Side Channel Attacks in Computation Offloading Systems with GPU Virtualization

Sihang Liu, Yizhou Wei, Jianfeng Chi, Faysal Hossain Shezan, and Yuan Tian
University of Virginia

***Abstract*—The Internet of Things (IoT) and mobile systems nowadays are required to perform more intensive computation, such as facial detection, image recognition and even remote gaming, etc. Due to the limited computation performance and power budget, it is sometimes impossible to perform these workloads locally. As high-performance GPUs become more common in the cloud, offloading the computation to the cloud becomes a possible choice. However, due to the fact that offloaded workloads from different devices (belonging to different users) are being computed in the same cloud, security concerns arise. Side channel attacks on GPU systems have been widely studied, where the threat model is the attacker and the victim are running on the same operating system. Recently, major GPU vendors have provided hardware and library support to virtualize GPUs for better isolation among users. This work studies the side channel attacks from one virtual machine to another where both share the same physical GPU. We show that it is possible to infer other user's activities in this setup and can further steal others deep learning model.**

***Keywords*-GPU; Side Channel Attack; Cloud; Virtualization;**

I. Introduction

The Internet of Things (IoT) and mobile devices are used to perform heavier workloads nowadays, including image recognition, facial detection, etc. As the need for computation increases while the physical limitation, such as thermal and size of these devices remains, it is impossible to perform all the heavy workloads locally on these devices. A possible solution is to offload these workloads to the cloud. In this scenario, the offloaded tasks from different devices that belong to different users can be executed on the same machine.

Using the cloud to accelerate mobile and IoT devices can certainly extend their capabilities, while concerns on security arise due to the sharing of computation resources. Recent works on side channel attacks, that leverages the information from the system (e.g., cache, memory, etc.) other than the vulnerabilities in the algorithm, have shown it is possible to bypass the existing system-level protection and access the private data of other processes [1, 4, 8, 29, 37, 38, 40]. The recent Spectre [15] and Meltdown [20] attacks have demonstrated the feasibility of breaking the inter-process isolation provided by the operating system. Their success has inspired a variety of research on attacks and defenses on the hardware [2, 5, 6, 14, 16, 32, 36], more specifically, side channel attacks on the central processing unit (CPU). The graphics processing unit (GPU), on the other hand, features high-performance parallel computing. Conventional systems use GPU as a dedicated accelerator for graphics processing, such as video streaming and gaming. Modern workloads, such as big data analytics, machine learning and artificial intelligence, require highly parallel computing that the CPU is not specialized at. As GPU features parallel computing, modern systems typically use GPUs to accelerate these workloads for better performance and efficiency. Therefore, using the GPU to bypass the system isolation becomes a possible approach. Moreover, in cloud computing scenarios, GPUs can be shared among different clients [28] for tasks such as 3D rendering, remote gaming [24, 27], and other acceleration for computation [25, 28]. This resource sharing on GPUs makes it possible for attackers to gain information about other users on the cloud.

The execution model of GPU differs from that of CPU due to its use cases and architectual design. First, GPU features parallel computing and optimizes for throughput. A group of GPU cores can execute hundreds of threads in parallel, where cores running the same instructions can have divergent control flows. To simplify the process of control flows on different threads and improve parallelism, GPUs have execution mask to individually disable some of the code when the branch diverges on different threads. This way, each thread has the same execution time, in spite of their differences in the control flow. For the same reason of optimizing for parallelism and throughput, GPU processors are simpler than those in CPUs, which do not perform speculation. These characteristics make many of the CPU side channel attacks ineffective to GPU. However, there have been works that exploit side channels that exist on GPUs [11, 12, 18, 23] and defend against GPU-specific side channel attacks [13]. Second, a GPU typically works as an external PCI-E device. Therefore, the program does not directly execute on the GPU, instead, a runtime system schedules and offload tasks (program kernels) to the GPU device. This execution model makes the runtime library a potential target for attackers. There have been recent works that launch attacks to access private data of other GPU processes, leveraging the vulnerabilities in GPU libraries and runtime environments [19, 35, 41]. These attacks assume a threat model where the attacker and the victim GPU processes run on the same OS and share the entire GPU system stack, including the GPU library and runtime system.

