Trustworthy Supercomputing in GPU-accelerated Clusters via Smart GPU Virtualization and Memory Isolation

Cong Liu
Department of Computer Science, The University of Texas at Dallas

Introduction. Graphics processing units (GPUs) are powerful for accelerating computation-intensive data-parallel applications, due to their highly multithreaded architecture and high-bandwidth memory. In order to achieve better performance and energy efficiency, GPU-accelerated systems have become the new standard for high performance computing (HPC) and a critical ingredient in the pursuit of exascle computing. For instance, the U.S. Department of Energy is building two GPU-accelerated supercomputers that will move the world closer to exascale.

Although GPUs are widely used in HPC clusters, their potential security problems are being largely ignored. One of the most serious security vulnerabilities is due to the current GPU memory management schemes. Recent research [3,4] demonstrates that information leakage may occur from one user to another because the GPU driver, the operating system (OS), and the GPU card itself do not implement any security-related memory cleanup measure. For example, widely-used GPUs, including both NVIDIA's and AMD's, do not initialize newly allocated GPU memory which may contain sensitive user data. An attacker can thus obtain sensitive data kept in GPU memory of a victim program by simply dumping the entire GPU memory. Specifically, GPUs incur this serious security vulnerability due to the facts that (*i*) GPUs do not clear newly allocated memory pages and (*ii*) GPUs have in-core memory without security mechanisms.

In a typical HPC environment, this uninitialized GPU memory problem may becomes even worse due to the virtualization technology, which has been widely applied in HPC clusters for various performance and flexibility purposes. With recent GPU virtualization technologies and the corresponding OS support (an overview of such technologies can be found in [5]), it is feasible for multiple VMs to share underlying GPUs in a reliable manner. Unfortunately, current GPU virtualization and memory isolation technologies cannot prevent information leakage. For example, an attacker can launch a virtual machine (VM) after the victim's VM using the same GPU, thus bypassing the GPU memory isolation mechanism. Several recent studies [3,4] demonstrate effective attack methods exploiting such GPU vulnerabilities. Unfortunately, no effective solutions exists in the literature for resolving the security issues. One simple fix would be to force GPUs to clean any newly allocated GPU memory buffer. However, this solution suffers in practice because it incurs an unaffordable amount of overheads, which may negate the performance benefits of using GPUs. According to [3], reinitializing the entire GPU memory space costs approximately three times longer than the GPU context switching time. The resulting state of affairs is rather unsettling: ***GPUs are enabling dramatically better performance and energy efficiency in HPC clusters, but they are causing serious security vulerabilities that cannot be efficiently protected by current solutions.***

Our proposed research. In order to ensure trustworthy supercomputing in GPU-accelerated HPC clusters, we propose to resolve the GPU-incurred security issues by designing and implementing a smart GPU virtualization and memory isolation ecosystem. We identified a key reason why current GPU virtualization and memory isolation mechanisms cannot prevent an attacker from launching an VM after the victim's VM using the same GPU and thus being able to retrieve the victim's data. The key reason is because the current GPU device drivers, including the official drivers or open-sourced drivers that support GPU virtualization (e.g., Gdev [2]), allocate GPU memory blocks with physically contiguous pages for better memory access latency and cache coherency. However, allocating memory in such a contiguous manner may allow attackers to easily dump out all the data residing in a contiguous memory space.

Motivated by this observation, we propose to build GM-Guard: a smart GPU virtualization and memory isolation ecosystem. The key idea behind GM-Guard that enables protection against GPU-incurred information leakage is the ***randomized GPU memory allocation*** technique, as illustrated in Fig. 1. As seen in this figure, cuMalloc(), the original function that must be called to allocate physically contiguous memory

chunks for GPU computations, is modified to allocate memory space at the granularity of 4K bytes page, which is contiguous in virtual memory but physically non-contiguous. The resulting advantage is obvious. Even if attackers are able to dump the entire GPU device memory, they cannot easily interpret the semantics, since all the contents are segmented and randomized.

