Security Implications of GPUs

Charalampos Stylianopoulos

Abstract. GPUs have become popular computing platforms for a wide spectrum of applications, offering promising acceleration capabilities through unique hardware features. In the realm of security, GPUs have had dual use: as means to increase systems security and as a source of vulnerabilities to compromise systems. We summarise two recent results in the field: PixelVault, a system that uses a GPU to protect cryptographic keys, as well as work that exploits GPU vulnerabilities to inspect user's data.

1 Introduction

Graphics Processing Units (GPUs) are standard components of modern computing systems, from massive servers to tablets and mobile phones. Initially designed for graphic rendering, in the last decade, GPUs have gained popularity as acceleration platforms for any general purpose computation (GPGPU computing). Applications that employ heavy computations and data parallelism are typical targets for GPU acceleration [7].

Recently, GPGPU computing paradigms have made their way into the field of security. GPUs have been successfully employed to speedup the computation involved in many security applications. Early work accelerated cryptographic operations on GPUs, exploiting the available parallelism to achieve significant speedups [8]. In a similar manner, Intrusion Detection Systems and kernel integrity monitors are example applications that have been ported to GPUs [9, 10].

On the other hand, GPUs can be used as potentially powerful attack tools [11, 12]. Recent work has also started investigating the security vulnerabilities of GPUs themselves, indicating that they can leak sensitive data [1, 13].

1.1 Brief Background

GPUs typically act as separate co-processors, connected to the host via a serial bus. They contain several multiprocessors, with each multiprocessor consisting of simple stream processors (SPs) that can be used for general purpose computation [4] (see fore example figure 1). Compared to CPUs, that have sophisticated control logic (e.g. branch prediction, out of order execution), GPU execution relies on many, simple threads that operate on a Single Instruction Multiple Threads fashion (the same instruction is executed in parallel, on different data, by many threads).

Stream processors within a multiprocessor share on-chip memory, that may be configured in a way that part of it is shared memory (directly addressable

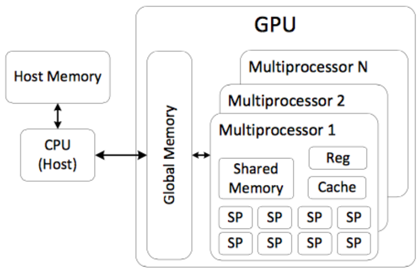

Fig. 1. A simplified view of a typical GPU memory hierarchy from [2]

by the programmer) and the rest is L1 cache (transparent to the programmer). On some architectures, different multiprocessors may share a common L2 cache. The global device memory is the lowest level in the memory hierarchy of the GPU, accessible by all threads.

There are two main APIs for general processing in GPUs, that allow the programmer to have control over the GPU: CUDA and OpenCL [5, 6]. Although each API uses it's own terminology, in rough terms, they enable the following basic steps of GPU execution: (i) Transfer input from the host to the device; (ii) launch kernels (functions written to run on GPUs); (iii) transfer results back to the host.

2 Frontier Paper 1: Information leaks through the GPU

Lee et al. [1] are among the first to explore memory related vulnerabilities of GPUs and present attacks that manage to inspect victim's user data left over in GPU memory. They demonstrate the applicability and effectiveness of their attacks by inferring which web-page the user is browsing, after inspecting the rendered pixels.

In more detail, Lee et al. identify that the memory management of modern GPUs raises many security concerns. The foremost source of vulnerabilities comes from the way new memory is allocated. Upon allocating new memory to a user, the GPU drivers do not initialise the previous contents, allowing the attacker to read any data that has been left over on the global or local memory. In fact, even if the user manually invalidates some of their sensitive data, a certain portion of the programs memory (e.g. kernel code, arguments) is still available for the attacker to see. Moreover, the fact that in the GPU compute paradigm the programmer has the ability to manually access and configure parts of local memory, allows the attacker to gain access even to user data saved in local and private memories.

The authors introduce two simple attacks that exploit these vulnerabilities, in a scenario where the attacker and the victim are both users of the same system and they share a GPU for rendering and computations. The End-of-Context attack, consists of a loop that monitors GPU available memory to determine when the victim application finished using the GPU, then attempts to allocate all available GPU memory to read the victims data. The End-of-Kernel attack is launched between different kernels run by the victim application and attempts to read data kept in the local and private memory. Lastly, a complementary attack that targets specifically AMD GPUs, relies on the running time variation of a kernel to determine if a victim application is running, then reads the users memory before they are able to cleanup and overwrite their sensitive data.

