12-2020

Walls have ears: Eavesdropping user behaviors via graphics-interrupt-based side channel

Haoyu MA
Singapore Management University, hyma@smu.edu.sg

Jianwen TIAN
Singapore Management University, jwtian@smu.edu.sg

Debin GAO
Singapore Management University, dbgao@smu.edu.sg

JIA Chunfu

Citation

1

Walls Have Ears: Eavesdropping User Behaviors via Graphics-Interrupt-Based Side Channel

Haoyu Ma[1,2], Jianwen Tian[1], Debin Gao[1], and Chunfu Jia[3*]

[1] Singapore Management University, Singapore 188065
{hyma,jwtian,dbgao}@smu.edu.sg
[2] Xidian University, Xi'an, P.R. China 710126
[3] Nankai University, Tianjin, P.R. China 300350
cfjia@nankai.edu.cn

Abstract. Graphics Processing Units (GPUs) are now playing a vital role in many devices and systems including computing devices, data centers, and clouds, making them the next target of side-channel attacks. Unlike those targeting CPUs, existing side-channel attacks on GPUs exploited vulnerabilities exposed by application interfaces like OpenGL and CUDA, which can be easily mitigated with software patches. In this paper, we investigate the lower-level and native interface between GPUs and CPUs, i.e., the graphics interrupts, and evaluate the side channel they expose. Being an intrinsic profile in the communication between a GPU and a CPU, the pattern of graphics interrupts typically differs from one GPU workload to another, allowing a spy process to monitor interrupt statistics as a robust side channel to infer behavior of other processes. We demonstrate the practicality of such side-channel exploitations in a variety of attacking scenarios ranging from previously explored tasks of fingerprinting the document opened and the application launched, to distinguishing processes that generate seemingly identical displays. Our attack relies on system-level footprints rather than API-level ones and does not require injecting any payload into the GPU resource space to cause contentions. We evaluate our attacks and demonstrate that they could achieve high accuracy in the assumed attack scenarios. We also present in-depth studies to further analyze the low-level rationale behind such effectiveness.

Keywords: Side-channel attacks · GPU · Graphics interrupts · Machine learning.

1 Introduction

Graphics Processing Units (GPUs) have become increasingly important components for today's computing devices, not only because applications may involve heavy graphics and multi-media workloads, but also because of the capability of GPUs in accelerating applications in domains such as security, computational finance, and bio-informatics [9]. Such development naturally makes GPUs a tempting target to attacks aiming to leak user privacy.

[*] Prof. Chunfu Jia is the corresponding author of this paper.

Several vulnerabilities have already been demonstrated in GPU security [13, 15, 18, 23, 32, 19], most of which focused on vulnerabilities caused by defective memory management and privacy-leaking APIs from GPU-related frameworks OpenGL, OpenCL, and CUDA. This includes the latest work on GPU side channel attacks [19] which demonstrated the practicability of exploiting resource tracking APIs provided by the aforementioned frameworks to leak user privacy. These previous attacks typically require injecting an attack process into the same GPU where the victim process resides, and running it in parallel with the victim process in order to capture any footprints it leaves. Although not having been explicitly admitted in existing work, such an attack strategy is not subtle enough considering that a defense opponent could potentially be able to detect the attacks by identifying the existence of their co-residing attack processes. In addition, with the GPU side-channel attacks drawing people's attention, corresponding defense approaches against GPU memory leakage were also proposed [21, 29]. Manufacturers like Nvidia were also reported to be taking actions to mitigate the resource tracking vulnerability [20].

In this paper, we consider a less demanding threat model and identify the statistics of graphics interrupts as another source for side-channel attacks on GPUs. Graphics interrupt statistics are available to non-privileged processes on Linux-based systems, which are typically readable at `/proc/interrupt`. The key insight is that footprints of the graphics stack exist not only within the GPU resource space (exploited by existing work) but also at the interface between a CPU and a GPU (interrupts as exploited in this paper). Specifically, a GPU sends interrupt requests (IRQs) to signal key events like completion of a graphics command or reporting a GPU error. Consequently, when handling different GPU workloads, it is likely for the CPU to capture relevant IRQs in different temporal patterns. As modern operating systems provide statistics of interrupts captured at runtime, a malicious party may use the graphics interrupt statistics as signatures to infer the exact workload that is being processed by the GPU. Such an attack, unlike the existing ones, operates completely in a passive manner, i.e., it does not require any payload to be co-resident with the victim process inside the GPU resource space to cause contentions of any kind.

To demonstrate that graphics interrupts are indeed exploitable, we implemented several side-channel attacks under various attacking scenarios, including webpage fingerprinting, application inferencing, and distinguishing processes that output seemingly identical displays, on two common graphics adapters of Nvidia's and Intel's. Our attack periodically samples counts of graphics interrupts and uses the pattern of increments as a time-series signature to identify the target workloads with a machine learning model. Evaluations showed that our attacks demonstrated comparable accuracy with the latest GPU side-channel attacks based on memory APIs and performance counters [19] in webpage fingerprinting. Experiments also demonstrated accuracy as high as 99.8% in GUI-application fingerprinting. Last but not least, we found our application fingerprinting attack being capable of identifying different types of graphics workloads which present the same visual perception. Experiments on this aspect demon-

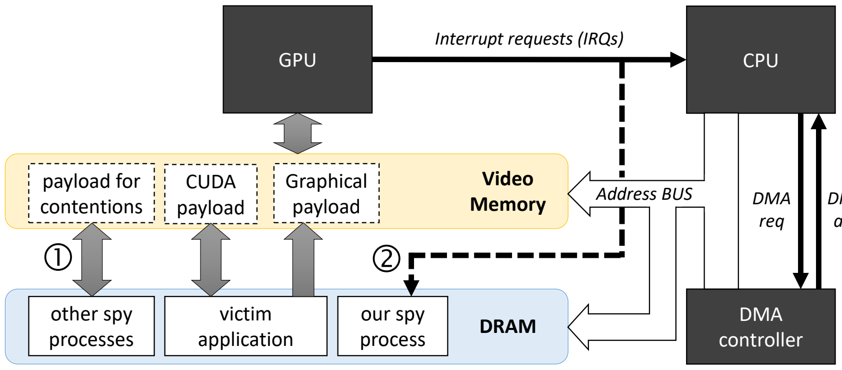


Fig. 1: Conventional GPU-related attacks and our attack strategy.

strated a high accuracy in distinguishing different video players when playing the same video, or detecting differences in playing the same video encoded with different codec.

