LockedDown: Exploiting Contention on Host-GPU PCIe Bus for Fun and Profit

Mert Side
Texas Tech University
Lubbock, TX, USA
mert.side@ttu.edu

Fan Yao
University of Central Florida
Orlando, FL, USA
fan.yao@ucf.edu

Zhenkai Zhang*
Clemson University
Clemson, SC, USA
zhenkai@clemson.edu

***Abstract*—The deployment of modern graphics processing units (GPUs) has grown rapidly in both traditional and cloud computing. Nevertheless, the potential security issues brought forward by this extensive deployment have not been thoroughly investigated. In this paper, we disclose a new exploitable side-channel vulnerability that ubiquitously exists in systems equipped with modern GPUs. This vulnerability is due to measurable contention caused on the host-GPU PCIe bus. To demonstrate the exploitability of this vulnerability, we conduct two case studies. In the first case study, we exploit the vulnerability to build a cross-VM covert channel that works on virtualized NVIDIA GPUs. To the best of our knowledge, this is the first work that explores covert channel attacks under the circumstances of virtualized GPUs. The covert channel can reach a speed up to 90 kbps with a considerably low error rate. In the second case study, we exploit the vulnerability to mount a website fingerprinting attack that can accurately infer which web pages are browsed by a user. The attack is evaluated against popular browsers like Chrome and Firefox on both Windows and Linux, and the results show that this fingerprinting method can achieve up to 95.2% accuracy. In addition, the attack is evaluated against Tor browser, and up to 90.6% accuracy can be achieved.**

1. Introduction

Modern graphics processing units (GPUs) have become increasingly popular over the past decade. On the one hand, they are needed for performing resource-intensive visual computing tasks such as ultra-high-definition video decoding and high-quality 3D graphics rendering. On the other hand, along with the development of general-purpose GPU programming models like CUDA and OpenCL, modern GPUs have also been extensively used for performing massively parallel computing tasks like physical dynamics simulation, data mining, and deep neural network training. In fact, very powerful GPUs designed exclusively for parallel computing have been built and favored in supercomputers. With the advent of the era of cloud computing, such powerful GPUs have also been widely deployed in data centers and offered to customers by major cloud service providers, such as Amazon EC2, Google Compute Engine, and Microsoft Azure, using state-of-the-art GPU virtualization techniques.

*. Part of the work was done while the author was affiliated to Texas Tech University.

Despite the rapidly growing use of GPUs in traditional and cloud computing platforms, the studies on the security of GPUs appear to be lagging behind. Many of the GPU security implications are often overlooked and/or not well understood. Some recent investigations have reflected the fact that GPUs indeed have many exploitable vulnerabilities [11], [19], [24], [29], [39], [40], [65], [66]. These studies signify that there may exist many other security vulnerabilities in GPUs as well as their ecosystems and strongly argue for more research efforts to discover any possible vulnerabilities for a better understanding of the GPU security.

In this paper, we examine the under-explored attack surface of host-GPU communication and disclose a new side-channel vulnerability that can be exploited to mount realistic attacks in the contexts of both traditional and cloud computing. In particular, we have discovered that contention on the PCIe bus, that is, the standard interconnect between a GPU and its host system, is measurable in the form of data transfer latencies. Because the host-GPU PCIe bus is shared among processes running in different security domains, measurable contention on this bus can be leveraged as a side-channel to leak information across strong isolation boundaries. To deliberately cause such contention for exploitation, however, continuous PCIe congestion is needed, which is not easy to achieve in reality due to the high-speed nature of PCIe. With the help of the page-locked memory allocation and transfer feature in CUDA, we make it possible to induce such contention.

To demonstrate the exploitability of this contention-based side-channel vulnerability, we have conducted two attacks. The first one is to construct a covert communication channel that enables data exfiltration between processes having access to the same physical GPU. Different from the prior work that assumes a native execution environment without any GPU resource partitioning [39], we show that our covert channel works effectively and robustly in realistic cloud computing settings, where server-grade GPUs (e.g., Tesla V100) are used and the state-of-the-art NVIDIA GPU virtualization technique is employed. *To the best of our knowledge, this is the first work that explores covert channel attacks under the circumstances of virtualized GPUs.* In addition, our covert channel is not affected by the mitigation approach proposed in [57], because that countermeasure only partitions cache and memory to effectively thwart the covert channel described in [39] but has no effect on the PCIe contention vulnerability exploited by ours.

The second exploitation we have conducted is a web-

site fingerprinting attack that deduces which websites have been visited by a user. Since all major web browsers, such as Google Chrome, Mozilla Firefox, and Microsoft Edge, leverage the GPU to help accelerate web page rendering, when a user opens a web page, the browser will generate a certain GPU workload corresponding to the web page rendering. In other words, there will be traffic on the host-GPU PCIe bus when a web page is rendered. We show that rendering different web pages can create distinguishable patterns of PCIe bus traffic (and thus PCIe contention), which can be exploited to fingerprint the corresponding web pages accurately, even in the anonymity network scenario where Tor is used.

In summary, the main contributions of this paper are:

- We disclose a new exploitable side-channel vulnerability that ubiquitously exists in systems equipped with modern GPUs. This vulnerability is due to measurable contention caused on the host-GPU PCIe bus. To create such contention for exploitation, we identify that the page-locked memory allocation and transfer feature in CUDA can be leveraged.
- We exploit the newly discovered vulnerability to construct a covert communication channel for data exfiltration across isolation boundaries established by up-to-date virtualization techniques. This covert channel can circumvent all the existing defenses against data exfiltration in cloud computing settings, and it can reach a speed up to 90 kbps with a considerably low error rate. Moreover, this covert channel is much more robust in practice compared to the one proposed in [39] under similar circumstances.
- We exploit the newly discovered vulnerability to implement a website fingerprinting attack that can accurately infer the web browsing activities of a user. We evaluate this website fingerprinting attack against popular browsers like Chrome and Firefox on both Linux and Windows. The results show that this fingerprinting method is highly effective where up to 95.2% accuracy can be achieved. We also evaluate this website fingerprinting attack against Tor in the anonymity network scenarios, and up to 90.6% accuracy can be achieved.

Notice that, very recently, Tan *et al.* also exploited PCIe congestion to mount side-channel attacks [51]. Our work and [51] were conducted *completely concurrently and independently*. Although similar, there are still many differences. Firstly, our and their approaches to contention generation differ. Our work focuses on communication between the CPU and the GPU through dedicated PCIe lanes, while the work in [51] focuses on exploiting inter-PCIe device communication over shared PCIe lanes. However, in reality, GPUs are nearly always installed on the PCIe slot that is connected to the CPU via dedicated lanes, as illustrated in Figure 1.[1] In such standard scenarios, no PCIe devices (e.g., NVMe or RDMA NIC) will compete with the GPU for PCIe bandwidth no matter how they are connected to the host (e.g., directly via dedicated lanes or indirectly via PCH). Therefore, the GPU-NIC scenarios studied in [51] may be very uncommon in practice. On the contrary, our work is performed in realistic setups where the GPU uses dedicated PCIe lanes and the system hardware is not intentionally arranged. Secondly, the mounted attacks are not exactly the same. Although the website fingerprinting attack is studied in both our work and [51], we also investigate the first virtualized GPU-based cross-VM covert channel while the work in [51] studies two other attacks that are to steal neural networks and keystrokes. Moreover, in terms of the same website fingerprinting attack, our work studies more GPUs with more web browsers (including Tor) under both Windows and Linux to provide more insights, while the work in [51] only inspects a single GPU using Chrome under Linux.

The rest of this paper is organized as follows: Section 2 lays the necessary background; Section 3 introduces the contention-based side-channel vulnerability; Section 4 and Section 5 describe two exploitations of this newly discovered vulnerability in the contexts of cloud and traditional computing respectively; Section 6 proposes some potential countermeasures; Section 7 gives the related work; and Section 8 concludes this paper.

2. Background

In this section, we describe some background knowledge about modern GPUs as well as the GPU programming model. We also briefly present some information on the PCIe bus that connects a GPU to its host system. In addition, we discuss the GPU virtualization with a focus on how NVIDIA implements it.

2.1. GPU Architecture and Programming Model

Modern GPUs have evolved from graphics accelerators to highly parallel many-core systems that have been used for a wide range of non-graphics applications. In general, a GPU consists of a number of streaming multiprocessors (SMs), each of which can typically run thousands of threads. Due to the large number of threads that can simultaneously run on a GPU, there is a demand for very high memory bandwidth.

To meet the potentially high memory bandwidth demands, GPUs are often equipped with a large amount of off-chip memory. Such a GPU memory is currently of the type GDDR5(X) or GDDR6(X) tailored for providing high bandwidth. Similar to the CPU, caches are utilized in the GPU as well to hide the latency of GPU memory accesses. There are local L1 caches in each SM, and there is also an L2 cache shared by all the SMs.

As shown in Figure 1, the GPU is connected to the host side through a PCIe bus which is described later in more detail. The CPU communicates with the GPU via memory-mapped input/output (MMIO). There are also direct memory access (DMA) engines for transferring large amounts of data over the PCIe bus between the host memory and the GPU memory.

Originally, GPUs could only be programmed using certain APIs such as OpenGL and DirectX for rendering 2D and 3D graphics. As the need for leveraging GPUs to perform massively parallel computing grows, multiple

1. Usually, the first one or two PCIe slots on a motherboard are for dedicated PCIe lanes, which are sometimes called primary PCIe, and the rest of the slots are connected to PCH/PCIe switch(es).

general-purpose GPU programming models have been developed. Among all these models, CUDA developed by NVIDIA is the most prevailing one.

The CUDA programming model allows heterogeneous parallel computing that involves both the CPU and the GPU. The CPU is in charge of the initiation of parallel computing on the GPU, for which it specifies the needed computation in the form of a kernel function and manages the GPU memory allocation/deallocation. Before launching the kernel function on the GPU, the CPU needs to send the needed data over the interconnect to the GPU. When the kernel function is launched, it is executed by a large number of GPU threads. The threads are grouped into warps that are scheduled to run on SMs. After the computation is finished, the CPU can retrieve the results from the GPU over the interconnect. In terms of transferring data between the CPU and GPU, a programmer often uses two functions provided by CUDA, which are `cudaMemcpy()` in its high-level API (CUDA Runtime API) and `cuMemcpy()` in its low-level API (CUDA Driver API). In this paper, we mainly use the `cudaMemcpy()` function in the CUDA Runtime API.

