.

References

[1] Rowhammer [https://en.wikipedia.org/wiki/Row_hammer]

[2] Yoongu Kim et al, Flipping Bits in Memory Without Accessing Them, 2014

[3] GoogleProjectZero, Exploiting the DRAM rowhammer bug to gain kernel privileges, 2015

[4] Pietro Frigo et al, Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU, 2018

[5] NVIDIA NVlink [https://www.nvidia.com/en-us/data-center/nvlink/]