2 Related Work

GPU-based side channels Existing GPU side-channel attacks focused on disclosing the webpage loaded and sensitive workload on cryptographic algorithms [13, 15, 18, 23, 32, 19]. Most of them exploited GPU vulnerabilities related to insecure memory management, e.g., not initializing newly allocated blocks [13, 32] and vulnerabilities in the CUDA driver [23]. Recently, Naghibijouybari et al. [19] studied the practicability of exploiting GPU resource tracking APIs.

These existing GPU side-channel attacks work according to an intrusive model in which contentions are introduced inside the GPU resource space. Figure 1 demonstrates this attacking strategy with the payload being deployed in the GPU memory. This strategy is not only intrusive to the victim process but also easy to defeat by simple countermeasures of software patching. For example, most browsers have now reduced the timer resolution and thus eliminated the timing signal used by the attacks. GPU manufacturers have also noticed the potential vulnerability caused by the resource tracking APIs and expressed plans to fix the problem with updates to OpenGL and CUDA. In this paper, we propose a novel GPU side-channel attack which works by collecting graphics-related interrupt footprints. Our approach operates passively rather than being intrusive to its victims, making it more stealthy than the existing attack strategies while being able to achieve similar effectiveness.

Interrupts Interrupts have been exploited in privacy leakage scenarios. Diao et al. [3] reported using interrupts to infer unlock patterns on Android devices. Tang et al. [25] further suggested that patterns of interrupt increment could be exploited to identify hardware related sensitive behaviors of Android apps. Another study demonstrated inferencing of instruction-granular execution states from hardware-enforced enclaves by measuring the latency of carefully timed interrupts [26]. There were also researches suggesting that attackers could establish covert channels based on the CPU time used for handling interrupts [16, 6]. In this paper, we focus specifically on using statistics of graphics interrupts as a side channel to infer GPU related activities, and study the potential risk of privacy leakage that can be caused by such an attack.

Webpage fingerprinting Early approaches for webpage fingerprinting include measuring web access time to exploit browser caching [5], measuring memory footprints [11], and analyzing network traffic [7, 22]. The relationship between webpage loading and graphics displaying behaviors was also proposed for webpage fingerprinting. For example, previous researches had proposed using display-related features of browsers to construct cross-origin timing attacks [12, 27]. Kotcher et al. [12] found that after applying CSS filters to a framed document, its rendering time becomes dependent on its content.

Proc filesystem The proc filesystem on Linux-based systems is another leakage vector that was used by side-channel attacks for inferring application UI status [2], keystrokes [30], TCP sequence numbers [24], and user identities [31].

3 Our idea

3.1 Graphics interrupts

Communication between CPUs and GPUs is critical to a computer's graphics pipeline; see Figure 1. Important components of such communication include DMA requests and acknowledgment to enable buffer sharing, the command FIFO between CPUs and GPUs, as well as interrupts from the GPU to CPU when certain events need to be processed immediately (IRQs as shown in Figure 1). These IRQs are reflections of the corresponding workload being processed.

Table 1 lists all IRQs defined in a popular open-source graphics driver on Linux, namely the drm/i915 Intel GFX Driver. Each of these interrupt types is either about a specific engine of the GPU, including the RCS (rendering), BCS (blitter copy), VCS (video en/decoding), and VECS (video enhancement) engine, or about basic events (such as vertical blanking). For example, displaying a PNG picture only involves rendering static frames which will be done by the RCS engine, while playing an MKV video may require the VCS engine to perform decoding throughout the process. This suggests that graphics interrupts are good reflections of content of the document being displayed. By the same token, the user interface of an application needs to be rendered and refreshed, which could be reflected on the corresponding graphics interrupts.

3.2 Threat model and our idea

Different from existing side-channel attacks on GPUs, our proposal considers a lower level interface which works completely in a passive manner by capturing

Table 1: Interrupt Request Definitions in drm/i915 Driver.

Name of IRQ	**Description**
GEN8_DE_MISC_IRQ	Miscellaneous interrupt raised by graphics system events (GSE) and panel self refresh events (PSR).
GEN8_DE_PORT_IRQ	The display engine port interrupt, related to AUX DDI A done event and hotplug events.
GEN8_PIPE_VBLANK	Related to vertical blanking events.
GEN8_PIPE_CDCLK_CRC_DONE	This displays core clock (CDCLK).
GEN8_PIPE_FIFO_UNDERRUN	Related to GPU's command FIFO when running into a buffer underrun.
GEN8_DE_PCH_IRQ	The south display engine interrupt, also deals with hotplug interruption and ambus events.
GEN8_GT_RCS_IRQ	Interrupt of the RCS engine which performs computing and rendering.
GEN8_GT_BCS_IRQ	Interrupt of the Blitter COPY engine.
GEN8_GT_VCS0_IRQ	Interrupt of the VCS engine used in processing videos where it performs encoding and decoding.
GEN8_GT_VCS1_IRQ	Same as the previous one.
GEN8_GT_VECS_IRQ	Interrupt of the video enhancement engine.
GEN8_GT_PM_IRQ	Related to power management events.
GEN8_GT_GUC_IRQ	Related to microprocess interruptions of the graphics microcontroller (GuC).

only statistical interrupt information provided by the OS kernel. As illustrated in Figure 1, unlike existing attacks which intrusively cause contentions in the GPU resource space (as highlighted by ① in the figure), our attack does not access GPU resources but instead reads interrupt statistics from the OS (as highlighted by ②). Although such a spy process could potentially exploit other system side channels (e.g., CPU cache and network related ones) to launch data-driven leakage attacks, our investigation here focuses on the leakage of GUI-related private information, which is more directly reflected over graphics interrupts.