To demonstrate the applicability of their approach, the authors launch these attacks to infer the web-page that a user is browsing, either in Chromium or Firefox. They compare the data gathered by the attack with pre-computed profiles of popular websites and compute a measure of similarity to correctly infer the rendered web-page. In the paper they propose 3 such measures: a Jacard Index based on the set of pixel sequences captured, the Euclidian distance between the histograms of two pixels and a combined approach that uses both metrics. The attacker is able to correctly infer the web-page most of the times, though for high accuracy, the approach requires enough knowledge of the user's web browser and GPU type.

3 Frontier Paper 2: PixelVault

PixelVault by Vasiliadis et al. [2] is a system for securing cryptographic keys and performing computations exclusively on the GPU. The idea is that, by keeping the secret keys in GPU registers, an attacker that has fully compromised the host will still be unable to retrieve the keys.

The security of computations in PixelVault relies on the following concepts: First, a kernel is created on bootstrap that stays active forever and consumes all available device resources, to make sure that it has exclusive control of the GPU. Second, the shared keys are always kept in GPU registers, that are reported to be automatically zeroed in the event that a kernel is terminated.

Following the above design principles, any computation with the secret keys is done using those registers, thus the keys are never leaked into the rest of the memory subsystem (which might be readable by an attacker). Sensitive data that does not fit in the register file, is kept encrypted in device memory. Moreover, the authors make sure that the entire source code of the system resides in instruction cache, to defend against code modification attacks.

The authors evaluate AES and RSA prototypes implemented on PixelVault that protect the secret keys in case of a full system compromise. As a side benefit, they report better encryption/decryption throughput, compared to CPU execution.

3.1 Attacks on PixelVault

Building a secure computation system on top of a GPU is a promising approach. However, systems like PixelVault rely on hardware and API features that are, often purposely, not thoroughly documented by the GPU vendors and may be subject to change. Attackers may exploit subtleties in those features to effectively bypass protection mechanisms.

Indeed, Zhu et at. [3], show how undisclosed or recently introduced features may negate the security guarantees of PixelVault. Exploiting special purpose registers, they are able to flush PixelVault instructions from the caches and replace them with their own malicious code. Using recent changes in debugging support, they are also able to stop a kernel execution and read the GPU registers, effectively extracting the secret keys.

References

1. S. Lee, Y. Kim, J. Kim and J. Kim, Stealing Webpages Rendered on Your Browser by Exploiting GPU Vulnerabilities, 2014 IEEE Symposium on Security and Privacy, San Jose, CA, 2014, pp. 19-33.
2. Giorgos Vasiliadis, Elias Athanasopoulos, Michalis Polychronakis, and Sotiris Ioannidis. 2014. PixelVault: Using GPUs for Securing Cryptographic Operations. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14). ACM.
3. Zhiting Zhu, Sangman Kim, Yuri Rozhanski, Yige Hu, Emmett Witchel, and Mark Silberstein. 2017. Understanding The Security of Discrete GPUs. In Proceedings of the General Purpose GPUs (GPGPU-10). ACM.
4. Kepler Compute Architecture White Paper http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf
5. CUDA C Programming Guide http://docs.nvidia.com/cuda/cuda-c-programming-guide
6. OpenCL - The open standard for parallel programming of heterogeneous systems https://www.khronos.org/opencl
7. Owens JD, Luebke D, Govindaraju N, Harris M, Krger J, Lefohn AE, Purcell TJ. A survey of generalpurpose computation on graphics hardware. InComputer graphics forum 2007 Mar 1 (Vol. 26, No. 1, pp. 80-113). Blackwell Publishing Ltd.
8. Cook DL, Ioannidis J, Keromytis AD, Luck J. CryptoGraphics: Secret key cryptography using graphics cards. InCryptographers Track at the RSA Conference 2005 Feb 14 (pp. 334-350). Springer Berlin Heidelberg.
9. Jamshed MA, Lee J, Moon S, Yun I, Kim D, Lee S, Yi Y, Park K. Kargus: a highly-scalable software-based intrusion detection system. InProceedings of the 2012 ACM conference on Computer and communications security 2012 Oct 16 (pp. 317-328). ACM.
10. Koromilas L, Vasiliadis G, Athanasopoulos E, Ioannidis S. GRIM: Leveraging GPUs for Kernel Integrity Monitoring. InInternational Symposium on Research in Attacks, Intrusions, and Defenses 2016 Sep 19 (pp. 3-23). Springer International Publishing.
11. Vasiliadis G, Polychronakis M, Ioannidis S. GPU-assisted malware. International Journal of Information Security. 2015 Jun 1;14(3):289-97.
12. Ladakis E, Koromilas L, Vasiliadis G, Polychronakis M, Ioannidis S. You can type, but you cant hide: A stealthy GPU-based keylogger. InProceedings of the 6th European Workshop on System Security (EuroSec) 2013 Apr 19.
13. Di Pietro R, Lombardi F, Villani A. CUDA leaks: Information leakage in GPU architectures. arXiv preprint arXiv:1305.7383. 2013 May 31.