2.2. PCIe Bus

Peripheral Component Interconnect Express (PCIe) is an interface standard for connecting fast I/O devices. It has been developed and maintained by the PCI Special Interest Group (PCI-SIG) as the successor to the older PCI interface [55]. Instead of a shared parallel bus of PCI, PCIe uses high-speed point-to-point serial buses. Currently, PCIe is the *de facto* interface to many high-speed peripherals, including GPU, RAID controller, and NVMe SSD, and it has been universally used on many personal computer and server motherboards.

Although PCIe is based on point-to-point communication, its topology is actually a tree structure, where the root is a component named root complex that connects the CPU and memory system to a PCIe switch. The PCIe switch creates multiple PCIe endpoints to which hardware devices can be attached. The communication between two PCIe interfaced devices is first negotiated to form a point-to-point interconnect referred to as a PCIe link. A PCIe link consists of one or more lanes, and each lane carries a dual-unidirectional byte stream. Depending on the specific implementation, a PCIe link can have 1, 2, 4, 8, or 16 lanes. Newer CPUs also have a limited number of PCIe lanes dedicated to latency-sensitive devices such as GPUs (e.g., Intel Comet Lake Core processors have up to 16 lanes of PCIe 3.0 and AMD Ryzen 5000 series processors have up to 24 lanes of PCIe 4.0), and these lanes are directly connected to the root complex. Note that a link having N lanes is designated as $\times N$. Figure 1 depicts a typical PCIe topology.

In general, the first one or two PCIe slots on a motherboard are connected to the GPU via dedicated PCIe lanes which are typically referred to as the primary PCIe slot. Any additional slots on a given motherboard need to go through PCIe switches commonly located in the Platform Control Hub (PCH) which has rather limited bandwidth. Both NVIDIA and AMD GPU's user manuals suggest connecting their devices to the primary PCIe slots on a given motherboard as shown in Figure 1. Therefore, for

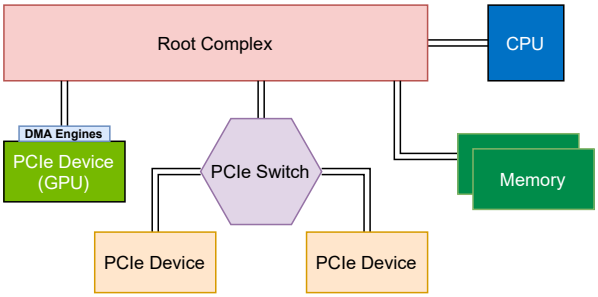


Figure 1: The PCIe topology.

a typical setup (i.e., Intel Comet Lake Core processors) equipped with a GPU, an RDMA NIC and/or an NVMe SSD, the GPU will be connected to the CPU via dedicated PCIe lanes, and the RDMA NIC and SSD will not share the lanes with and interfere with the GPU. (The NVMe SSD is usually connected to the PCH, and the RDMA NIC may be connected to the CPU through another group of dedicated lanes.)

The PCIe protocol mainly consists of three layers, which are the transaction layer, data link layer, and physical layer. The transaction layer is responsible for creating PCIe requests and completion transactions via transaction layer packets (TLPs). A TLP consists of a header, an optional data payload, and an optional TLP digest. If a TLP has a data payload, the size of the data payload must be a multiple of four bytes without exceeding 4096[2]. The data link layer is responsible for ensuring the integrity and ordering of TLPs via an error detection and correction mechanism and a sequence number. The physical layer sends and receives all the data transferred across the PCIe links.

2.3. GPU Virtualization

Virtualization plays an important role in cloud computing, and has become the essential technology to enable sharing various computing resources with multiple parties in an effective, efficient, and secure manner. Infrastructure-as-a-Service (IaaS) clouds such as Amazon EC2, Google Compute Engine, and Microsoft Azure pervasively rely on virtualization to offer users CPU, storage, and networking resources on demand. Although modern GPUs appear hard to virtualize, as the need for using GPUs in cloud computing continuously grows, several GPU virtualization techniques have emerged [9], [36], and some of them have been employed by major cloud service providers [2], [13], [38].

In general, GPU virtualization techniques can be divided into four classes, which are device emulation, driver paravirtualization, fixed pass-through, and mediated pass-through [36]. Although techniques based on device emulation and driver paravirtualization are relatively easy to implement, they usually introduce large performance overheads and are not used in practice [9]. By contrast, techniques based on fixed or mediated pass-through can achieve high performance and fidelity, and they are the

2. The PCIe specification also defined a parameter named maximum payload size (MPS) to further restrict the maximum allowable TLP payload size.

commonly used ones. While fixed pass-through gives a virtual machine (VM) fully exclusive access to a physical GPU, mediated pass-through facilitates Single Root I/O Virtualization (SR-IOV) capable devices to expose themselves as several devices.

NVIDIA implements and offers mediated pass-through GPU virtualization techniques umbrellaed under the NVIDIA virtual GPU (vGPU) architecture [41]. The NVIDIA vGPU architecture specifically centers on a virtual GPU manager program that runs along with a hypervisor. The virtual GPU manager is responsible for creating vGPU devices and partitioning the GPU memory for the created vGPUs. Each vGPU can be assigned to a VM by the hypervisor, and from the perspective of the VM, the assigned vGPU appears like a directly attached physical GPU. Figure 2 illustrates the NVIDIA vGPU architecture.

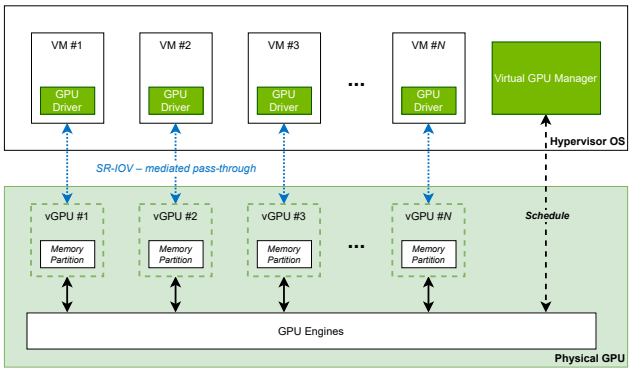


Figure 2: NVIDIA vGPU architecture.

Prior to the Ampere architecture, all the NVIDIA GPUs that can be virtualized leverage only temporal isolation to share the GPU hardware resources (except for the GPU memory, which is spatially partitioned) among the vGPUs. There is a scheduler that assigns each vGPU a time slice, and in the assigned time slice, the vGPU can have exclusive access to many of the GPU resources (e.g., 3D graphics engines and video codec engines). The time slices may have different lengths, and the length of a time slice may be adjusted dynamically. There are also certain resources that can be used by vGPUs without temporal isolation, and one example is the copy engines for data transfers.

With the Ampere architecture, NVIDIA further introduces a new feature named multi-instance GPU (MIG) to enable the virtual GPU manager to fully partition the hardware resources (e.g., SMs and L2 cache) of a server-grade GPU into several instances. At the time of this writing, MIG is still in its infancy, and there are only two GPU models, the NVIDIA A30 and A100, that support this technology, which has not been widely deployed.

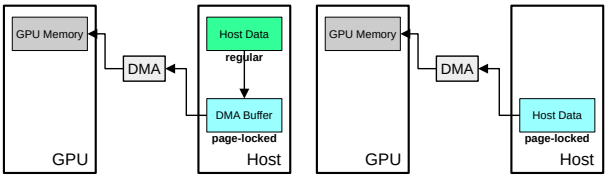
3. Contention on Host-GPU PCIe Bus

As the host-GPU PCIe bus is responsible for transferring data between the host side and the GPU device, we anticipate that contention on this bus can lead to increased data transfer latencies. To make the effect of bus contention observable, we need to find an approach to continuously impose congestion on this bus. In this section, we demonstrate that with the help of the page-locked memory allocation feature in CUDA, we can repeatedly congest the host-GPU PCIe bus such that the use of the bus by others can be detected and measured quantitatively.

3.1. Page-Locked Memory Allocation in CUDA

For the purpose of amplifying the effect of contention on the host-GPU PCIe bus, we attempt to exhaust the bandwidth of the bus as much as possible. To this end, we leverage the `cudaMemcpy()` function to continuously perform data transfers from the host to the GPU via the bus. (We focus only on the host-to-GPU transfer direction in this paper, but note that the opposite direction can be identically used.) However, we find that it may not work as expected if the data to be transferred naively resides in regular host memory pages.

As mentioned in Section 2, DMA takes charge of the host-GPU data transfers on the PCIe bus. Essentially, DMA operates on physical (instead of virtual) addresses directly. To avoid being potentially disrupted by virtual memory management operations like swapping, DMA requires that the memory pages corresponding to the DMA buffer be locked in the host memory, which means that the physical locations of the buffer are immune to any paging activities. Under the circumstances where we need to transfer a piece of data residing in regular host memory pages to the GPU memory, the CUDA runtime first copies the data into a page-locked memory buffer that is temporarily allocated and then uses DMA to transfer the data from this buffer to the GPU memory. This two-step process is illustrated in Figure 3 (a). However, the extra copy adds delays to the `cudaMemcpy()` function, which greatly reduces the achievable bandwidth consumption. For example, the NVIDIA Quadro RTX 6000 uses PCIe 3.0 ×16 whose theoretical bandwidth is 15.8 GB/s, and when we use the `cudaMemcpy()` function to continuously move data in regular pages to this GPU, only about 30% of the bandwidth (4.6 GB/s) is consumed. As shown in Section 3.2, light congestion like this cannot enable us to stably measure contention on the bus.



(a) Two-step data transfer. (b) Direct data transfer.

Figure 3: CUDA runtime data transfers.

Interestingly, CUDA has a feature that can allow a user to directly request allocation of page-locked memory through the `cudaMallocHost()` function or the `cudaHostAlloc()` function. If the data to be transferred is already in such page-locked memory, the CUDA runtime will not involve any extra memory buffer but instead just uses DMA to copy the data from the host to the GPU, as shown in Figure 3 (B). By exploiting this direct data movement, we find that heavy congestion can be imposed on the host-GPU PCIe bus. For example, in case of the NVIDIA Quadro RTX 6000 being tested, we observe that more than 77% of the bandwidth (12.2 GB/s)

is consumed when page-locked memory is leveraged. As demonstrated later, such heavy congestion enables us to measure contention caused by other uses of the bus in a stable way.