Specifically, our threat model assumes a (non-privileged) spy process which periodically reads the aggregated graphics interrupt counts reported by the operating system, and uses a sliding window to extract subsequences of the collected time series of interrupt statistics. We then use a trained machine learning model to determine the task being processed by the GPU.

3.3 Challenges and experiments

Although modern operating systems like Linux report graphics interrupt statistics to any unprivileged user process via the proc filesystem (procfs), the specific types of graphics interrupts (e.g., those reported in Table 1) are aggregated in the report. It is therefore not clear whether such coarse grained reporting of graphics interrupt reveals GUI-related private information. In this paper, we evaluate the extent to which such aggregated graphics interrupt information masks or reveals workloads on the GPU, and the extent to which such masking/revealing of workload leaks private information of victim processes.

We experimented with the graphics interrupts on two different microarchitectures, namely an Nvidia GeForce GTX 760M (with Nvidia driver version 340.107) and an Intel HD Graphics 520 GT2 (with drm/i915 driver integrated in Linux kernel 5.4.2). The Nvidia unit is chosen due to its popularity and potential use in general-purpose computing. The Intel unit is chosen because it is controlled by an open-source driver integrated in the Linux kernel, which allows us to observe the low-level details of the collected graphics interrupt patterns to make our experimental results explainable. The experiments were conducted on an Ubuntu 18.04 machine with an Intel i7-4700MQ Processor and 8GB RAM, where interrupt statistics are obtained by reading `/proc/interrupt`. Note that in case of Windows, information of IRQs are managed by the interrupt descriptor table (IDT). Although there had not been software (via legitimate APIs or hacking techniques) reported specifically designed for extracting interrupt statistics on Windows, documentations suggest that it can be done in a similar way in which system call information is extracted with a kernel driver overwriting the system service descriptor table (SSDT) [14, 10].

4 Attack Scenario I: Webpage Fingerprinting

Our first attack implements webpage fingerprinting as it has been targets of many existing attack strategies (see Section 2). We make a comparative study with one of the latest attacks using GPU side channels [19]. To this end, we tested our attack on the same Alexa top 200 websites [1] with the Chrome browser and used the same basic machine learning models as in Naghibijouybari et al. for our classification, namely Gaussian Naive Bayes (NB), K-Nearest Neighbor with 3 neighbors (KNN-3), and Random Forest with 100 estimators (RF). We additionally included a state-of-the-art deep learning model on time series classification, the Residual Neural Network (ResNet) [8, 28]. This is because a previous research on time series classification [4] suggested that deep learning methods typically outperform conventional statistics-based models because they do not require preprocessing the input data to extract feature vectors. Our ResNet model used the same hyperparameters as in the original proposal [28] with 3 residual blocks each built by stacking 3 convolutional blocks consisting of a convolutional layer followed by a batch normalization layer and a ReLU activation layer. The number of filters in the residual blocks are, respectively, set to 64, 128, and 128, with the convolution operation fulfilled by three 1-D filters of sizes 9, 5, and 3 without striding.

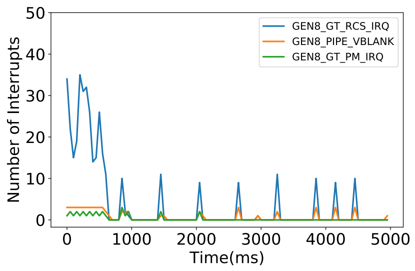
We automatically load each webpages 100 times with a script while having the timestamp of each events logged. Upon each webpage loading, we pick up 100 continuous samples of (aggregated) graphics interrupt counts collected by our spy process to form a time series corresponding to the event, with the value of each sample indicating the increment of graphics interrupts since the previous sampling. We use a sampling interval of 50ms for negligible performance overhead. Note that in such a side-channel attack, data sampling of the spy process and the targeted sensitive events are supposed to be asynchronous for mimicking a practical attacking scenario. Therefore, we start establishing a time series using the last interrupt count collected before the timestamp of its corresponding webpage loading event as its first sample. Finally, we used 10 fold cross validation to measure the accuracy of the corresponding machine learning models.

Result and analysis: As shown in Table 2, conventional machine learning models could no longer provide effective classification on side-channel leakage of graphics interrupts. Out of the three such learning methods tested, only random forest could maintain a precision of around 85% and 79%, respectively, on the Intel and Nvidia GPU. However, the state-of-the-art deep learning model on time series classification, namely ResNet, demonstrated much better accuracy on the Nvidia GPU (88.2% F-measure) and even better on the Intel GPU (92.0% F-measure). Although our results are not as good as those reported by Naghibijouybari et al. [19] when using the same conventional machine learning classifiers, we remind readers that our results are achieved without injecting GPU payload or causing contention in the GPU resource space, unlike those in Naghibijouybari et al. [19]. Such results suggest that graphics interrupts provide a valid privacy leakage vector to support side-channel attacks in the scenario of website fingerprinting, with an unprivileged spy process reading only aggregated graphics interrupts from `/proc/interrupt`.

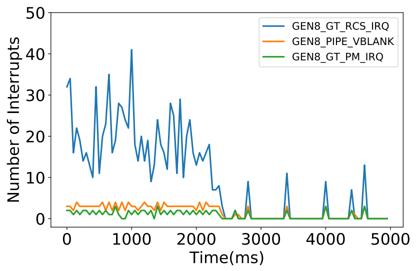
Table 2: Performance of webpage fingerprinting: average and standard deviation.