Recently, major GPU vendors have released GPUs that support virtualization to better satisfy the demand for GPU in cloud computing environments [10, 25, 39]. In a virtualized GPU system, each virtual machine (VM) has an exclusive virtualized GPU (vGPU) and thus having its own GPU library and runtime system. This lower-level, virtualization-based isolation makes cross-VM attacks on GPU more difficult in the following two ways: First, virtualization provides stronger

DOI 10.1109/SPW.2019.00037

isolation. Each VM has its own GPU system stack. Therefore, the weakness in GPU library no longer exists. Second, GPU hardware performance counters are (typically) not available to VMs, therefore, attackers cannot easily launch a side channel attack using these performance counters. Our goal is to exploit the potential vulnerabilities in a virtualized GPU environment.

To overcome the challenges in attacking virtualized GPUs, we have the following key ideas: (1) We observe that even though virtualization has isolated vGPUs, the contention among GPU workloads still exists, as they share the same physical GPU device. Therefore, we can launch probing GPU kernels and measure their execution time as a replacement for the GPU performance counters. The change of prober's execution time provides information about the workload running on the victim VM. (2) The execution time, however, is a low-resolution performance indicator. To overcome this problem, we take an approach from a prior work [23] that utilizes machine learning approaches to better identify the victim's GPU workload.

In this work, We use an Intel's GPU (HD530) and virtualization support (Intel GVT) as our platform, and test our attacking method with three GPU activities and five deep learning models. Applying the aforementioned key ideas, we achieve an F1 Score of 0.95 when identifying the victim's activities, and a 100% accuracy in extracting the deep learning models.

II. Background and Motivation

A. GPU Architecture

GPU features high parallelism and high throughput, being highly optimized for graphics and other parallel general purpose computing tasks. Different from CPUs, a GPU consists of a number of Graphical Processing Clusters (GPCs), each of which includes a group of graphics units such as raster engine and Streaming Multiprocessors (SMs). SMs are all shared among the computational threads that are mapped to it. These units are originally designed for accelerating graphics and multimedia workloads, such as 3D rendering and video streaming. Recently, as the need is growing for workloads such as big data analytics, machine learning and scientific computing, GPUs are also used to accelerate these general purpose applications. There have been library and languages, such as CUDA and OpenCL, to support this general purpose computation on GPUs.

B. GPU Virtualization

Similar to CPUs, GPUs can also be virtualized to provide an abstraction over the hardware to VM users. Typically, there are two ways to divide the GPU resources: physical-slice and time-slice [7]. The physical-slice method allocates different VM separate computation units (GPU cores), while the time-slice method allocates all GPU computation units to one VM periodically. AMD takes the physical-slice approach, and Intel and Nvidia use time-slice. With GPU virtualization, different virtual machines (VMs) have their own exclusive virtualized GPU. Therefore, each VM has its own GPU library. The only sharing resource is the physical GPU device.

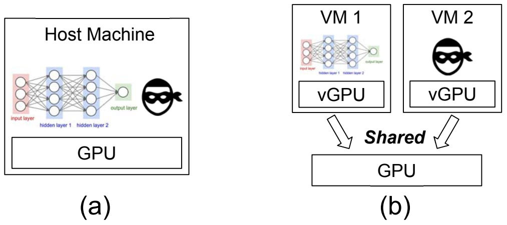


Figure 1. Threat Models. (a) The threat model in prior works: the victim and attacker execute in the same OS. (b) The threat model in this work: the victim and attacker reside in different VMs where each VM has its own virtualized GPU.

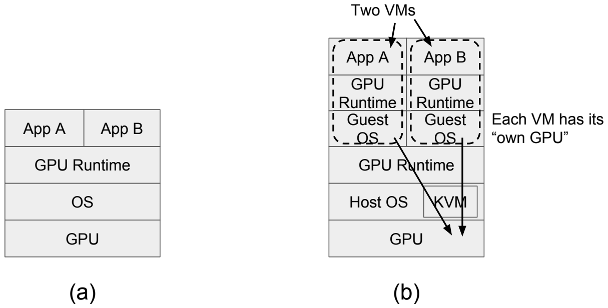


Figure 2. The system stack in (a) normal GPU systems, and (b) virtualized GPU systems.