3.2. Contention Measurement

Having an approach to heavily congesting the host-GPU PCIe bus, we verify our anticipation that contention on the bus can lead to increased data transfer latencies. To this end, we use a CUDA program *Alice* that measures how much time it takes when using the `cudaMallocHost()` function to send 32 KB data to the GPU under different scenarios. We choose to send 32 KB data based on our observations from experiments shown in Figure 7 and Figure 10. We simply use the `RDTSCP` instruction to measure the time in clock cycles for each `cudaMemcpy()` function invocation. We test two cases where the 32 KB data resides in page-locked memory in one case and in pageable memory in the other case.

To introduce contention to the host-GPU PCIe bus, we use another CUDA program *Bob* that repeatedly invokes the `cudaMemcpy()` function to transfer an amount of data to the GPU. We vary the size of the data to inspect its impacts on the data transfer latency of *Alice*. This contender program *Bob* runs continuously when *Alice* measures the time. We also test two cases where the data to be transferred by *Bob* resides in page-locked memory in one case and in regular pages in the other case.

We perform the contention measurement experiments on an NVIDIA Quadro RTX 6000 GPU. To make the experiments more interesting, instead of directly running *Alice* and *Bob* in a native environment, we execute them in two VMs. The GPU is configured into 6 vGPUs, and we assign one to *Alice* and another one to *Bob*. *Bob* ranges the size of the transferred data from 4 KB to 512 KB. The data transfer latencies measured by *Alice* are shown in Figure 4. (Table 11 in Appendix B reports the specific numbers.) There are four scenarios, and in each scenario, nine latency ranges are illustrated in Figure 4, which are measured under different sizes of data sent by *Bob*, and each range is derived from 1000 measurements.

The "Locked-Locked" and "Regular-Locked" give the results under the scenarios where *Bob* transfers data residing in page-locked memory. The "Base" presents the latencies when there is no contention introduced (i.e., *Bob* is not executed), which serve as the baseline. From such baseline results, we can verify that transferring the same amount of *Alice*'s data takes much less time if her data is in page-locked memory. (For example, in the "Locked-Locked" scenario, it takes 21,440 – 23,281 clock cycles for *Alice* to transfer her 32 KB data, but takes at least 33,406 cycles if her data is in pageable memory.) When *Bob* starts repeatedly transferring 4 KB data to create some contention on the bus, we do not observe much change if the data transferred by *Alice* is in page-locked memory, but we occasionally detect some large delays if the data is in the regular memory. However, when the size of data transferred by *Bob* increases (i.e., the amount of contention increases), we can observe that the minimum latencies in the "Locked-Locked" increase monotonically, but the minimum ones in the "Regular-Locked" change

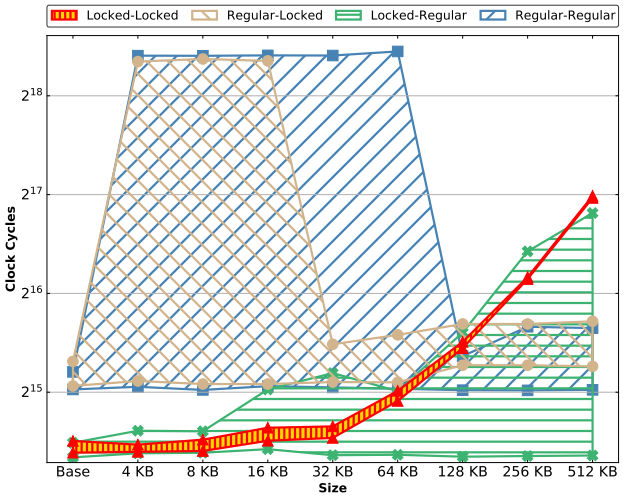


Figure 4: Data transfer latencies measured by *Alice*. The first part of "Locked/Regular – Locked/Regular" indicates whether *Alice*'s data resides in page-locked memory or regular memory, and the second part indicates *Bob*'s.

negligibly; and, the maximum latencies in the "Locked-Locked" also increase monotonically, but the maximum ones in the "Regular-Locked" change irregularly.

Notice that when the size of data transferred by *Bob* is above 16 KB, the resultant latency range will have no intersection with the baseline latency range in the "Locked-Locked" scenario. However, from the results in the "Regular-Locked", we find that even when the size of the data reaches 512 KB, the latency range is still not well separated from the baseline one. Moreover, we can observe that the latency variation in terms of "Locked-Locked" is always within 2300 clock cycles (i.e., less than 0.9 μs given a 2.6 GHz processor), but the variation in terms of "Regular-Locked" can be more than 300,000 cycles (i.e., 115 μs).

As for comparisons, we also present the results when the data transferred by *Bob* is in regular pageable memory. From the results shown in the "Locked-Regular" and "Regular-Regular", we can find that the maximum latencies are similar to the corresponding ones shown in the "Locked-Locked" and "Regular-Locked" respectively. This indicates that when the data transferred by *Alice* is in page-locked memory, the high data transfer latency measured by *Alice* represents the amount of contention on the PCIe bus. Yet, compared with the latency ranges in the "Locked-Locked", we notice that the corresponding ranges in the "Locked-Regular" have a larger variation. We summarize the investigation as follows:

> Contention on the host-GPU PCIe bus can lead to observable and consistent increases in the host-to-GPU data transfer time if the host data resides in page-locked memory.

4. Cross-VM Covert Channel Attack

In this section, we exploit contention on the host-GPU PCIe bus to construct a covert communication channel. This covert channel can be easily used to exfiltrate data

across boundaries established by VMs in many cloud computing situations.

4.1. Threat Model

We assume that there are two colluding parties, a sender and a receiver, on the same platform but in different security domains. The sender has access to some sensitive data, and it attempts to transmit this piece of data to the receiver through a covert channel. The platform is equipped with a modern GPU. The GPU can be accessed by both the sender and receiver, but its sharing between security domains is well protected, similar to other hardware resources.

In practice, such a threat model is very realistic in GPU cloud computing, where the sender and receiver are in different VMs and have access to a virtualized GPU. Although this model can be found in several other circumstances, we focus on the GPU cloud computing setting in this paper. In particular, we assume that the sender and receiver run on an IaaS cloud, which provides the tenant VMs with access to powerful GPUs enabled by the state-of-the-art NVIDIA GPU virtualization technique.[3] The concrete examples of such clouds include Amazon EC2 [2], Google Compute Engine [13], and Microsoft Azure [38].

4.2. Cross-VM Covert Channel over Host-GPU PCIe Bus

As shown in Section 3.2, contention on the host-GPU PCIe bus can be reliably detected and measured when the data involved in the transfers resides in page-locked memory. Therefore, we can exploit such contention to determine if the sender and receiver VMs collocate on the same platform and physically share the same GPU (namely the co-residency detection [20], [47], [50], [53], [56], [59]), and then to encode information to achieve covert communication between the colluding sender and receiver.

Since the process of measuring contention is also a process of generating contention to others, a straightforward co-residency detection protocol will be that both the sender and receiver continuously measure contention at the beginning; if the sender experiences constant contention for a predefined period of time, it knows it currently collocates with the receiver on the same platform and vice versa. After co-residency is detected, the colluding pair can start communication.

Instead of constructing a robust communication protocol similar to that presented in [37], we use a simple protocol to demonstrate the potential of exploiting contention on the host-GPU PCIe bus to create cross-VM covert channels in the GPU clouding computing settings. In this simple protocol, we use obvious contention to represent bit 1 and no contention to represent bit 0. The procedure of the receiver is shown in Figure 5. The receiver uses the CUDA API to allocate page-locked memory (line 2) for holding the data that is to be transferred over the PCIe bus to the allocated GPU memory. As summarized at the end of Section 3.2, to quantify the bus contention of interest, we measure the time needed for transferring the data residing in the page-locked memory to the GPU (lines 5 – 7). If the data transfer time is shorter than a threshold T, it is considered as contention-free, and a bit 0 is received; otherwise, a bit 1 is received (lines 8 – 12). The threshold T depends on the size B_R of the data transferred by the receiver, and the optimal size regarding B_R is actually always 32 KB which will be discussed later.

```
// N is the number of measurements to take
// T is the threshold for recognizing a bit 1
// B_R is the number of bytes transferred to the GPU by the
   receiver
1  Use an array n of N bits for the reception of message;
2  Allocate an array a_H of B_R bytes in page-locked memory of
     host;
3  Allocate an array a_D of B_R bytes in GPU memory;
4  for i ← 0 to N − 1 do
5  |   Use the RDTSCP to mark the start t_1;
6  |   Execute cudaMemcpy() to copy a_H to a_D;
7  |   Use the RDTSCP to mark the end t_2;
8  |   if t_2 − t_1 < T then
9  |   |   n[i] ← 0;
10 |   else
11 |   |   n[i] ← 1;
12 |   end
13 end
```

Figure 5: The procedure of the receiver.

```
// M is the number of bits to send
// K is the number of NOPs for sending a bit 0
// B_S is the number of bytes transferred to the GPU by the
   sender
1  Retrieve a message m of M bits to send;
2  Allocate an array a_H of B_S bytes in page-locked memory of
     host;
3  Allocate an array a_D of B_S bytes in GPU memory;
4  for i ← 0 to M − 1 do
5  |   if m[i] = 0 then
6  |   |   for j ← 0 to K − 1 do
7  |   |   |   Execute a NOP;
8  |   |   end
9  |   else
10 |   |   Execute cudaMemcpy() to copy a_H to a_D;
11 |   end
12 end
```

Figure 6: The procedure of the sender.

The procedure of the sender is shown in Figure 6. As illustrated in the "Locked-Locked" of Figure 4, stable contention will be created if the transferred data also resides in page-locked memory. Therefore, the sender also allocates a piece of page-locked memory (line 2) and transfers the data residing in it when contention is needed for sending a bit 1 (line 10). To send a bit 0, the sender does nothing but executes a loop of NOPs (lines 6 – 8). The number K of iterations depends on how long it takes for the receiver to transfer the data of B_R bytes without contention. Although the sender does not know this time beforehand, it can try to discover the time by itself, given

3. Notice that our covert channel also works perfectly with respect to non-virtualized GPUs. Virtualized GPU makes it much harder and more interesting as clouds use virtualization.

the fact that the sender and receiver run on the same platform and B_R is always a fixed value (namely 32 KB). Different from B_R, the optimal size B_S of data transferred by the sender is platform-dependent.