		F-Measure	**Precision**	**Recall**
Graphics Interrupt (on Intel)	NB	46.3% (7.51)	48.7% (10.6)	49.7% (8.26)
	KNN-3	32.4% (6.12)	36.5% (8.72)	34.1% (5.12)
	RF	83.1% (7.02)	85.5% (5.78)	83.9% (5.47)
	ResNet	92.0% (1.35)	93.4% (1.27)	92.2% (1.31)
Graphics Interrupt (on Nvidia)	NB	46.7% (1.76)	49.0% (2.96)	50.1% (2.02)
	KNN-3	29.3% (1.12)	31.9% (1.26)	30.5% (1.41)
	RF	76.5% (0.56)	79.3% (0.65)	77.2% (0.66)
	ResNet	88.2% (0.51)	89.9% (0.31)	88.3% (0.44)
Naghibijouybari et al. [19] (on Nvidia)	NB	83.1% (13.5)	86.7% (20.0)	81.4% (13.5)
	KNN-3	84.6% (14.6)	85.7% (15.7)	84.6% (14.6)
	RF	89.9% (11.1)	90.4% (11.4)	90.0% (12.5)

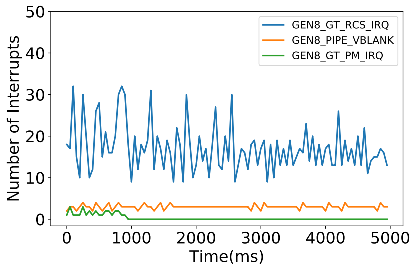
To better understand the results, we dive into the low-level details of the interrupt handling process by hooking the IRQ handlers of the drm/i915 driver to gain more detailed logs on the graphics interrupts captured, which enabled us to investigate the interrupt counts for each IRQ listed in Table 1 Note that an unprivileged attacker (main threat model used in our paper) could not obtain such information. We do this solely for the purpose of better understanding our attacking capability behind the scene. Figure 2 demonstrates such detailed interrupt patterns on opening four webpages (homepages of Google, Facebook,



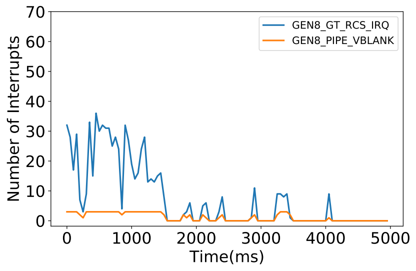
(a) Google (`Chrome`)



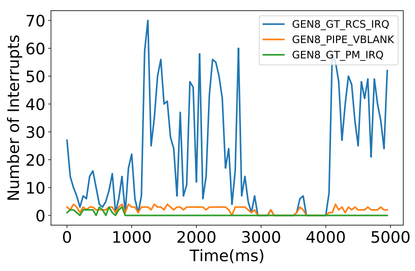
(b) Facebook (`Chrome`)



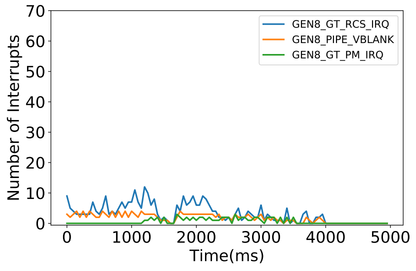
(c) Tencent (`Chrome`)



(d) Amazon (`Chrome`)



(e) Amazon (`Falkon`)

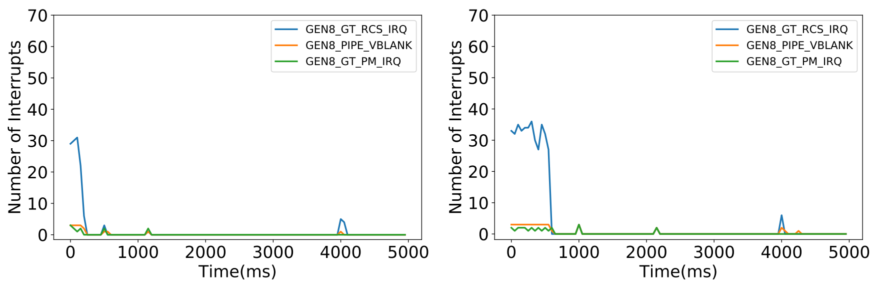


(f) Amazon (`Firefox`)

Fig. 2: Interrupt patterns (Intel) of different webpages (and the corresponding browser). Missing lines correspond to zero readings of IRQ types.

Amazon, and Tencent) using three browsers (`Chrome`, `Falkon`, and `Firefox`). Our analysis reveals two interesting observations.

First, Google's homepage has the simplest layout and correspondingly, the `GEN8_GT_RCS_IRQ` interrupt boost (indicating events signaled by the rendering engine) of its loading was the shortest among the four webpages (for around 1.2s, while those for Amazon and Facebook were respectively around 2.0s and 3.7s). In addition, all the tested webpages are static except that of Tencent which contains animation effects. As a result, we can see that the RCS interrupt pattern of Tencent's corresponds to continuous refreshing of the webpage, unlike



(a) Standard version (b) Prerendered version

Fig. 3: Interrupt patterns (Intel) of two versions of a same `Vue` webpage, with and without pre-rendering.

what happened to the other tested webpages. These confirm our intuition (see Section 3) that graphics interrupts reflect layouts and objects of the display.

Second, we found that on opening the same webpage, different browsers resulted in distinct graphics interrupt patterns (see Figure 2.d, 2.e and 2.f). This suggests that the detailed implementation of GPU acceleration in different browser engines also has a significant impact on our side-channel attacks. At this moment, we did not dig deeper into source code of the browsers to find out the decisive answer on how the implementation of different engines affects website-related graphics interrupt patterns. A reasonable guess on this is that each browser has a unique strategy with regard to the type and amount of data to be submitted to the GPU for processing, which will translate to different number of `GEN8_GT_RCS_IRQ` interrupts per sampling. We believe this is why browsing with `Firefox` causes significantly smaller amount of rendering-related interrupts on average compared with `Falkon` and `Chrome`. This also suggests that graphics interrupts could not only be used to fingerprint data (e.g., webpages) processed, but also for fingerprinting applications; see Section 5.

