POSTER: Grand Pwning Unit: remote GPU-accelerated microarchitectural attacks

Pietro Frigo
Vrije Universiteit
Amsterdam
p.frigo@vu.nl

Cristiano Giuffrida
Vrije Universiteit
Amsterdam
giuffrida@cs.vu.nl

Herbert Bos
Vrije Universiteit
Amsterdam
herbertb@cs.vu.nl

Kaveh Razavi
Vrije Universiteit
Amsterdam
kaveh@cs.vu.nl

***Abstract*—Dark silicon is pushing processor vendors to add more specialized units such as accelerators to commodity processor chips. Unfortunately this is done without enough care to security. In this research we look at the security implications of integrated Graphics Processing Units (GPUs) found in almost all mobile processors. We demonstrate that GPUs, already widely employed to accelerate a variety of benign applications such as image rendering, can also be used to "accelerate" microarchitectural attacks (i.e., making them more effective) on commodity platforms. In particular, we show that an attacker can build the necessary primitives to deploy effective side-channel and Rowhammer attacks from JavaScript. We then prove their efficacy by showing the first end-to-end microarchitectural compromise of the Firefox browser running on a mobile phone in under two minutes by orchestrating our GPU primitives.**

I. Introduction

Microarchitectural attacks are increasingly popular for leaking secrets such as cryptographic keys or compromising the system by triggering Rowhammer bit flips in memory. Recent work shows that these attacks are even possible through malicious JavaScript applications, significantly increasing their real-world impact. To counter this threat, the research community has proposed a number of sophisticated defense mechanisms. However, these defenses implicitly assume that the attacker's capabilities are limited to those of the main CPU cores.

In this paper, we revisit this assumption and show that it is insufficient to protect only against attacks that originate from the CPU. We show, for the first time, that the Graphics Processing Units (GPUs) that manufacturers have been adding to most laptops and mobile platforms for years, do not just accelerate image rendering and a host of other benign applications, but also boost microarchitectural attacks. Worse, attackers can unlock the latent power of GPUs even from JavaScript code running inside the browser paving the way for a new and more powerful family of remote microarchitectural attacks. In this paper we show how the GPU, through the JavaScript WebGL API, provides an attacker with all the necessary primitives to carry out microarchitectural attacks. As a proof of this processor's power we show how we can combine these primitives to build the first GPU-*accelerated* remote Rowhammer attack on ARM that compromises the Firefox browser in under two minutes without relying on any software bug.

II. Threat Model

We consider an attacker with access to an integrated GPU. This can be achieved from JavaScript (and WebGL) when the user visits a malicious website. To compromise the target system, we assume the attacker can only rely on microarchitectural attacks by harnessing the primitives provided by the GPU. We also assume a target system with all defenses up.

Experimental Environment: We run our experiments against the Adreno 330 GPU. This GPU is embedded on the Qualcomm Snapdragon 800 and 801 SoCs found on smartphones such as the LG Nexus 5 or HTC One M8.

III. WebGL

WebGL is the result of the increasing demand of porting graphically intensive applications to the Web. This API exposes GPU acceleration to the Web to bolster the development of such applications. It is currently supported by every major browser and allows seamless translation of almost every OpenGL ES application to the Web.

IV. Attacker Primitives

"*Microarchitectural attacks*" aim to either (*a*) steal data using variety of side channels or (*b*) corrupt data using hardware vulnerabilities such as Rowhammer. Here we introduce the main primitives required to build such attacks and we show how we manage to obtain them from the GPU.

#P1. Shared resources:

Prerequisite of any microarchitectural attack is having access to resources shared with other (distrusting) processes. Typical resources targeted from microarchitectural attacks are the CPU caches or system memory. Integrated GPUs need to share the memory with the other processors on the die. As a consequence, DRAM represents the perfect target.

The rendering pipeline: The rendering pipeline relies on developer-provided programs called *shaders*. The pipeline can be divided in 4 steps. ① The CPU provides the GPU with vertices as inputs. Then ② the GPU runs the *vertex shader* on every vertex constructing polygons from these inputs. These polygons are composed of different fragments ($\approx$ pixels). ③ Each of these fragments then gets sent to the *fragment shader* who fills them with colors usually extracted from textures (i.e., *texture sampling*). Finally ④ the outcome gets

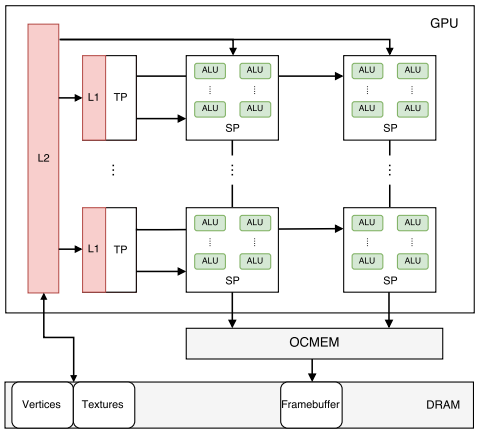

Fig. 1: Building blocks of an integrated GPU

exposed to the Framebuffer (≈ screen). Since textures are stored in system memory, *texture sampling* represents the best mean to build our primitive *P1* and get access to DRAM.