To determine the optimal size of the data transferred to the GPU by the sender and receiver, we have several criteria. For the receiver, which measures and uses its data transfer latency to discover the exfiltrated bits, the size of the data should make the data transfer latencies sensitive to contention on the PCIe bus, and it should also be small enough not to slow down the covert communication significantly. To this end, we use the ratio of averaged latency under contention to that under no contention as an indicator of sensitivity and derive such ratios with respect to different sizes ranging from 1 KB to 16 MB. For the purpose of comprehensiveness, we also vary the size of data transferred by the sender from 32 KB to 16 MB. The experiments are conducted against two NVIDIA virtualizable GPUs, Quadro RTX 6000 and Tesla V100, which are later used in our evaluations. The details of the GPU platforms are described in Table 1.

Note that the experiments are performed on the two GPUs in both the native setting and virtualized setting. Interestingly, no matter what data size is on the sender side and which setting is chosen, the ratio of our interest is most remarkable at 32 KB. Figure 7 shows the derived ratios when the sender transfers a piece of 16 MB data. With respect to other sender data sizes, the ratio patterns are exactly the same as the ones shown in Figure 7. Moreover, Figure 10 in Section 5 shows such ratios where several customer-grade GPUs are used. From that figure, we can also find that 32 KB is the optimal size meeting our criteria. Therefore, in the procedure of the receiver, we will always measure the latency of transferring 32 KB data in the host's page-locked memory to the GPU memory.

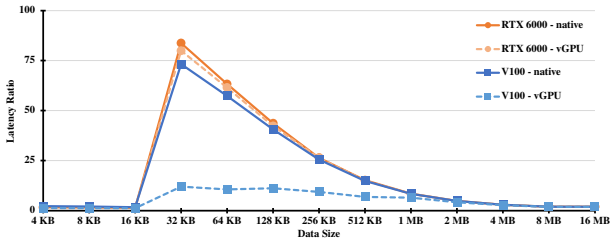


Figure 7: Ratio of averaged latency under contention to averaged latency under no contention, where the data size of the sender is fixed at 16 MB.

For the sender, we select the size that can make the 32 KB data transfer latency ranges under contention and no contention well separated. We find that the size boundary is GPU-specific. For example, in terms of Quadro RTX 6000, the boundary is 16 KB, but in terms of Tesla V100, it is 32 KB. When the size is above the boundary, the larger it is, the more separated the ranges will be, and the more resilient to noise the channel will be, but unfortunately, the less speed the channel can reach. In other words, there is a trade-off between the channel bandwidth and its noise resilience.

4.3. Evaluation

We evaluate this covert channel on a real-world large-scale cloud computing research platform, Chameleon Cloud [27]. We use two systems of Chameleon Cloud that are equipped with virtualizable GPUs and often used in data centers. The details of the two systems are listed in Table 1. In terms of hardware specifications, major cloud platforms like Amazon EC2 have little to no difference compared to our testbeds provided by Chameleon. To ensure the practicality of our attack, we evaluated LockedDown on VMs created under the-state-of-the-art GPU virtualization techniques. One common challenge for such attacks in the cloud is to achieve co-residency between the attacker and victim, but note that this problem has been widely studied in prior studies [50], [53], [56]. On system A, an NVIDIA Quadro RTX 6000 GPU is used, and we virtualize it with NVIDIA's "GRID RTX6000-4Q" configuration that creates up to 6 vGPUs, each of which has 4 GB GPU memory. On system B, an NVIDIA Tesla V100 GPU is used, and we virtualize it with NVIDIA's "GRID V100DX-4Q" configuration that creates up to 8 vGPUs, each of which has 4 GB GPU memory. Such configurations simulate the commonly used GPU cloud computing platforms.

The sender and receiver run on two VMs, and each VM is assigned a vGPU. We skip the process of finding co-residency as they are already on the same platform physically sharing the same GPU, but a simple simulation shows that host-to-GPU data transfer latencies can be used for co-residency detection.

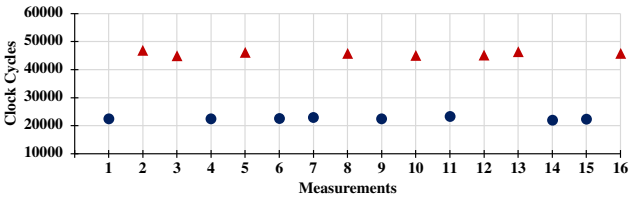


Figure 8: 16 measured data transfer latencies corresponding to bits "0110100101011001" on system A.

As mentioned above, B_R is always 32 KB, but B_S changes with the GPU. On system A, we set B_S to 128 KB, K to 5250, and T to 40,000. For example, Figure 8 shows 16 data transfer latencies measured by the receiver under this setting. From the figure, we can clearly distinguish bits 1 and 0. On system B, we set B_S to 512 KB, K to 19250, and T to 100,000.

For evaluation, we choose nine paragraphs from a piece of randomly generated Lorem Ipsum text as the message sent over the covert channel. The message consists of 5000 bytes, i.e., 40,000 bits. We evaluate how fast this message can be sent over our contention-based covert channel, and we also calculate the error rate with respect to the Levenshtein edit distance[4] between the sent message and the received bits. For the error rate evaluation, we send the message 6 times, and derive their mean and the standard deviation. Table 2 shows the results.

4. The Levenshtein edit distance is the minimum number of edits required to change one string into another string. An edit is either an insertion, a deletion, or a substitution.

TABLE 1: Platforms of Chameleon Cloud on which the covert channel is evaluated.

System	Platform	CPU	Memory	OS	GPU	# vGPUs
A	Dell PowerEdge R740	2 × Xeon Gold 6126	12 × 16 GB DDR4-2666	CentOS 8.3	Quadro RTX 6000	6
B	Dell PowerEdge C4140	2 × Xeon Gold 6230	8 × 16 GB DDR4-2933	CentOS 8.3	Tesla V100	8

TABLE 2: Bandwidth and error rate of the covert channel in a controlled environment.

System	Bandwidth	Error Rate μ (σ)
A	64 kbps	0.0088 (0.0043)
B	20 kbps	0.0029 (0.0005)

From the results, we can observe that this covert channel has a competitive bandwidth with a very low error rate. Note that the errors are not due to other devices competing for the host-GPU PCIe bus's bandwidth (because in reality GPUs are installed on the primary PCIe slots connected to CPUs via dedicated PCIe lanes). In general, the errors that appear in this evaluation are caused by synchronization issues between the sender and receiver. Essentially, the parameter K in Figure 6 cannot be precisely determined, so insertions of 0's can occur if K is chosen too big, or insertions of 1's can occur if it is chosen too small. Nevertheless, such errors can be easily removed by using advanced encoding methods.

In [39], Naghibijouybari *et al.* propose several GPU covert channels, but they consider only a native execution environment without any virtualization. We have tried to adapt their covert channels for the more realistic virtualized GPU circumstances. However, we did not succeed in making any of their covert channels work across vGPUs. We find that only their L2 cache covert channel can work with the sender and receiver being in the same VM and using the same vGPU. The highest bandwidth it can reach is 30 kbps on system A and 11 kbps on system B, each of which is only about a half of the corresponding one of ours. Whether their covert channels can work under the threat model described in Section 4.1 needs further investigation, even though our initial strenuous attempts gave a negative answer[5].

The evaluation above is merely performed in a controlled environment where the sender and receiver are the only two processes intensively running on the platform. However, in reality, especially in cloud computing, there may be many other processes running in other VMs in parallel with the sender and receiver. To create a practical situation where GPU cloud computing is specifically needed, we have three other VMs, and inside each VM, a CNN model is trained using the attached vGPU. The CNN model is an example in the TensorFlow framework [52], and its training involves 50,000 images from the CIFAR10 dataset.

In such a practical scenario, we evaluate the covert channel again using the message consisting of 40,000 bits. For comparison, we also evaluate the L2 cache covert channel proposed in [39] within one VM on one vGPU (as mentioned above, it cannot work across vGPUs). The results are shown in Table 3.

Compared to the results listed in Table 2, we can observe that both the bandwidth and error rate are negatively affected (e.g., in terms of system A, the bandwidth is reduced by 4 kbps, and the error rate is increased by about 3%). However, they are still acceptable, especially considering the large amount of background noise created by other VMs. By contrast, the L2 cache covert channel proposed in [39] is significantly degraded (e.g., in terms of system A, its bandwidth is reduced by 28 kbps, and the error rate becomes too high so that the covert channel is no longer very useful).

5. If the newest MIG virtualization (described in Section 2.3) is used, their covert channels will certainly not work.

TABLE 3: Bandwidth and error rate of the covert channel in a realistic environment.

System	Covert Channel	Bandwidth	Error Rate μ (σ)
A	Host-GPU PCIe bus	60 kbps	0.0391 (0.004)
	L2 cache [39]	2 kbps	0.2853 (0.005)
B	Host-GPU PCIe bus	18 kbps	0.0413 (0.003)
	L2 cache [39]	1 kbps	0.2951 (0.015)

In addition, we need to mention that the parameters of our covert channel stay the same as the ones used in the controlled scenario. However, the parameters of the L2 cache covert channel of [39] need to be well-tuned; otherwise, the channel will behave similar to randomly guessing bits. Therefore, the adaptiveness and resilience of our covert channel is also much better than the one proposed in [39].

Next, we further explore the design space of the covert channel protocol by explicitly adding synchronization. In [39], Naghibijouybari *et al.* show that their cache covert channel's bandwidth can reach as high as 75 kbps by introducing full synchronization. Here, we choose to synchronize our covert channel communication using a 16-bit header and a 16-bit ending. Additionally, Naghibijouybari *et al.* demonstrate that by tolerating an error rate of 18 to 26%, their L2 cache covert channel bandwidth can increase by 25%. It is worth noting that their high error rate was observed as a result of decreasing the number of iterations without introducing additional noise. We tweaked the parameters of the procedure shown in Figure 6 on system A to increase our bandwidth by tolerating a higher error rate. To this end, we set B_S to 32 KB, K to 5500, and T to 23,000. As shown by Scenario 1 in Table 4, we can achieve a higher speed up to 90 kbps with a slightly increased mean error rate and standard deviation.

TABLE 4: Bandwidth and error rate of the covert channel in different scenarios with synchronization.

Scenario	Bandwidth	Error Rate μ (σ)
1	90 kbps	0.0140 (0.005)
2	81 kbps	0.0038 (0.006)
3	88 kbps	0.1569 (0.076)

We can further optimize our communication protocol by dividing a message into fixed-size chunks and sending them individually. Since the synchronization we use needs 32 bits per package, if the chunk size is too small, the overhead ratio will be too big. We empirically find that the

package size of 1024 bits, including the 32-bit synchronization overhead is optimum. By splitting our message into 992-bit chunks and adding synchronization, we are able to reduce our mean error rate to 0.0038 at 81 kbps, as shown by Scenario 2 in Table 4.