We also note that modern web browsers utilize the GPU to accelerate their rendering processes. Many webpages now contain optimized frontend/backend code to take advantage of it [17]. As a result, it is likely for different webpages to have adopted different acceleration techniques including server- and client-side rendering, rehydration, and prerendering, which also leads to differences in their resulting graphics interrupt patterns. To confirm this intuition, we used an open-source prerendering tool, `pre-render`[4], to convert a simple `Vue` webpage into its pre-rendered variant[5], and recorded the corresponding graphics interrupts when the two pages were loaded and displayed in `Chrome`. Figure 3 showed noticeable differences between the interrupt patterns on the two instances.

[4] https://github.com/kriasoft/pre-render.
[5] The tested webpage can be accessed via http://pay.his.cat/app.html (original version) and http://pay.his.cat/index.html (prerendered version).

5 Attack Scenario II: GUI Application Fingerprinting

Our second attack attempts to fingerprint GUI applications with the same spy process monitoring graphics interrupts. Application fingerprinting has implications not only on revealing end user activities (e.g., which application is launched), but also on picking the best machine learning model for webpage fingerprinting. This is especially important since different browsers result in different graphics interrupt patterns even for the same webpages (see Section 4). With an effective application fingerprinting, it could then be possible to first identify the specific web browser being used and then pick the suitable machine learning model for webpage fingerprinting to achieve optimized accuracy.

We downloaded 20 popular applications on Ubuntu as test subjects (see Table 3 for the list of selected applications), and launched each of them 100 times with our scripts. Note that to demonstrate the connection between this attack and webpage fingerprinting, we included two web browsers, Firefox and Brave, into the test set. Since the goal of this attack is to infer the application launched, we did not further use them to process any input. Again, each time a subject application is launched, 100 samples (with sampling interval at 50ms) of interrupt count were collected to form the corresponding time series.

Table 3: Subjects for our application fingerprinting attack.

Application	**Category**	**Application**	**Category**
`Inkscape`	graphics editor	`libreoffice`	text editor
`GIMP`		`Notepadqq`	
`Krita`		`ClamTk`	antivirus
`atril`	doc viewer	`Deluge`	download
`Thunderbird`	e-mail	`Audacity`	multimedia
`Geary`		`Clementine`	
`Pidgin`	social	`Kdenlive`	
`Corebird`		`VLC`	
`Neofetch`	system management	`Firefox`	web browser
`Synaptic`		`Brave`	

Result: Our attack on application fingerprinting demonstrated very high accuracy with all tested machine learning models on both Nvidia and Intel GPUs (as shown in Table 4). This suggests that graphics interrupts could effectively leak information about the running desktop applications, indicating good generality of our application fingerprinting attack. We believe that this is due to the higher degree of flexibility in the design of GUI of desktop applications, compared to the design of webpages which is governed by the `html` protocol.

6 Attack Scenario III: Beyond Visual Perception

In our attack scenarios I and II, webpage and application fingerprinting are both targeting objects that present unique visual perception to human users. The idea

Table 4: Results of application fingerprinting, average and standard deviation.

		F-Measure	**Precision**	**Recall**
Intel	NB	98.7% (0.26)	98.8% (0.19)	98.7% (0.26)
	KNN-3	91.4% (3.53)	91.9% (2.99)	91.5% (3.51)
	RF	99.6% (0.07)	99.7% (0.06)	99.7% (0.07)
	ResNet	99.5% (1.09)	99.5% (0.91)	99.6% (1.11)
Nvidia	NB	97.9% (3.09)	98.2% (1.91)	97.9% (3.31)
	KNN-3	95.4% (3.62)	95.6% (2.89)	95.5% (3.51)
	RF	99.3% (1.58)	99.4% (1.17)	99.3% (1.71)
	ResNet	99.8% (0.08)	99.8% (0.07)	99.8% (0.08)

is that each unique GUI or view of the webpages correspond to unique workload on the GPU, resulting in classifiable graphics interrupts. In this section, we investigate a more challenging problem in using aggregated graphics interrupt to differentiate objects with the same visual perception, i.e., can we differentiate something even a human being cannot differentiate with visual inspection? Such a capability has a strong implication on the research of human factors in security, e.g., in assisting human to detect phishing websites, to detect re-packaged applications, and in digital forensics.

As a first step in evaluating such a capability, we take video playback as an example. Specifically, we consider the following two experimental settings:

- Same video clip encoded with the same codec, played back with different video players, in which we played back a video clip in FLV format using four different video players (`VLC`, `SMPlayer`, `TOTEM`, and `MPV`);
- Same video clip encoded with different codec and played back with the same video player, in which we encoded a video using four codec (H264, MPEG4, WMV2, and XIVD) and had them played back using the VLC player.

Time series of graphics interrupt counts in this experiment were collected from the 2nd to the 6th seconds into the subject video[6] to avoid potential noise from setting up GUI of the video players. We repeated the experiment 100 times and performed a 10 fold validation over the collected data as usual. Unlike the previous experiments, here we only used ResNet as the classifier since it outperformed the other tested models (especially in webpage fingerprinting).

Result and analysis: Table 5 shows the performance of our attack in the two settings listed above. We found that in the scenario of distinguishing different video players, our attack worked accurately without a single misclassification. While in the scenario of distinguishing different codec, the attack on the Nvidia GPU outperformed that on the Intel (86.3% vs. 70.2%).