III. System Overview

This section gives an overview of our system, threat model and attack approaches.

A. Threat Model

Prior works assume that the attacker and victim GPU processes execute on the same machine, as shown in Figure 1(a). In this project, we assume a different threat model, that is the attacker and victim are on different virtual machines. Virtual machines have their own GPU memory, but still share the same GPU device with others, as shown in Figure 1(b).

B. System Setup and Key Ideas

Figure 2 shows the system stack in the aforementioned threat models. Figure 2(a) shows the system stack in prior works where different GPU applications (App A and App B) run on the same GPU runtime in the same OS and access the same GPU device. Figure 2(b) shows the system stack in this work. We assume that the host machine creates vGPUs on top of the physical GPU device (e.g., using Intel GV-T [10]). Then, different guest machines (VMs) use their own vGPU. From the perspective of each VM, it has its "exclusive" access to the GPU. Therefore, each VM has its own GPU system stack, including the driver and runtime system. In our

Category	GPU Workload	Description
Baseline	Idle	Having Ubuntu desktop on, but no GPU load.
Entertainment	Online video streaming	Watch a 4K YouTube video.
	OpenArena	A game that runs on Linux.
Machine learning models	VGG-16 [33]	Deep Neural Networks that run on an OpenCL-based deep learning framework – clDNN [9].
	AlexNet [17]	
	GoogleLeNet-V1,V2,V4 [34]	

Table I
GPU VICTIM WORKLOADS.

system setup, different GPU applications (App A and App B) are isolated by the lower-level virtualization technology. This isolation minimizes the resource sharing between the attacker and victim, as compared to a scenario where the attacker and victim run in the same OS, and share the same GPU library and runtime system. Therefore, the two major attacking methods no longer work:

Library-based Attacks. Different VMs have their own exclusive vGPU device, and the upper-level GPU library and runtime support. As a result, attacks based on flaws in the GPU library and runtime does not work anymore – the underlying virtualization has provided strong isolation between different VMs.

Performance-counter-based Side Channel Attacks. As directly stealing data from the victim via GPU library and runtime is no longer a feasible solution, side channel attack becomes an appealing choice. GPU-based side channel attacks require certain performance-related information, e.g., memory size, computation unit utilization, etc. to infer the details about the victim workload. A typical approach is to read the GPU performance counters. However, virtualization blocks the guest machine (VM) from accessing these performance counters. Therefore, side channel attacks require other indirect methods to infer the victim's activities.

Given that fewer methods are feasible in this virtualized scenario, the difficulty of launching GPU-based attacks increases. Next, we present our *key ideas* that overcome these difficulties.

Probing Program. Due to the unavailability of GPU performance counters, we need a new method to monitor the performance status of the victim VM. We observe that even though virtualization has provided strong isolation between VMs, the resource contention still exist. Therefore, programs that utilize GPU on the victim VM can lead to a performance impact on the attacker VM. To leverage this impact, we run a *probing* program on the attacker side. In the system we study, the proper program consists of a simple read-compute-store pattern that can be contended with the victim both in terms of memory and computation units. The more contention the victim creates, the slower the prober executes. Therefore, the execution time of the prober act as a low-resolution "performance counter". We discuss the details about the prober in Section IV-A3.

Machine-learning-based Characterization. Given performance-counter-like results, the next step is to determine what GPU workload the victim is running. However, due to the low resolution, i.e., the execution time of the prober only approximately tells how much memory and computation contention the victim causes, it is still hard to directly tell what GPU workload the victim is running. To improve the accuracy of identification, we take a similar method as [23], that is to use machine-learning techniques to extract richer features within the raw data (execution time of prober). We detail our method in Section IV-A3, and present our classification results in Section IV-B.

IV. Experimentation

In this section, we present our real-system-based attack. First, we describe the system setup, including the virtualization method, the victim workload, and the attacker's probing program. Second, we detail our machine-learning-based analysis that classifies the GPU workload running on the victim VM.