The implementations we discussed so far evaluate the performance and resilience of our covert channel under realistic workloads, with three other VMs, using different vGPUs, and training neural network models. Table 3 shows that our approach has a low error rate and a high bandwidth under representative noise levels. An extreme level of noise can have a considerable impact on the accuracy, which is common for all contention-based side/-covert channels. To find how resilient our covert channel is under scenarios where a large amount of traffic occurs on PCIe, we decide to use a well-known implementation of the DGEMM GPU micro-benchmark [42] which measures performance for matrix-matrix multiplication. In the DGEMM, the CPU loads and stores each data block to a pinned memory buffer optimized with blocking and parallelization techniques. In [34], Matsumoto *et al.* have shown that this benchmark can consume up to 80% of the PCIe bandwidth. We anticipate this benchmark to generate excessive noise, since it is an extreme case for a High-Performance Computing Challenge Benchmark [33] to be used in a virtualized environment. As shown by Scenario 3 in Table 4, we can achieve 88 kbps, and we calculate the mean and the standard deviation to be 0.1569 and 0.076, respectively. Compared to the L2 cache covert channel proposed in [39] where no noise is actually introduced, our covert channel can achieve a significantly higher bandwidth with a significantly lower error rate even in such an extreme scenario.

5. Website Fingerprinting Attack

As online presence plays a very prominent role in today's society, one way to extract intelligence about an individual's behavior is to monitor his/her web browsing activities. The collection of such information leads to a massive breach of privacy because direct or indirect interpretations can be made about someone's economic, political and social affiliations. Website fingerprinting attacks have emerged to collect web browsing information [16], [22], [43], [48], [49], [54]. In this section, we demonstrate that the measurable contention caused on the host-GPU PCIe bus can be exploited to build a website fingerprinting attack. We show that each website exhibits a unique pattern of PCIe bus contention that can be used as its fingerprint.

5.1. Threat Model

We assume that there is an attacker who wants to stealthily learn information about which websites have been visited by a victim. The victim uses a personal computer (e.g., a desktop or a laptop) to browse websites, and the computer is assumed to have a CUDA-enabled NVIDIA GPU. Considering the dominant market share of NVIDIA GPUs (e.g., 82% in the Q4 2020 [1]), this assumption is regarded as reasonable. We do not impose strong assumptions on the OS being used by the victim, as long as it is supported by CUDA (which is true for widely used OSes like Windows and Linux) and the CUDA runtime is installed. We do not have strong assumptions on the web browser either, as long as it uses the GPU to help render websites (which is true for broadly used browsers like Chrome and Firefox).

We assume that the attacker has placed a piece of malware on the victim's computer. How this malware is placed on the victim's computer is out of scope, but as mentioned by the prior works [16], [22], [29], [64], this can be achieved through methods like phishing or physical accessing. This malware does not require any privilege higher than the normal user, and it does not require the existence of any software vulnerabilities in the OS or browser.

We assume the victim will visit only the popular websites on some list (e.g., Alexa top sites), and the attacker knows this list. In other words, we focus on a closed-world scenario, in which the attacker strives to pinpoint the websites browsed by the victim from a set of possibilities.

5.2. Website Fingerprinting based on PCIe Bus Contention

Modern web browsers use GPUs not only for displaying but also for helping render web pages. Many objects during rendering a web page are sent to the GPU [29], [40]. According to the study conducted in [40], rendering different web pages gives rise to distinct GPU memory utilization traces.

We anticipate that when visiting different websites, the patterns of the induced traffic on the host-GPU PCIe bus should also be different. If such patterns can be captured, we should be able to use them to identify which websites are being browsed (i.e., website fingerprinting). Because a different amount of traffic creates a different amount of bus congestion, to capture the traffic patterns of interest, we can leverage the contention measurement approach of the receiver discussed in Section 4, where the `cudaMemcpy()` function is repeatedly used to copy a piece of data to the GPU, and the data transfer latency is measured. To this end, we adapt the approach of the receiver to have the procedure shown in Figure 9.

```
// S is the number of samples to take
// R is the number of repeats to adjust time granularity
// B_W is the number of bytes transferred to the GPU
Use an array s of length S for storing samples;
Allocate an array a_H of B_W bytes in page-locked memory of host;
Allocate an array a_D of B_W bytes in GPU memory;
for i ← 0 to S − 1 do
    Use the RDTSCP to mark the start t_1;
    for j ← 0 to R − 1 do
        Execute cudaMemcpy() to copy a_H to a_D;
    end
    Use the RDTSCP to mark the end t_2;
    s[i] ← t_2 − t_1;
end
```

Figure 9: The procedure of capturing the pattern of PCIe bus traffic induced by rendering a web page.

As described in Section 3, to reliably utilize the host-to-GPU data transfer time to measure contention on the PCIe bus, the data to be transferred needs to reside in page-locked memory. Therefore, we still exploit the page-locked memory allocation in CUDA (line 2). Again, we select how much data should be transferred each time with respect to the sensitivity to contention in the form of latency ratios. Similar to Figure 7, Figure 10 shows the ratios when three popular GPUs that are commonly used in personal computers are tested. (The three GPUs are NVIDIA GeForce GTX 1080, RTX 2060, and RTX 2080.) From the figure, we can clearly find that B_W should also be 32 KB.

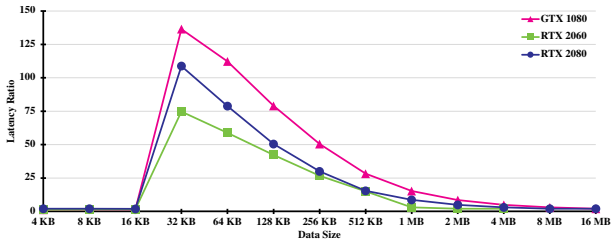


Figure 10: Ratio of averaged latency under contention to averaged latency under no contention.

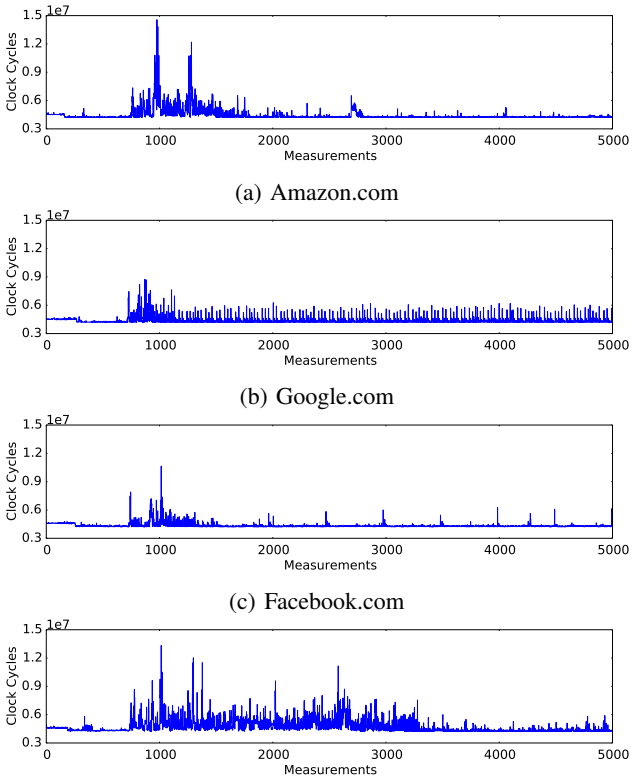
Since transferring 32 KB data over the PCIe bus to the GPU usually takes several microseconds, measuring and recording the execution time of each `cudaMemcpy()` invocation will result in an unnecessarily long sequence, given the fact that rendering a web page may need several seconds. Accordingly, we measure R invocations together as one sample in Figure 9 (lines 6 – 8). The selection of R is a trade-off between the time granularity of a sample and the total number of samples. We empirically choose an R that makes each sample take about 1.5 ms under no contention.

To verify our anticipation and also that the captured traffic patterns can be used to fingerprint websites, the approach is tested against four commonly visited websites using Chrome on a computer equipped with an NVIDIA RTX 2080. Figure 11 shows the traces taken for several websites using the procedure in Figure 9. The numbers on the X-axis correspond to the samples, and each sample corresponds to one iteration in Figure 9. The Y-axis shows the time taken for one `cudaMemcpy()` in clock cycles (using `RDTSCP`). From the figure, we can see that each website gives rise to a distinctive pattern, which can be interpreted as its fingerprint.

More importantly, we find that the patterns corresponding to multiple visits of a given website do not have significant variations, which means that the bus contention measurement is very stable. As illustrated in Section 3.2, such a stable measurement happens in the "Locked-Locked" scenario. Therefore, we speculate that a web browser has a DMA buffer dedicated to the communication between the GPU and the browser for hardware-accelerated web page rendering.

5.3. Evaluation

We evaluate the PCIe bus contention-based website fingerprinting against different settings of GPU, OS, and



(a) Amazon.com

(b) Google.com

(c) Facebook.com

(d) Nytimes.com

Figure 11: Data transfer latency traces for visiting four websites.

browser. The GPUs on which the evaluation is performed include NVIDIA GeForce GTX 1080, RTX 2060, and RTX 2080. These are very popular desktop GPUs currently being used in practice. The evaluated OSes are Windows 10 and Ubuntu 20.04, which are both very widely used on desktops. On both Windows and Ubuntu Linux systems, we install Chrome and Firefox and use them for the evaluation. (Both Chrome and Firefox leverage the GPU to help render web pages by default.) In addition to the evaluation in the normal network scenario against commonly used browsers, we further evaluate this PCIe bus contention-based website fingerprinting attack under the anonymity network scenario. Specifically, we focus on Tor (The Onion Router) that encrypts network traffic and relays it between users and website servers anonymously. The evaluation settings are listed in Table 5.

Based on the Alexa Top websites globally, we have created a list of 100 websites. These 100 websites are listed in the appendix. As stated above, we evaluate the website fingerprinting in a closed-world scenario, where we classify a given trace into one of the 100 websites. Since the captured traces are essentially time series data, we simply use a Fully Convolutional Network (FCN) model in Keras that is used for time series classification [10]. Figure 12 illustrates the architecture of the model. The hyperparameters and parameters of this model can be found in [10], [21].

Under each setting, we train an FCN model where the training dataset consists of 100 captured data transfer latency traces for each website, namely, 10,000 traces in total. We also collect another 150 traces per website for

TABLE 5: Website fingerprinting evaluation settings.