To further understand the low-level details behind such results, we again leveraged the hooked drm/i915 driver to demonstrate the IRQ-specific patterns of the tested events (as was done in Section 4). Figure 4 demonstrates such

[6] The video used can be found at https://www.dropbox.com/sh/vqd8ffi7eer8urd/AAAU9MYDg1bKkTtsSzjkpUp5a

Table 5: Distinguishing video playback events with ResNet: average and standard deviation.

		F-Measure	**Precision**	**Recall**
Diff players	**Intel**	100% (0)	100% (0)	100% (0)
	Nvidia	100% (0)	100% (0)	100% (0)
Diff codecs	**Intel**	70.2% (37.0)	76.0% (46.8)	72.0% (31.0)
	Nvidia	86.3% (24.4)	90.0% (19.4)	87.0% (21.0)

detailed patterns for six tested events (three for each setting). We can see that all demonstrated interrupt patterns show typical features of stream displaying, making different patterns appear to be similar to a certain extent. We believe this is the main contribution to the relatively low accuracy of our attack on the Intel GPU. On top of this, there are still two observations worth noting.

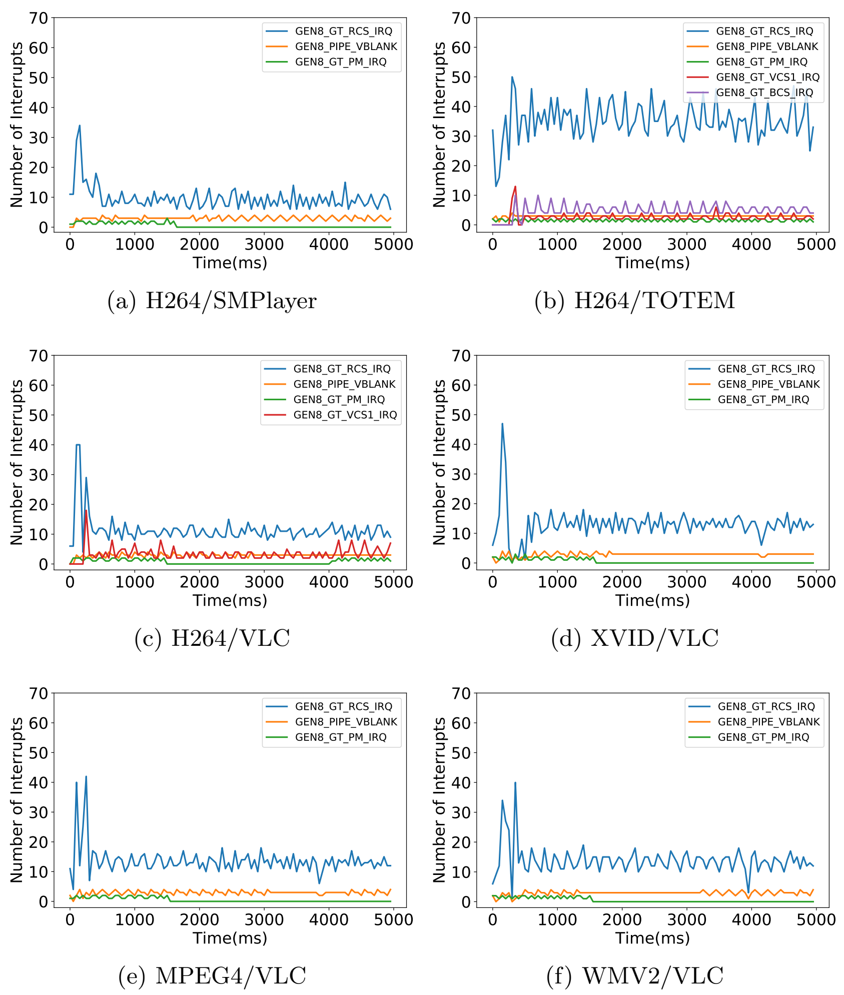
First, we found that different video player engines use different rendering techniques. Figure 4.a, 4.b, and 4.c show that when playing the same FLV video, `VLC`, `SMPlayer`, and `TOTEM` used different GPU engines. Specifically, `SMPlayer` relied purely on the basic RCS engine, while both `VLC` and `TOTEM` used the VCS engine (VCS engine is for video encoding and decoding). This means that `SMPlayer` resorts to a pure software solution while `VLC` and `TOTEM` utilized hardware acceleration. Furthermore, we observed that `TOTEM` additionally leveraged the BCS engine, i.e., the blitter engine, to accelerate 2D rendering. We believe that such differences on the implementation details are the main factors that make the tested video players distinguishable from one another.

Secondly, the same video player also behaves differently when decoding videos of different codec. In the case of `VLC` playing the H264 videos, patterns of `GEN8_GT_VCS1_IRQ` interrupts can be observed, indicating that hardware accelerated decoding were leveraged. However, the other cases, i.e., `VLC` playing the XVID, MPEG4 and WMV videos, only involved the RCS engine, indicating pure software-level decoding. To demonstrate how such implementation details affect effectiveness of our attack, we present the heatmap for classification results of distinguishing the aforementioned 4 types of codec on the Intel GPU in Figure 5, where we can see that our attack never misclassified any event of playing back the H264 video — unlike the playback of other clips where a certain level of ambiguity existed.

7 Additional Experiments, Discussion, and Limitation

7.1 Tradeoff between Accuracy and Timeliness

When considering an attack scenario with on-the-fly monitoring of GPU usage, classifications are expected to be made in real time. As discussed in Section 3, our spy process uses a sliding window to feed its machine learning model subsequences of the interrupt time series. Intuitively, a larger sliding window (corresponding to longer inputs to our deep learning model and better accuracy) will



(a) H264/SMPlayer (b) H264/TOTEM

(c) H264/VLC (d) XVID/VLC

(e) MPEG4/VLC (f) WMV2/VLC

Fig. 4: Interrupt patterns (Intel) of playing the same video using different video players and codec. Missing lines correspond to zero readings of the IRQ types.

result in longer latencies, given that classification only happens after the subsequences are collected. Therefore, in this subsection, we investigate the impact of reducing the length of such subsequences on the effectiveness of our attack.