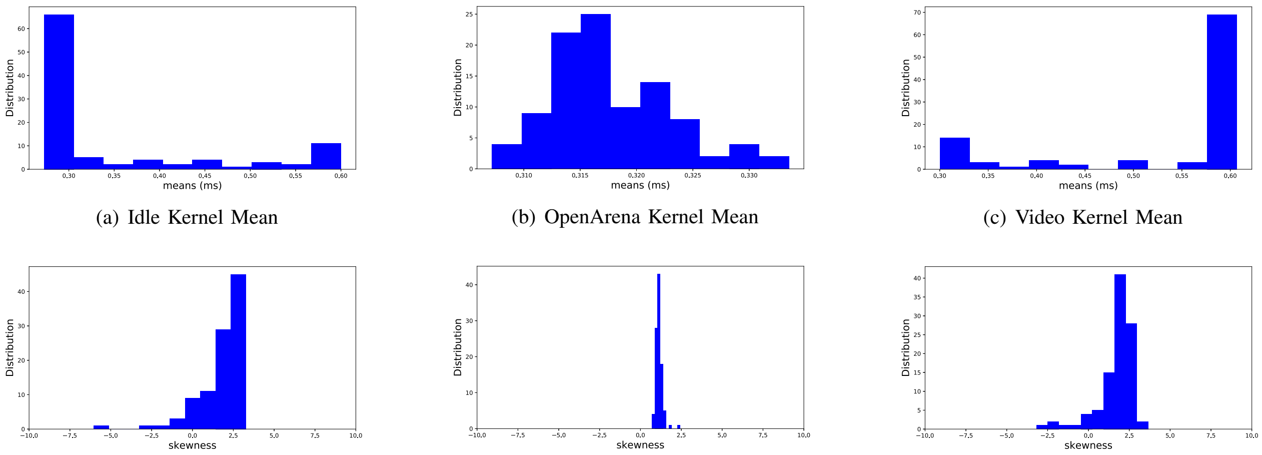
A. Methodology

1) Virtualized System.: In this work, we use a system based on Intel's GPU and virtualization technologies. Table II shows the system configuration. Our virtualization is based on a KVM-accelerated QEMU, that uses the vGPU devices. Note that due to the limitation of GPU memory, we can only instantiate two vGPUs and run at a low display resolution of 960 × 640. One VM acts as the victim that runs the GPU workload, and the other VM runs the prober program. Next, we describe the victim and the attacker in detail.

Hardware	
CPU	Host: Intel i7 6700, 8 logical cores
	Guest: 2 logical cores
GPU	Intel GT2 HD530
Memory	Host: 48GB DDR4, 2133Mhz
	Guest: 4GB
Display	Host: 1920 × 1080
	Guest: 960 × 640
Software	
OS	Host and guest: Ubuntu 16.04
Kernel	Host: 4.17 (with Intel GVT support)
	Guest: 4.15
QEMU	2.12.0 (with Intel GVT support)

Table II
SYSTEM CONFIGURATION.

2) Victim Programs.: We select a few common GPU workloads, as listed in Table I. The baseline is an idle system that has the Ubuntu desktop on, but does not actively use the



(a) Idle Kernel Mean

(b) OpenArena Kernel Mean

(c) Video Kernel Mean

(d) Idle Kernel Skewness

(e) OpenArena Kernel Skewness

(f) Video Kernel Skewness

Figure 3. Feature distribution visualization for side channel attack that infers the victim's activity.

GPU. The second category of applications is for entertainment purposes. We take 4K video streaming and the OpenArena game as examples. The third category is main-stream deep learning workloads. As NVIDIA recently announced their partnership with ARM for accelerating deep learning on IoT devices [26], we expect that deep learning will be a common application in the cloud with vGPUs. We choose five commonly used models: VGG-16 [33], AlexNet [17] and three versions of GoogleLeNet [34]. These deep learning models perform inference over the ILSVRC [31] images. Note that the training procedure can also become a targeting workload, however, due to the computation capability of the Intel integrated GPU, we only consider the inference workloads.