Setting	OS	Web Browser	GPU
W-C-1080	Windows 10	Chrome	GTX 1080
W-C-2060	Windows 10	Chrome	RTX 2060
W-C-2080	Windows 10	Chrome	RTX 2080
W-F-1080	Windows 10	Firefox	GTX 1080
W-F-2060	Windows 10	Firefox	RTX 2060
W-F-2080	Windows 10	Firefox	RTX 2080
W-T-1080	Windows 10	Tor	GTX 1080
W-T-2060	Windows 10	Tor	RTX 2060
W-T-2080	Windows 10	Tor	RTX 2080
U-C-1080	Ubuntu 20.04	Chrome	GTX 1080
U-C-2060	Ubuntu 20.04	Chrome	RTX 2060
U-C-2080	Ubuntu 20.04	Chrome	RTX 2080
U-F-1080	Ubuntu 20.04	Firefox	GTX 1080
U-F-2060	Ubuntu 20.04	Firefox	RTX 2060
U-F-2080	Ubuntu 20.04	Firefox	RTX 2080

the purpose of testing, namely, 15,000 traces in the test dataset. Each captured trace contains 5000 samples, which lasts between 7 to 9 seconds depending on how much PCIe bus traffic is created during rendering the corresponding web page. The trace collections are mainly conducted in two locations. One location is the campus of a university, and the other one is an apartment. The traces in different settings are collected in different weeks.

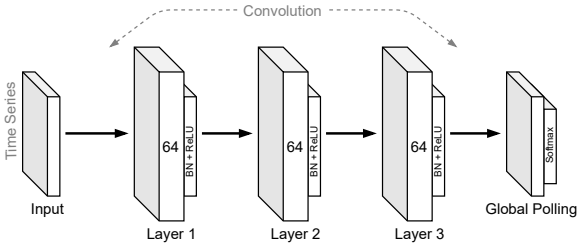


Figure 12: The architecture of the FCN model.

5.3.1. Normal Network Scenario. We first evaluate the attack against Chrome and Firefox under normal network circumstances. The website fingerprinting accuracy results are presented in Table 6. The main diagonal entries in the table give the results when the setting under which the testing traces are collected matches the setting under which the training examples are collected. From the results, we can see that the website fingerprinting accuracy is very high no matter which setting is chosen. For example, the accuracy can reach 95.2% in the W-C-2080 setting, and even the lowest accuracy that happens under the U-F-2080 setting can still be 84.4%.

We examine what websites are often misclassified. We find that there are mainly three types of characteristics shared by these websites: (1) They often have similar layouts and outlooks, e.g., Dailymail and Foxnews. (2) They often have a typical navigation bar with wide empty margins on the body, e.g., Tripadvisor and Expedia. (3) They often have many dynamic contents with occasionally autoplayed video pop-ups on the side, e.g., ESPN and Rottentomatoes. Note that some websites may share all these characteristics, but they are still well distinguished. Therefore, these characteristics do not form a sufficient condition for misidentification.

Moreover, the entries that are not on the main diagonal show the respective model tested against traces collected under a different setting. While most of the results show that cross-setting prediction is similar to random guessing, some entries that are highlighted in the table interestingly show the opposite. For example, the model trained under the U-F-2080 setting can classify the traces captured in the U-F-2060 setting with a 52.6% accuracy, and likewise, the model trained under the U-F-2060 setting can classify the traces captured in the U-F-2080 setting with a 50.6% accuracy. (Collecting the two testing datasets is separated by weeks.) Initially, we thought this is due to that both RTX 2060 and 2080 belong to the same architecture. However, the other pairs (e.g., W-C-2060 and W-C-2080) do not show similar results. We conclude that exploiting contention on the host-GPU PCIe bus to fingerprint websites is browser and platform-sensitive in general.

Table 7 additionally shows the average along with the minimum precision and recall values for website fingerprinting under each setting. With respect to a website, we have

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{and} \quad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where TP refers to true positives (i.e., a trace of a website is correctly classified as that website), FP refers to false positives (i.e., a trace that does not belong to a website is incorrectly classified as that website), and FN refers to false negatives (i.e., a trace of a website is incorrectly classified as not belonging to that website). Precision shows how much we can trust that the identified visits to a website are indeed made by a user, and recall indicates how much portion of the visits to a website made by a user can be correctly identified using this website fingerprinting approach. From the results, we can observe that the precision and recall values are sufficiently high even in the worst cases, which means that this website fingerprinting approach is reliable and accurate.

5.3.2. Anonymity Network Scenario. We also evaluate the attack against Tor in the anonymity network circumstances. However, when using Tor browser, we find that some of the 100 popular websites which we use for evaluation (listed in the appendix) cannot be reached through the Tor network (i.e., access denied, 403 forbidden, etc.). Consequently, we have to remove these websites from our list in this evaluation against Tor, and in the end there are 64 websites left.

Compared with Chrome and Firefox, Tor introduces many difficulties. First, due to the unpredictable latency incurred by Tor routers and its overlay network, the captured traces may be distorted in time. Second, due to the encryption and decryption operations used in each packet, the traces are noisier. Third, due to the changeable exit node, websites adapting to geolocations may use different languages or provide different contents, which can affect the consistency of the captured traces.

Nevertheless, even in the presence of the intricacies introduced by Tor, our website fingerprinting attack still performs well. The website fingerprinting accuracy results for Tor browser on Windows are presented Table 8. From the results, we can see that up to 90.6% accuracy can be achieved in the W-T-2060 setting. Compared to the result in the W-T-2060 setting, the accuracy in the W-T-1080 or the W-T-2080 setting is only slightly lower (which is

TABLE 6: Accuracy for testing the models against traces from the same and different platforms.

		Testing cases											
		W-C-1080	W-C-2060	W-C-2080	W-F-1080	W-F-2060	W-F-2080	U-C-1080	U-C-2060	U-C-2080	U-F-1080	U-F-2060	U-F-2080
Classifier	W-C-1080	91.8%	36.2%	27.6%	4.7%	4.1%	3.5%	1.5%	2.8%	2.0%	0.6%	1.0%	1.1%
	W-C-2060	18.6%	91.5%	4.1%	4.1%	4.5%	5.4%	1.2%	4.3%	2.0%	1.1%	1.0%	1.2%
	W-C-2080	3.4%	7.0%	95.2%	2.4%	4.2%	3.7%	1.8%	1.7%	1.8%	0.8%	1.6%	1.0%
	W-F-1080	3.4%	3.1%	1.5%	90.7%	7.1%	6.7%	1.2%	2.2%	1.2%	1.4%	1.1%	1.2%
	W-F-2060	3.5%	4.1%	0.9%	8.6%	93.7%	41.5%	2.0%	1.8%	2.7%	1.4%	0.7%	1.2%
	W-F-2080	3.1%	6.6%	2.0%	12.0%	30.5%	93.3%	2.5%	2.4%	2.8%	1.5%	1.0%	1.2%
	U-C-1080	2.1%	3.2%	1.7%	3.8%	2.9%	3.4%	91.0%	16.9%	33.7%	1.2%	0.8%	1.4%
	U-C-2060	3.0%	3.6%	2.0%	2.0%	2.5%	2.2%	14.4%	89.0%	11.3%	0.9%	1.1%	1.1%
	U-C-2080	2.1%	1.7%	2.2%	2.6%	1.6%	1.8%	28.7%	7.1%	93.8%	1.6%	1.1%	1.5%
	U-F-1080	0.7%	0.9%	0.7%	1.5%	0.8%	0.8%	1.3%	0.4%	2.5%	85.4%	0.4%	0.5%
	U-F-2060	2.5%	1.6%	0.9%	1.6%	0.6%	0.2%	2.6%	3.1%	2.8%	11.4%	88.5%	50.6%
	U-F-2080	0.2%	1.1%	0.8%	1.6%	0.2%	0.2%	1.8%	2.0%	1.7%	8.2%	52.6%	84.4%

TABLE 7: The average and minimum precision and recall for evaluation against Google Chrome and Firefox browsers on Windows and Ubuntu Linux.

	Precision		Recall	
GPU	Mean	Min.	Mean	Min.
W-C-1080	92.8%	47.2%	91.8%	44.0%
W-C-2060	92.5%	52.8%	91.5%	44.0%
W-C-2080	95.5%	73.4%	95.2%	60.0%
W-F-1080	92.0%	56.7%	90.7%	56.0%
W-F-2060	94.0%	78.0%	93.7%	54.7%
W-F-2080	93.6%	66.9%	93.3%	60.0%
U-C-1080	91.9%	59.0%	91.0%	43.3%
U-C-2060	90.1%	46.4%	89.0%	60.7%
U-C-2080	94.2%	73.6%	93.8%	72.7%
U-F-1080	86.0%	45.9%	85.4%	42.7%
U-F-2060	89.1%	55.5%	88.5%	38.7%
U-F-2080	84.9%	50.0%	84.4%	46.0%

89.9%). The cross-setting prediction results also show that some of the different classifier and testing pairs (W-T-1080 and W-T-2060; W-T-2060 and W-T-2080) can achieve an accuracy higher than 21%.

TABLE 8: Accuracy for testing the models against traces from the same and different platforms on Tor.

		Testing cases		
		W-T-1080	W-T-2060	W-T-2080
Classifier	W-T-1080	89.9%	21.2%	9.3%
	W-T-2060	12.8%	90.6%	21.5%
	W-T-2080	7.8%	19.4%	89.9%

In addition, Table 9 shows the average and minimum precision and recall values for the evaluation against Tor. Compared to the corresponding values shown in Table 7 for the evaluations against Chrome and Firefox, we do not observe significant differences. This means that this website fingerprinting approach is still reliable and accurate even if Tor browser is used.

TABLE 9: The average and minimum precision and recall for evaluation against Tor browser on Windows.

	Precision		Recall	
GPU	Mean	Min.	Mean	Min.
W-T-1080	90.2%	58.5%	89.9%	52.7%
W-T-2060	90.9%	57.0%	90.6%	54.0%
W-T-2080	90.9%	45.1%	89.9%	56.0%

6. Countermeasures

The most naive approach to thwarting our attacks is to remove the page-locked memory allocation and transfer feature from CUDA (e.g., the `cudaMallocHost()` and `cudaHostAlloc()` functions). This would significantly hinder our control over using data transfer latencies to measure contention on the bus. After this change, any attempt to rebuild the attacks presented in this paper would have to resort to using regular pageable data transfers. However, data transfers from regular pageable memory can introduce a considerable amount of overhead and display irregularities in latency as shown in Section 3.2. Therefore, this countermeasure collaterally introduces many downsides for many parallel computing programs that require large amounts of asynchronous data transfers. For highly parallel applications optimizing data transfers on CUDA is already a challenge on its own. Currently, NVIDIA's own CUDA programming documentation recommends three optimization methods: minimizing the data transfers, batching small transfers, or using page-locked host memory. Page-locked data transfers are essential when a programmer needs to transfer a small amount of data for an asynchronous process, since page-locked memory can be handled asynchronously without introducing a colossal data transfer overhead, and thus keeping the SMs from stalling.