We changed the length of subsequences with 10 different settings to train new machine learning models and observe the accuracy of them. Note that the sampling rate remains at 50ms to minimize workload of our spy process. As presented in Figure 6, the shortest interrupt time series length for reaching 99% accuracy in application fingerprinting was 50 samples, while that for reaching 80% accuracy in webpage fingerprinting was 60 (or 80 if we wish to reach 85% accuracy). This difference implies that launching applications splashes differently

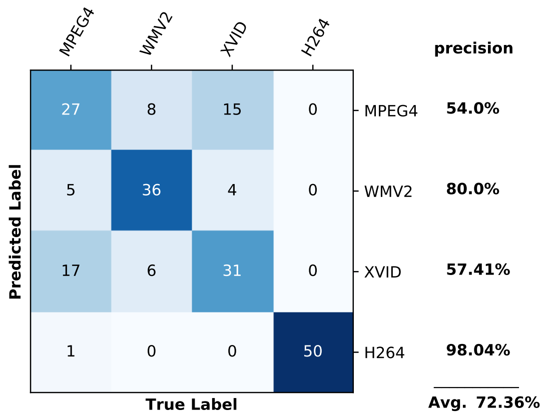


Fig. 5: Classifying (using ResNet) video playback of different codec using the same video player.

from the very beginning while loading webpages with the same browser splashes differently within a slightly longer period. Also, such result suggests that using time series of 60 to 80 samples, which translates to 3 to 4 seconds, would be good hyperparameter configurations to optimize the accuracy and timeliness tradeoff.

7.2 Robustness Against Noise

Since interrupt statistics is a fine-grained measurement, attacks based on such information could be interfered by other events which trigger screen refreshing or redrawing. The most typical example of such noise source is the movement of mouse cursor, in which areas at the past and present locations of the cursor have to be redrawn. Another possible scenario is when a multitasking user is conducting more than one screen redrawing activities at the same time, e.g., reading a document while playing a video simultaneously. We tested the robustness of our webpage fingerprinting attacks by collecting a group of new interrupt time series from the Nvidia GPU, in which the experiments involved manually moving the mouse cursor during the process of webpage loading, or having a random movie being played throughout the experiment. The test was conducted on the top 50 websites (given by Alexa) and repeated 100 times for each webpage. Two classification strategies were tested: the first to train two ResNet models for the "noisy" and "clean" (free of noise) data, respectively, under an assumption that the two environments could be effectively differentiated (e.g., by observing mouse movement interrupts or by monitoring other side channels like CPU utilization), while the second to train only one model with both types of data mixed.

From Table 6 we can see that when classifying noisy data with mouse clicks as the noise source, F-measure of both strategies only slightly exceeded 54%. These strategies performed better in classifying data with video playing as the noise source, but the resulted F-measures were still only around 68%. Meanwhile, when

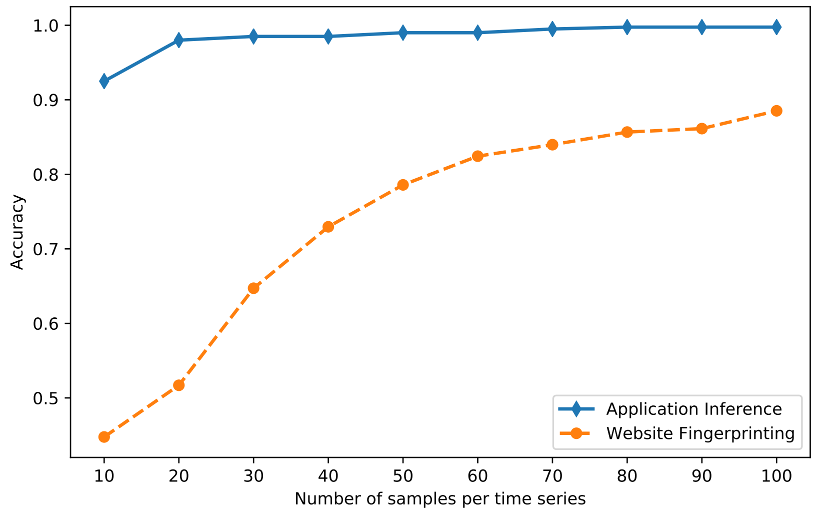


Fig. 6: Webpage and application fingerprinting with shorter interrupt time series.

Table 6: Webpage fingerprinting with/without noise: average F-measure.

	clean data	**noisy data**	
		mouse induced noise	video induced noise
Dual models	88.2%	54.2%	67.9%
Mixed model	85.2%	54.6%	67.8%

using one model to classify both types of data, F-measure of classifying clean data is 3% less than that with two different models. Therefore, we consider the robustness against noise a limitation of our attack, which could also be pointing toward a potential mitigation against GPU side-channel attacks.

8 Conclusion

This paper systematically studied the possibility of utilizing graphics interrupts as a leakage vector to drive GPU side-channel attacks. We introduced a series of attack scenarios in which graphics interrupt patterns were leveraged to respectively infer webpage opening, GUI application starting, and GUI tasks with the same graphics perception. Being a passive attack strategy, our attacks demonstrated high accuracy in the tested attack scenarios, suggesting that graphics interrupts could indeed leak sensitive information related to user activities.

Acknowledgment

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-100E-2018-

004), National Natural Science Foundation of China (Grant No. 61702399 and 61972215) and National Key R&D Program of China (2018YFA0704703). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and AI Singapore.