3) Attacker Approach.: To perform an attack on the victim VM, the attacker needs to take two steps: (1) launch the probing program, and (2) interpret the results from the prober to classify the victim's GPU workload. Note that we assume the attacker has a brief knowledge about the victim's GPU workload, such as the commonly used deep learning models that we have listed above.

Probing Program. The probing program consists of an OpenCL kernel that runs on GPU, and a wrapper that offloads this kernel repeatedly. The OpenCL kernel first accesses a random location in a float32 array (the array is on GPU memory), then calculates the square of the selected data value in the array, and eventually writes back this new value to the original location. This simple read-compute-write pattern utilizes both the memory and computation units on the GPU device, therefore any contention from the victim program can change the execution time of this probing OpenCL kernel. The wrapper program uses OpenCL Events (`cl_event`) to measure the execution time of the OpenCL kernel.

Probing Result Interpretation. For each GPU workload, we create a dataset of 1000 probing sequences, where each sequence has 1000 probing data points (execution time of the probing kernel). Then, we use machine learning approaches to interpret these probing results, where we take 80% of the sequences as the training data and the remaining 20% for testing. As our probing results represent the activity over time, we use the common pipeline of time-series analysis to process the results: feature extraction, feature selection and classification.

(1) Feature Extraction: We use the `tsfresh` package [3] to extract the time-series-based features. In total, we extract over 1200 features from the probing results, including the most common features in time series classification such as means, skewness and Kurtosis. We showcase some of the features in Figure 3, where different workloads present distinguishable features.

(2) Feature Selection: We use the univariate feature selection method [21] to extract the features that are most useful to classification. Univariate feature selection works by selecting the best features based on univariate statistical tests. It computes the scoring function (e.g., Chi-squared statistics and mutual information) of each feature and discards the less relevant ones with low scores. In our experiment, we only keep 400 features after feature selection.

(3) Classification: We use different machine learning models to learn the victim's GPU workload based on the features we extracted from the probing results. We use the following models for classification: Random Forest (RF), K-nearest Neighbors (KNN), Support Vector Machine (SVM), Adaptive Boosting (AdaBoost), and Gaussian Naive Bayes (NB).

B. Evaluation Results

We use precision, recall and F1 scores to evaluate the classification results. We present the results using these metrics in Table III and Table IV. Table III shows the results of user (victim) activity classification, that includes the first three activities in Table I. All classifiers indicate that we can successfully infer the user activities. The optimal classifiers

Classifier	Precision	Recall	F1 score
RF	0.96	0.95	0.95
KNN	0.82	0.82	0.82
SVM	0.95	0.95	0.95
AdaBoost	0.86	0.85	0.85
NB	0.74	0.71	0.69

Table III
RESULTS OF SIDE CHANNEL ATTACK ON USER'S ACTIVITIES.

Classifier	Precision	Recall	F1 score
RF	1.00	1.00	1.00
KNN	1.00	1.00	1.00
AdaBoost	0.47	0.60	0.50
NB	0.92	0.90	0.90

Table IV
RESULTS OF SIDE CHANNEL ATTACK ON MODEL EXTRACTION.

for probing the victim's activities are RF and SVM that could achieve an F1 scores of 0.95.

We further demonstrate the model extraction performance in Table IV, that includes the five deep learning models in Table I. The results show that the attacker is also able to infer which deep learning model the victim is running for model inference with 100% accuracy when the attacker using the RF and KNN classifier.

V. DEFENSE

Our attacks require that there exists a significant resource contention between different vGPUs. During our experiment, we find out that the attacker's prober has a noticeable slowdown due to the intensive deep learning workload running on the victim VM. The performance impact finally leads to information leakage. On the other hand, a successful probing program needs to perform a specific pattern of work – repeating a simple GPU kernel for example. This clear pattern can also expose the attacker. To mitigate (in part) the side channel across vGPUs, we have the following proposals.