To specifically mitigate our covert channel from being a danger for cloud users using vGPUs, NVIDIA can implement a time-division multiple access (TDMA) method that divides hardware resource usage into time-sharing slices. If the PCIe is shared via TDMA across multiple vGPUs, then the intermission between succeeding data transfers would not depend on the size of data transfers. Therefore, it would eliminate our methodology from being a threat across vGPUs. In such a scenario, the delays from a VM's perspective would not be meaningful to build such a covert channel. However, this method would not be the most optimal approach in terms of optimizing the efficiency of the data transfer pipeline. This could hinder data transfers across all VMs that utilize a vGPU instance of the same physical GPU and this scheduling would introduce additional overhead. Similarly, TDMA can also be applied in a way that can eliminate our side-channels use in the website fingerprinting attack. For instance, each process can be assigned to a security domain, which would then be assigned to distinct channels. Therefore, each security domain would have its fair share of the PCIe in complete isolation. Nonetheless, this approach could also cause system-wide stagnation from the perspective of the

data transfer pipeline.

Additionally, a mechanism that detects our attacks can be employed as a countermeasure. Our side-channel relies on repeated transfer of data to detect contention. These transfers themselves cause a noticeable contention that might be detected by a third-party that occasionally measures the transfer bandwidth. Moreover, our attacks increase the power draw of the GPU noticeably by constantly utilizing DMA engines. This is particularly noticeable in a consumer PC since the cooling system audibly ramps up during our fingerprinting process. As a result, anomaly detection mechanisms that rely on monitoring GPU performance metrics may be employed to uncover our attacks and warn the user.

7. Related Work

There has been plenty of interest in security vulnerabilities caused by shared hardware resources. Wu *et al.* [56] brought attention to the possibility of implementing high-speed covert channel attacks in the cloud and showed that virtualized environments still offer covert channel mediums. Lately, researchers have also started to study GPU vulnerabilities.

Frigo *et al.* [11] studied GPU-based microarchitectural attacks on integrated GPUs of ARM platforms. In contrast, we target discrete GPUs and cloud systems. Naghibijouybari *et al.*, [39] presented the first covert channel attacks on GPGPUs and exploited the scheduling algorithms to force colocation. They showed cache-based covert channels between two colluding kernels running on the same GPU on the cloud. We failed to fully replicate their work across separate vGPUs and evaluate its practicality under our threat model.

Analogous to our work, Tan *et al.* exploited PCIe congestion to mount side-channel attacks that use RDMA NIC to attack GPU and NVMe SSD to attack Ethernet NIC [51]. Our approaches to contention generation differ since their attacks utilize NICs to introduce PCIe congestion, while our attack uses GPUs. Website fingerprinting attack is a scenario discussed in both of our works. However, we have evaluated our work on more GPUs under two operating systems and with various web browsers, whereas the work in [51] only considers a single GPU using Chrome under Linux. For webpage inference based on PCIe contention, we use host-GPU contention in contrast to their NIC-GPU contention. They have investigated stealing neural networks and keystrokes while we introduce the first virtualized GPU-based cross-VM covert channel. Our work differs by providing a covert communication channel for data exfiltration across isolation boundaries established by state-of-the-art virtualization techniques. In addition to these conjectural differences our attacks also differ in practicality. Although Tan *et al.* proposes the RDMA scenario as not needing execution capabilities, in cloud computing setups, GPUs are nearly always installed on the PCIe slot that is connected to the CPU via dedicated lanes, as illustrated in Figure 1. As described in Section 2.2, the first few PCIe slots on a motherboard are for dedicated lanes, which are sometimes called primary PCIe, and the rest of the slots are connected to a PCIe switch in the PCH. In such standard scenarios, no other PCIe devices (e.g., NVMe or RDMA NIC) will compete with the GPU for PCIe bandwidth, regardless of whether they are directly via dedicated lanes or indirectly via PCH. We have evaluated our work in a more realistic setup (i.e., Chameleon Cloud [27]) where the GPU uses the dedicated lanes to the CPU and the system's hardware is not intentionally arranged.

7.1. GPU Covert-/Side-Channel Attacks

Over the recent years, computer systems have been increasingly facing challenges introduced by covert channel attacks [37], [56], [59], [61], [63], [64]. However, to the best of our knowledge, fully virtualized GPUs have not been previously targeted by covert channel attacks.

Naghibijouybari *et al.* reverse engineered hardware scheduling algorithms and showed that GPUs are vulnerable to covert channel attacks between two concurrently running kernels [39]. They introduced several types of covert channels, but their evaluations were conducted in a controlled manner on one bare-metal GPU running two concurrent kernels. To the best of our exhaustive attempts, we cannot achieve their covert channels across two vGPUs, as described in Section 4.3. Moreover, in [7], Davidov and Oldenburg exploited the EM emanations of a malware-infected AMD GPU to construct a physical covert channel.

Numerous works have shown the threat of timing side-channel attacks on CPUs [4], [14], [15], [44]. With the advancements of GPUs as crucial components of contemporary computing devices, researchers show great interest in finding the GPU side-channel vulnerabilities. The works in [6] and [58] took cache-timing attacks to the GPU environments.

Jiang *et al.* conducted a timing attack on a CUDA AES implementation where the encryption time is utilized to recover the private key [24]. Later, they have also conducted another work where they showed that bank conflicts in a GPU's shared memory could be used to create a timing channel [25]. Frigo *et al.* investigated the security implications of integrated GPUs on mobile devices, and they showed that the WebGL timing APIs could be leveraged to build side-channel and Rowhammer attacks using JavaScript [11].

7.2. Website Fingerprinting Attacks

One group of website fingerprinting attacks relies on network traffic analysis. Packets on a network cause distinct traffic patterns due to timing variations and varying sizes. Researchers have shown that these attacks can be implemented on many web browsers, operating systems, or encryption scenarios [3], [17], [18], [23], [26], [30], [32], [45]. Recently, methods such as traffic splitting [8] and obfuscation with dummy packets [12] were introduced for mitigating these types of website fingerprinting attacks.

Another group of website fingerprinting attacks leverages side-channel information. As described in [64], the side-channel information can be either physical or logical. The physical website fingerprinting attacks exploit some observable physical side effects correlated with rendering different web page [5], [31], [35], [46], [60], [62]. In terms of logical fingerprinting attacks, it has been shown that cache-based side-channel information [43], [48], hardware

performance counters [16], memory footprints [22], GPU memory dumps [29], GPU utilization patterns [40], and mobile phone statistics [28], [49] can be exploited for this purpose. Similarly, our PCIe bus contention-based website fingerprinting attack also relies on a logical side-channel.

8. Conclusion

In this paper, we disclose a novel side-channel vulnerability on systems equipped with GPUs. Side-channels caused by contention on the PCIe bus are overlooked by manufacturers. Motivated by the observation that heterogeneous parallel computing models on GPUs require immense amounts of data to transfer, we constructed two realistic attacks exploiting the contention on the host-GPU PCIe bus. In the first attack, we have built a covert communication channel that can exfiltrate information across virtual GPUs assigned to different VMs. This high-speed covert channel attack raises questions about the security of cloud computing systems equipped with GPUs. In the second attack, we have implemented a website fingerprinting attack using the unique pattern of PCIe bus contention to infer the websites visited by a victim. We evaluated this website fingerprinting attack against popular browsers like Chrome and Firefox on both Windows and Linux as well as against Tor, and showed that it is reliable and accurate. In addition, we have conferred various countermeasures to thwart these attacks.

Availability

The artifacts needed to reproduce this work are available at https://github.com/mertside/lockeddown.

Acknowledgment

This work is supported in part by the National Science Foundation (CNS-2147217, CNS-1739328, and SaTC-2019536). The authors would like to thank the anonymous reviewers for their comments and suggestions that help us improve the quality of the paper. The authors would also like to thank Chameleon for providing experimental cloud platforms for our research.

References

[1] T. Alsop, "PC discrete GPU market share worldwide by vendor 2020," Mar 2021. [Online]. Available: https://www.statista.com/statistics/1131242/pc-discrete-gpu-shipment-share-by-vendor-worldwide/

[2] Amazon, "Install NVIDIA drivers on Linux instances," https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/install-nvidia-driver.html#nvidia-GRID-driver, 2021.

[3] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 605–616.

[4] M. H. I. Chowdhuryy and F. Yao, "Leaking secrets through modern branch predictor in the speculative world," *IEEE Transactions on Computers*, 2021.

[5] S. S. Clark, H. Mustafa, B. Ransford, J. Sorber, K. Fu, and W. Xu, "Current events: Identifying webpages by tapping the electrical outlet," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 700–717.

[6] B. Cope, P. Y. Cheung, W. Luk, and L. Howes, "Performance comparison of graphics processors to reconfigurable logic: A case study," *IEEE Transactions on computers*, vol. 59, no. 4, pp. 433–448, 2010.

[7] Davidov, Mikhail and Oldenburg, Baron, "TEMPEST@Home - Finding Radio Frequency Side Channels," https://duo.com/labs/research/finding-radio-sidechannels#section9, 2020.

[8] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko, "TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1971–1985.

[9] M. Dowty and J. Sugerman, "GPU Virtualization on VMware's Hosted I/O Architecture," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, p. 73–82, Jul. 2009. [Online]. Available: https://doi.org/10.1145/1618525.1618534

[10] H. I. FAWAZ, "Keras documentation: Timeseries classification from scratch," 2020. [Online]. Available: https://keras.io/examples/timeseries/timeseries_classification_from_scratch

[11] P. Frigo, C. Giuffrida, H. Bos, and K. Razavi, "Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 195–210.

[12] J. Gong and T. Wang, "Zero-delay lightweight defenses against website fingerprinting," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 717–734.

[13] Google, "Installing GRID drivers for virtual workstations," https://cloud.google.com/compute/docs/gpus/install-grid-drivers, 2021.