References

1. Alexa: The top 500 sites on the web (2019), https://www.alexa.com/topsites
2. Chen, Q.A., Qian, Z., Mao, Z.M.: Peeking into your app without actually seeing it: UI state inference and novel android attacks. In: Proc. of the 23rd USENIX Security Symposium. pp. 1037–1052 (2014)
3. Diao, W., Liu, X., Li, Z., Zhang, K.: No pardon for the interruption: New inference attacks on android through interrupt timing analysis. In: Proc. of the 2016 IEEE Symposium on Security and Privacy. pp. 414–432. IEEE (2016)
4. Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data Mining and Knowledge Discovery **33**(4), 917–963 (2019)
5. Felten, E.W., Schneider, M.A.: Timing attacks on web privacy. In: Proc. of the 7th ACM conference on Computer and communications security. pp. 25–32. ACM (2000)
6. Gay, R., Mantel, H., Sudbrock, H.: An empirical bandwidth analysis of interrupt-related covert channels. International Journal of Secure Software Engineering **6**(2), 1–22 (2015)
7. Hayes, J., Danezis, G.: k-fingerprinting: A robust scalable website fingerprinting technique. In: Proc. of the 25th USENIX Security Symposium. pp. 1187–1203 (2016)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proc. of the 29th IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)
9. Hwu, W.M.W.: GPU computing gems emerald edition. Elsevier (2011)
10. inaz2: Abusing interrupts for reliable windows kernel exploitation (2015), https://www.slideshare.net/inaz2/abusing-interrupts-for-reliable-windows-kernel-exploitation-en
11. Jana, S., Shmatikov, V.: Memento: Learning secrets from process footprints. In: Proc. of the 2012 IEEE Symposium on Security and Privacy. pp. 143–157. IEEE (2012)
12. Kotcher, R., Pei, Y., Jumde, P., Jackson, C.: Cross-origin pixel stealing: timing attacks using css filters. In: Proc. of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 1055–1062. ACM (2013)
13. Lee, S., Kim, Y., Kim, J., Kim, J.: Stealing webpages rendered on your browser by exploiting gpu vulnerabilities. In: Proc. of the 2014 IEEE Symposium on Security and Privacy. pp. 19–33. IEEE (2014)
14. Lukan, D.: Hooking the system service dispatch table (ssdt) (2014), https://resources.infosecinstitute.com/hooking-system-service-dispatch-table-ssdt
15. Luo, C., Fei, Y., Luo, P., Mukherjee, S., Kaeli, D.: Side channel power analysis of a gpu aes implementation. In: Proc. of the 2015 33rd IEEE International Conference on Computer Design. pp. 281–288. IEEE (2015)

16. Mantel, H., Sudbrock, H.: Comparing countermeasures against interrupt-related covert channels in an information-theoretic framework. In: Proc. of the 20th IEEE Computer Security Foundations Symposium. pp. 326–340. IEEE (2007)
17. Miller, J., Osmani, A.: Rendering on the web (2019), https://developers.google.com/web/updates/2019/02/rendering-on-the-web
18. Naghibijouybari, H., Khasawneh, K.N., Abu-Ghazaleh, N.: Constructing and characterizing covert channels on gpgpus. In: Proc. of the 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 354–366. IEEE (2017)
19. Naghibijouybari, H., Neupane, A., Qian, Z., Abu-Ghazaleh, N.: Rendered insecure: Gpu side channel attacks are practical. In: Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 2139–2153. ACM (2018)
20. Nvidia: Security notice: Nvidia response to "rendered insecure: Gpu side channel attacks are practical" - november 2018 (2018), https://shorturl.at/efJO6
21. Olson, L.E., Power, J., Hill, M.D., Wood, D.A.: Border control: Sandboxing accelerators. In: Proc. of the 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 470–481. IEEE (2015)
22. Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., Wehrle, K.: Website fingerprinting at internet scale. In: Proc. of the Network and Distributed System Security Symposium 2016 (2016)
23. Pietro, R.D., Lombardi, F., Villani, A.: Cuda leaks: a detailed hack for cuda and a (partial) fix. ACM Transactions on Embedded Computing Systems **15**(1), 15 (2016)
24. Qian, Z., Mao, Z.M., Xie, Y.: Collaborative tcp sequence number inference attack: how to crack sequence number under a second. In: Proc. of the 2012 ACM conference on Computer and communications security. pp. 593–604. ACM (2012)
25. Tang, X., Lin, Y., Wu, D., Gao, D.: Towards dynamically monitoring android applications on non-rooted devices in the wild. In: Proc. of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks. pp. 212–223. ACM (2018)
26. Van Bulck, J., Piessens, F., Strackx, R.: Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic. In: Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 178–195. ACM (2018)
27. Van Goethem, T., Joosen, W., Nikiforakis, N.: The clock is still ticking: Timing attacks in the modern web. In: Proc. of the 22Nd ACM SIGSAC Conference on Computer and Communications Security. pp. 1382–1393 (2015)
28. Wang, Z., Yan, W., Oates, T.: Time series classification from scratch with deep neural networks: A strong baseline. In: Proc. of the 2017 international joint conference on neural networks. pp. 1578–1585. IEEE (2017)
29. Yao, Z., Ma, Z., Liu, Y., Amiri Sani, A., Chandramowlishwaran, A.: Sugar: Secure gpu acceleration in web browsers. In: ACM SIGPLAN Notices. vol. 53, pp. 519–534. ACM (2018)
30. Zhang, K., Wang, X.: Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems. In: Proc. of the 18th USENIX Security Symposium. vol. 20, p. 23 (2009)
31. Zhou, X., Demetriou, S., He, D., Naveed, M., Pan, X., Wang, X., Gunter, C.A., Nahrstedt, K.: Identity, location, disease and more: Inferring your secrets from android public resources. In: Proc. of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 1017–1028. ACM (2013)
32. Zhou, Z., Diao, W., Liu, X., Li, Z., Zhang, K., Liu, R.: Vulnerable gpu memory management: towards recovering raw data from gpu. Proceedings on Privacy Enhancing Technologies **2017**(2), 57–73 (2017)