Side-channel-aware Resource Scheduling. The GPU virtualization support allocates resources to each vGPU and schedules the tasks. Defending against side channel attacks on vGPUs requires the scheduler to expose fewer characteristics about the workload, such as memory and computation intensity. When the GPU resource is sufficient, it is possible to limit the maximum resource that one vGPU can acquire. For example, in our experiments, GoogleLeNets are much more intensive compared to other models. Limiting the maximum resource for each vGPU can reduce the difference in memory/computation intensity between different workloads. This way, the variation of resource utilization (due to workload's characteristics) is lower, and therefore, providing less information to the attacker.

Attacker Behavior Detection. The detection of an attacker requires the hypervisor (e.g., KVM, Xen, etc) monitor the GPU activities on each VM. Existing products or proposals have considered CPU activities, but requires more awareness on the GPU side. Once a potential attacker has been detected, the hypervisor can either change the scheduling mechanism or move the VM to another machine.

VI. RELATED WORKS

GPUs have been widely used for tasks beyond its original multimedia tasks. As GPU computing is becoming more and more general purpose, concerns on its security arise.

GPU Runtime Information Leakage. Recent works demonstrate that the information leakage in GPUs can lead to security issues [11, 19, 22, 23]. There are two major categories of GPU-based attacks. The first type of attacks leverages the design flaws of the GPU library, such as inappropriate data sharing, switching the processes without clearing the buffer, etc. Yao et al. [41] discovers the vulnerabilities in the WebGL library can lead to graphics memory leakage and propose to use virtualized GPU platforms to mitigate this leakage. Pietro et al. [30] showcases leakages in different types of GPU memory, including shared memory, global memory and registers, and successfully steal the SSL key stored in the GPU. In a virtualized system, each VM has its own GPU driver and runtime library. Therefore, it is impossible to read data from the GPU runtime in other VMs.

GPU Side Channel Attacks. The second category of attacks uses the side channel in the GPU device, such as performance counters and cache, to infer information of other GPU processes. Naghibijouybari et al. [23] demonstrate that by leveraging the GPU performance counters, it is possible to gain user information with an OpenGL-based spy process. For example, the attacker can fingerprint the websites' GPU activity and infer the website the user is browsing. Using a similar technique, the attacker can also estimate some of the parameters, such as the number of neurons and the size of input layers, of user's neural network models. Jiang et al. [11] conduct a timing attack on a CUDA-based Advanced Encryption Standard (AES) implementation. Their key idea is to capture the difference in timing between addresses generated by different threads as they access GPU memory. By observing the execution time of an encryption algorithm on GPU, the attacker can infer the likely encryption key. Luo et al. [22] demonstrate a power-based side channel attack on AES encryption executing on a GPU. In virtualized systems, performance-counter-based side channel attacks no longer work as the VM does not have the privilege to access performance counters. As a result, launching side channel attacks to another VM is challenging.

VII. CONCLUSIONS

The integration of GPU in the cloud accelerates mobile and IoT devices, while causing potential vulnerabilities due to resource sharing. The GPU virtualization technologies provide better isolation among vGPU users, making many of the existing side channel attacks ineffective on vGPUs. In this work, we demonstrate that it is still possible to launch side channel attacks on vGPUs.

Acknowledgement

We thank the valuable feedback from the anonymous reviewers from SafeThings, Professor Atul Prakash, and Korakit Seemakhupt.

References

[1] O. Aciicmez, C. Kayakoc, and J. pierre Seifert. Predicting secret keys via branch prediction. In *Proceedings of the 7th Cryptographers' Track at the RSA Conference on Topics in Cryptology*, 2006.

[2] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security)*, 2018.

[3] M. Christ and J. Nils Braun. Tsfresh, 2017.

[4] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo. On the feasibility of online malware detection with performance counters. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013.

[5] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev. BranchScope: A new side-channel attack on directional branch predictor. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[6] B. Gras, K. Razavi, H. Bos, and C. Giuffrida. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In *27th USENIX Security Symposium (USENIX Security)*, 2018.

[7] R. Groves. Virtualized GPUs are now an option for VDI and DaaS! Now what do I do? https://www.brianmadden.com/geekout365/player/5402474608001.