[14] 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 18)*, 2018, pp. 955–972.

[15] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games–bringing access-based cache attacks on aes to practice," in *2011 IEEE Symposium on Security and Privacy*, 2011, pp. 490–505.

[16] B. Gulmezoglu, A. Zankl, T. Eisenbarth, and B. Sunar, "PerfWeb: How to violate web privacy with hardware performance events," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 80–97.

[17] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 1187–1203.

[18] A. Hintz, "Fingerprinting websites using traffic analysis," in *International workshop on privacy enhancing technologies*. Springer, 2002, pp. 171–178.

[19] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "Deepsniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 385–399.

[20] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2016, pp. 368–388.

[21] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[22] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 143–157.

[23] R. Jansen, M. Juarez, R. Galvez, T. Elahi, and C. Diaz, "Inside job: Applying traffic analysis to measure tor from within." in *NDSS*, 2018.

[24] 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)*. IEEE, 2016, pp. 394–405.

[25] ——, "A novel side-channel timing attack on GPUs," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 167–172.

[26] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 263–274.

[27] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, "Lessons Learned from the Chameleon Testbed," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[28] H. Kim, S. Lee, and J. Kim, "Inferring browser activity and status through remote monitoring of storage usage," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, pp. 410–421.

[29] 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*. IEEE, 2014, pp. 19–33.

[30] S. Li, H. Guo, and N. Hopper, "Measuring information leakage in website fingerprinting attacks and defenses," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1977–1992.

[31] P. Lifshits, R. Forte, Y. Hoshen, M. Halpern, M. Philipose, M. Tiwari, and M. Silberstein, "Power to peep-all: Inference attacks by malicious batteries on mobile devices," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, pp. 141–158, 2018.

[32] L. Lu, E.-C. Chang, and M. C. Chan, "Website fingerprinting and identification using ordered feature sequences," in *European Symposium on Research in Computer Security*. Springer, 2010, pp. 199–214.

[33] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The hpc challenge (hpcc) benchmark suite," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, vol. 213, no. 10.1145, 2006, pp. 1 188 455–1 188 677.

[34] K. Matsumoto, N. Nakasato, T. Sakai, H. Yahagi, and S. G. Sedukhin, "Multi-level optimization of matrix multiplication for gpu-equipped systems," *Procedia Computer Science*, vol. 4, pp. 342–351, 2011.

[35] N. Matyunin, Y. Wang, T. Arul, K. Kullmann, J. Szefer, and S. Katzenbeisser, "MagneticSpy: Exploiting Magnetometer in Mobile Devices for Website and Application Fingerprinting," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, 2019, pp. 135–149.

[36] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "Confidentiality Issues on a GPU in a Virtualized Environment," in *Financial Cryptography and Data Security*, N. Christin and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 119–135.

[37] C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. A. Boano, S. Mangard, and K. Römer, "Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud," in *NDSS*, vol. 17, 2017, pp. 8–11.

[38] Microsoft, "Install NVIDIA GPU drivers on N-series VMs running Linux," https://docs.microsoft.com/en-us/azure/virtual-machines/linux/n-series-driver-setup, 2019.

[39] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and Characterizing Covert Channels on GPUs," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17, 2017, p. 354–366.

[40] 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*, 2018, pp. 2139–2153.

[41] NVIDIA, "Virtual GPU Technology," https://www.nvidia.com/en-us/data-center/virtual-gpu-technology/, 2021.

[42] NVIDIA Corporation, "CUDA Samples," 2022. [Online]. Available: https://docs.nvidia.com/cuda/cuda-samples/index.html#matrix-multiplication--cublas-

[43] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in javascript and their implications," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1406–1418.

[44] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *Cryptographers' track at the RSA conference*. Springer, 2006, pp. 1–20.

[45] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website Fingerprinting at Internet Scale." in *NDSS*, 2016.

[46] Y. Qin and C. Yue, "Website Fingerprinting by Power Estimation Based Side-Channel Attacks on Android 7," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 1030–1039.

[47] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 199–212.

[48] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 639–656.

[49] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard, "Exploiting data-usage statistics for website fingerprinting attacks on Android," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2016, pp. 49–60.

[50] D. Sullivan, O. Arias, T. Meade, and Y. Jin, "Microarchitectural Minefields: 4K-Aliasing Covert Channel and Multi-Tenant Detection in Iaas Clouds." in *NDSS*, 2018.

[51] M. Tan, J. Wan, Z. Zhou, and Z. Li, "Invisible Probe: Timing Attacks with PCIe Congestion Side-channel," in *2021 2021 IEEE Symposium on Security and Privacy (SP)*, may 2021, pp. 1016–1032.

[52] TensorFlow, "Convolutional Neural Network (CNN)," 2021. [Online]. Available: https://www.tensorflow.org/tutorials/images/cnn

[53] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. Swift, "A placement vulnerability study in multi-tenant public clouds," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 913–928.

[54] P. Vila and B. Köpf, "Loophole: Timing attacks on shared event loops in Chrome," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, 2017, pp. 849–864.

[55] A. Wilen, R. Thornburg, and J. P. Schade, *Introduction to PCI Express: a hardware and software developer's guide*. Intel, 2003.

[56] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud," *IEEE/ACM Transactions on Networking*, vol. 23, no. 2, pp. 603–615, 2014.

[57] Q. Xu, H. Naghibijouybari, S. Wang, N. Abu-Ghazaleh, and M. Annavaram, "GPUGuard: mitigating contention based side and covert channel attacks on GPUs," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 497–509.

[58] W. Xu, H. Zhang, S. Jiao, D. Wang, F. Song, and Z. Liu, "Optimizing sparse matrix vector multiplication using cache blocking method on Fermi GPU," in *2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*. IEEE, 2012, pp. 231–235.

[59] Z. Xu, H. Wang, and Z. Wu, "A measurement study on co-residence threat inside the cloud," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 929–944.

[60] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. S. Balagani, "On inferring browsing activity on smartphones via USB power analysis side-channel," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1056–1066, 2016.

[61] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 168–179.

[62] Z. Zhan, Z. Zhang, S. Liang, F. Yao, and X. Koutsoukos, "Graphics Peeping Unit: Exploiting EM Side-Channel Information of GPUs to Eavesdrop on Your Neighbors," in *2022 2022 IEEE Symposium on Security and Privacy (SP) (SP)*, 2022.

[63] Z. Zhan, Z. Zhang, and X. Koutsoukos, "Bitjabber: The world's fastest electromagnetic covert channel," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 35–45.

[64] Z. Zhang, S. Liang, F. Yao, and X. Gao, "Red alert for power leakage: Exploiting intel rapl-induced side channels," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 162–175.

[65] Z. Zhou, W. Diao, u. Liu, Z. Li, K. Zhang, and R. Liu, "Vulnerable GPU Memory Management: Towards Recovering Raw Data from GPU," *arXiv preprint arXiv:1605.06610*, 2016.

[66] Z. Zhu, S. Kim, Y. Rozhanski, Y. Hu, E. Witchel, and M. Silberstein, "Understanding the security of discrete GPUs," in *Proceedings of the General Purpose GPUs*, 2017, pp. 1–11.

Appendix A.

Table 10 shows the list of websites used in this paper for the evaluation of the website fingerprinting. This list was created primarily according to the Alexa Top websites ranking.

TABLE 10: The 100 websites used in this paper.

Adobe.com	Fidelity.com	Quora.com
Aliexpress.com	Foxnews.com	Realtor.com
Alipay.com	Gamepedia.com	Reddit.com
Allrecipes.com	Github.com	Rottentomatoes.com
Amazon.com	Glassdoor.com	Shopify.com
Aol.com	Google.com	Speedtest.net
Apartments.com	Healthline.com	Spotify.com
Apple.com	Homedepot.com	Stackoverflow.com
Att.com	Hulu.com	T-mobile.com
Baidu.com	Ign.com	Target.com
Bankofamerica.com	Imdb.com	Tripadvisor.com
Bbc.com	Imgur.com	Twitch.tv
Bestbuy.com	Indeed.com	Twitter.com
Bing.com	Instagram.com	Ups.com
Blogger.com	Intuit.com	Usa.gov
Britannica.com	Irs.gov	Usps.com
Businessinsider.com	Linkedin.com	Vk.com
Ca.gov	Live.com	Walmart.com
Capitalone.com	Lowes.com	Washingtonpost.com
Cdc.gov	Mayoclinic.org	Weather.com
Chase.com	Merriam-webster.com	Weather.gov
Cheatsheet.com	Microsoft.com	Webmd.com
Cnn.com	Msn.com	Weibo.com
Costco.com	Nbcnews.com	Wellsfargo.com
Craigslist.org	Netflix.com	Wikipedia.org
Dailymail.co.uk	Nfl.com	Xfinity.com
Duckduckgo.com	Nih.gov	Yahoo.com
Ebay.com	Npr.org	Yandex.com
Espn.com	Nypost.com	Yelp.com
Etsy.com	Nytimes.com	Youtube.com
Expedia.com	Office.com	Zillow.com
Facebook.com	Paypal.com	Zoom.us
Fandom.com	Pinterest.com	
Fedex.com	Quizlet.com	

Appendix B.

Table 11 shows the data transfer latencies corresponding to Figure 4 in Section 3.2. Each latency in Table 11 is the mean of 100 measurements in the corresponding scenario, and for each scenario, we derive such an averaged latency 10 times with the minimum and maximum being reported.

TABLE 11: Data transfer latencies measured by *Alice*. The first part of "Locked/Regular – Locked/Regular" indicates whether *Alice*'s data resides in page-locked memory or regular memory, and the second part indicates *Bob*'s.

		Locked-Locked	Regular-Locked	Locked-Regular	Regular-Regular
Base	Min	21440	34238	20749	33406
	Max	23281	40680	22982	37783
4 KB	Min	21567	35451	21345	34035
	Max	22685	333299	25002	347414
8 KB	Min	21693	34698	21429	33297
	Max	23487	339393	24917	347073
16 KB	Min	23312	34698	21977	34186
	Max	25539	334922	33403	348141
32 KB	Min	23777	35193	21062	33891
	Max	25780	45860	37475	347733
64 KB	Min	30819	35112	21126	33582
	Max	33078	48998	33019	357859
128 KB	Min	44621	39595	20844	33235
	Max	46644	52889	49793	42201
256 KB	Min	72469	39595	20942	33194
	Max	73474	52889	88003	51872
512 KB	Min	128023	39276	21017	33314
	Max	129908	53884	115199	51373