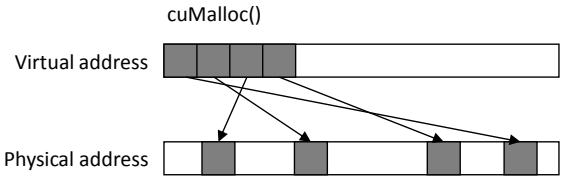

Figure 1: Randomized GPU memory allocation

A potential shortcoming of this approach is that it may introduce additional overheads to translate virtual addresses to non-contiguous physical addresses and fetch/update the corresponding GPU page table. In order to reduce such overheads, we propose to implement an optimized solution by introducing a gpu_freelist structure to avoid repeatedly allocating and deallocating randomized memory chunks. The gpu_freelist structure maintains blocks of free pages with sizes of $2^0, 2^1, ..., 2^k$. An array is maintained for each block size (called the order) that points to a linked list of blocks of free pages, as illustrated in Fig. 2. With the gpu_freelist structure, when a GPU memory allocation request is received, GM-Guard will first check the cached memory blocks in gpu_freelist, and return one block or combinations of blocks that match the requested size. In the memory deallocation stage, memory blocks are not actually freed, but are instead re-linked in the gpu_freelist.

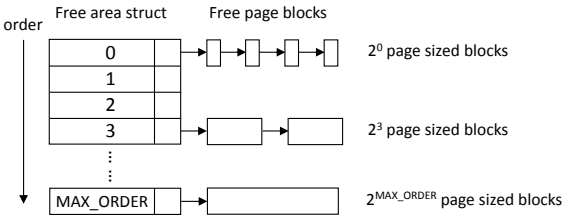

Figure 2: The idea behind implementing the gpu_freelist structure.

Another interesting idea we will explore is to selectively delete used memory. For many cases, only part of the GPU memories contains sensitive information that needs to be protected. For example, some applications use GPUs to increase the performance of encryption algorithms such as AES and RSA. In this scenario, only the keys are critical for the applications. If we can track these keys in the GPU memory and only reinitialize the corresponding memory space, we successfully resolve the GPU data leakage issue while incurring a reasonably low amount of overheads. We plan to implement this idea in GM-Guard by tracking the sensitive data structure at the compiler level.

We will implement GM-Guard leveraging our recently built GPES [7] and GRU [6], which are efficient GPU resource management systems that target at non-virtualized and virtualized environments, respectively. GPES and GRU integrate runtime support for GPUs into the OS, allowing the OS and the user space to use GPUs while protecting GPUs from being excessively occupied by non-privileged or even malicious user-space programs at the OS level. They also incorporate several techniques that make tasks on GPU more preemptible and interruptible in a multi-tasking environment. The prototypes of GPES and GRU were implemented based on the NOUVEAU open-source driver [1], which is the mainstream open-source GPU driver of Linux, and the open-source GPU driver module Gdev [2].

References Cited

[1] FREEDESKTOP. Nouveau Open-Source Driver. `http://nouveau.freedesktop.org`.

[2] Shinpei Kato, Michael McThrow, Carlos Maltzahn, and Scott A Brandt. Gdev: First-Class GPU Resource Management in the Operating System. In *Proc. of USENIX Annual Technical Conference*, pages 401–412, 2012.

[3] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. Stealing webpages rendered on your browser by exploiting gpu vulnerabilities. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 19–33. IEEE, 2014.

[4] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Confidentiality issues on a gpu in a virtualized environment. In *Financial Cryptography and Data Security*, pages 119–135. Springer, 2014.

[5] Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji Kono. Gpuvm: Why not virtualizing gpus at the hypervisor? In *Proceedings of the USENIX Annual Technical Conference*, pages 109–120, 2014.

[6] H. Zhou, Y. Fu, and C. Liu. Supporting Dynamic GPU Computing Result Reuse in the Cloud. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '15)*, to appear, 2015.

[7] H. Zhou, G. Tong, and C. Liu. GPES: A Preemptive Execution System for GPGPU Computing. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, to appear, 2015.