[8] C. Hunger, M. Kazdagli, A. Rawat, A. Dimakis, S. Vishwanath, and M. Tiwari. Understanding contention-based channels and using them for defense. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[9] Intel. Compute library for deep neural networks (clDNN). https://github.com/intel/clDNN.

[10] Intel. Intel graphics virtualization technology (Intel GVT). https://01.org/igvt-g, 2018.

[11] Z. H. Jiang, Y. Fei, and D. Kaeli. A complete key recovery timing attack on a GPU. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[12] Z. H. Jiang, Y. Fei, and D. Kaeli. A novel side-channel timing attack on GPUs. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017.

[13] G. Kadam, D. Zhang, and A. Jog. RCoal: Mitigating GPU timing attack via subwarp-based randomized coalescing techniques. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[14] V. Kiriansky and C. Waldspurger. Speculative buffer overflows: Attacks and defenses. https://people.csail.mit.edu/vlk/spectre11.pdf, 2018.

[15] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. *CoRR*, 2018.

[16] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh. Spectre returns! Speculation attacks using the return stack buffer. In *The 12th Workshop on Offensive Technologies*, 2018.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[18] E. Ladakis, L. Koromilas, G. Vasiliadis, M. Polychronakis, and S. Ioannidis. You can type , but you can't hide : A stealthy GPU-based keylogger. 2013.

[19] S. Lee, Y. Kim, J. Kim, and J. Kim. Stealing webpages rendered on your browser by exploiting gpu vulnerabilities. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 19–33. IEEE, 2014.

[20] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown. *CoRR*, 2018.

[21] H. Liu and H. Motoda. *Feature selection for knowledge discovery and data mining*, volume 454. Springer Science & Business Media, 2012.

[22] C. Luo, Y. Fei, P. Luo, S. Mukherjee, and D. Kaeli. Side-channel power analysis of a GPU AES implementation. In *2015 33rd IEEE International Conference on Computer Design (ICCD)*, 2015.

[23] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh. Rendered insecure: GPU side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[24] Nvidia. Cloud gaming - Gaming as a service (GaaS). https://www.nvidia.com/object/cloud-gaming.html.

[25] Nvidia. Remote visualization on server-class Tesla GPUs. https://www.nvidia.com/content/pdf/remote-viz-tesla-gpus.pdf, 2014.

[26] NVIDIA. https://nvidianews.nvidia.com/news/nvidia-and-arm-partner-to-bring-deep-learning-to-billions-of-iot-devices, 2018.

[27] Nvidia. GeForce NOW. https://www.nvidia.com/en-us/shield/games/geforce-now/, 2018.

[28] Nvidia. GPU cloud computing. https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/, 2018.

[29] C. Percival. Cache missing for fun and profit, 2005.

[30] R. D. Pietro, F. Lombardi, and A. Villani. CUDA leaks: Information leakage in GPU architectures. *CoRR*, 2013.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[32] M. Schwarz, M. Schwarzl, M. Lipp, and D. Gruss. Netspectre: Read arbitrary memory over network. *CoRR*, abs/1807.10535, 2018.

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[35] J. Taft. Gpu security exposed - exploiting shared memory. https://www.blackhat.com/docs/eu-16/materials/eu-16-Taft-GPU-Security-Exposed.pdf, 2016.

[36] A. Tang, S. Sethumadhavan, and S. Stolfo. CLKSCREW: Exposing the perils of security-oblivious energy management. In *26th USENIX Security Symposium (USENIX Security 17)*, 2017.

[37] G. Vasiliadis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis. PixelVault: Using GPUs for securing cryptographic operations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[38] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *2006 22nd Annual Computer Security Applications Conference (ACSAC)*, 2006.

[39] T. Wong. AMD multiuser GPU: Hardware-enabled GPU virtualization for a true workstation experience. https://www.amd.com/en/graphics/workstation-virtualization-solutions, 2016.

[40] M. Wu, S. Guo, P. Schaumont, and C. Wang. Eliminating timing side-channel leaks using program repair. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2018.

[41] Z. Yao, Z. Ma, Y. Liu, A. Amiri Sani, and A. Chandramowlishwaran. Sugar: Secure GPU acceleration in web browsers. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.