The GPU architecture: The fundamental units of the GPU are the *Stream Processors* (SPs) that are in charge of running the shaders (Figure 1). Shaders running on the SPs can then query the *texture processors* (TPs) to fetch additional input data from the textures residing in DRAM. Between these processors and the data stored in DRAM reside two level of caches. These caches are small and implement a FIFO (*deterministic*) replacement policy. This makes it easier to control memory accesses from the GPU.

#P2. Timers:

To build powerful timing side channels the attacker needs to be in possession of high-resolution timers. WebGL provides two main tools to build such timers. ① Explicit timers can be recovered from the `EXT_DISJOINT_TIMER_QUERY` WebGL extension. These use the internal GPU clock to measure execution time. Otherwise, ② other implicit timers can be built by using the `WebGLSync` objects introduced with the WebGL2 specification for synchronizing CPU and GPU.

Both timers' families provide sub-μs resolution. In particular, ① can reach up to $2\,ns$ resolution.

#P3. Knowledge of the physical location:

In order to build efficient Rowhammer attacks we need to know the physical layout of memory allocations. On ARM platforms, where huge pages are not available, van der Veen et al. [2] relied on contiguous DMA allocations. Since this solution is not available from JavaScript we need to rely on a timing side channel to gain this primitive. This side-channel attack builds upon our primitive *P2* and it exploits the nature of DRAM reads.

Every time an application issues a read the requested row needs to be *activated*; i.e., moved to the *row buffer*. The row buffer contains the currently active row. This means that accesses to adjacent rows require the data of the active row to be restored before the new row can be loaded in the row buffer (i.e., *row conflict*). This time difference can be measured to discern if the underlying memory allocation is contiguous.

#P4. Fast memory access:

Rowhammer attacks require fast access to memory to induce bit flips in other DRAM cells residing on different rows. JavaScript-based Rowhammer implementations have always relied on cache eviction to trigger bit flips. However, this technique has been proven ineffective on ARM due to the slow eviction process [2]. Unfortunately, the GPU also includes two level of caches, making it necessary to still rely on this approach to trigger bit flips. However, opposed to CPU cache eviction, its GPU counterpart turns out to be successful due to the small size and FIFO replacement policy of its caches.

We evaluated the mean access time between two accesses used for *hammering*. This resulted in $\sim 180\,ns$, which is lower than the maximum threshold value reported by van der Veen et al. (i.e., $\sim 260\,ns$) [2].

V. Exploiting the GLitch

To demonstrate the effectiveness of the 4 primitives presented in Section IV here we present GLitch: our remote end-to-end exploit that allows an attacker to escape the Firefox JS sandbox on Android platforms. This exploit is the first instance of remote Rowhammer exploitation on ARM devices. Furthermore, it is the first instance of a remote Rowhammer exploit running in a reasonable time (i.e., less than two minutes on avarage), bringing microarchitectural attacks always closer to a plausible threat model. The attack follows the *Flip Feng Shui* methodology. It starts with (i) *memory templating*, which consists in identifying the exploitable bit flips, then it performs (ii) *memory massaging*, which allows to reliably drop sensitive data on the vulnerable page and finally (iii) *exploitation*.

The exploit relies on a primitive known as *type flipping* [1]. This primitive takes advantage of the *NaN-boxing* technique which repurposes the $2^{52} - 1$ unused NaN values of the IEEE-754 double specification to encode other data — in this case pointers. Which means that a carefully triggered bit flip can turn pointers into doubles and vice versa. This allows an attacker to gain an arbitrary read/write primitive by first breaking ASLR with a 1-to-0 bit flip and then crafting a reference to a fake `ArrayBuffer` with a 0-to-1 bit flip.

VI. Conclusions

We showed that it is possible to perform advanced microarchitectural attacks directly from integrated GPUs found in almost all mobile devices. These attacks are quite powerful. For example, we showed for the first time that by relying only on GPU-accelerated microarchitectural attacks a malicious user can fully compromise a browser running on a mobile phone in less than 2 minutes. We hope our efforts make processor vendors more careful when embedding the next specialized unit into our commodity processors.

References

[1] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector," in *S&P'16*.

[2] V. van der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer Attacks on Mobile Platforms," in *CCS'16*.