# On the Effectiveness of Using Graphics Interrupt as a Side Channel for User Behavior Snooping

Haoyu Ma, Jianwen Tian, Debin Gao, and Chunfu Jia

**Abstract**—Graphics Processing Units (GPUs) are now a key component of many devices and systems, including those in the cloud and data centers, thus are also subject to side-channel attacks. Existing side-channel attacks on GPUs typically leak information from graphics libraries like OpenGL and CUDA, which require creating contentions within the GPU resource space and are being mitigated with software patches. This paper evaluates potential side channels exposed at a lower-level interface between GPUs and CPUs, namely the graphics interrupts. These signals could indicate unique signatures of GPU workload, allowing a spy process to infer the behavior of other processes. We demonstrate the practicality and generality of such side-channel exploitation with a variety of assumed attack scenarios. Simulations on both Nvidia and Intel graphics adapters showed that our attack could achieve high accuracy, while in-depth studies were also presented to explore the low-level rationale behind such effectiveness. On top of that, we further propose a practical mitigation scheme which protects GPU workloads against the graphics-interrupt-based side-channel attack by piggybacking mask payloads on them to generate interfering graphics interrupt "noises". Experiments show that our mitigation technique effectively prohibited spy processes from inferring user behaviors via analyzing runtime patterns of graphics interrupt with only trivial overhead.

**Index Terms**—Side-channel attacks, GPU, graphics interrupts, machine learning.

## 1 Introduction

Graphics Processing Units (GPUs) have become increasingly important components in computing devices. Not only more and more applications now involve heavy graphics and multi-media workload, but the development of general-purpose computing has also highlighted the value of GPUs in accelerating tasks related to security, computational finance, and bioinformatics [9]. Naturally, a price is always exacted for what technological advance bestows, and the development of GPU applications has nurtured a tighter bond between these hardware components and vital system security factors, such as user privacy. This makes GPUs now a tempting target for attacks aiming at information leakage.

### 1.1 GPU-based side channels

Previous research had demonstrated several vulnerabilities on GPU security [14], [16], [20], [21], [25], [34], mostly due to defective memory management and privacy-leaking APIs from GPU-related frameworks, e.g., not initializing newly allocated blocks [14], [34] and vulnerabilities in the CUDA driver [25]. Recently, Naghibijouybari et al. [21] also 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. Although not having been explicitly admitted in existing work, such an intrusive strategy is not subtle enough considering that a defense opponent could detect such attacks by identifying the existence of co-residing attack processes. In addition, attacks based on such a strategy are not hard to defeat via 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 [22]. Last but not the least, given that memory contention is in essence a security design negligence, mitigation at system level had also been proposed to eliminate the aforementioned attacks with GPU memory sandboxing [23] and GPU virtualization [31].

*Haoyu Ma is currently with School of Computing and Information Systems, Singapore Management University and School of Cyber Engineering, Xidian University, e-mail:hyma@xidian.edu.cn.*
*Jianwen Tian and Debin Gao are both with School of Computing and Information Systems, Singapore Management University, e-mail:{jwtian,dbgao}@smu.edu.sg.*
*Chunfu Jia (corresponding author) is with College of Cyber Science, Nankai University, e-mail:cfjia@nankai.edu.cn.*

### 1.2 Our contribution

This paper considers the possibility of conducting side-channel attacks on GPUs under a less demanding threat model. We identify graphics interrupt statistics as the side-channel source for such attacks, which is available to non-privileged processes on Linux-based systems and 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): a GPU sends interrupt requests (IRQs) to the CPU to signal key events like completion of a graphics command or reporting a hardware error. When the GPU is handling different workloads, relevant GPU IRQs would be captured by the CPU in different temporal patterns. Therefore, it is possible for a malicious process to exploit the runtime statistics of graphics interrupts as signatures to infer exactly what is being processed by the GPU. Such an attack, unlike those studied in existing work, is fully passive in that it does not require co-residing a payload with the victim process inside the GPU resource space to cause contentions of any kind.

To demonstrate that graphics interrupts are indeed exploitable, we have simulated several side-channel attacks under various scenarios, including document (webpages and PDF files) fingerprinting, application inferencing, distinguishing processes with seemingly identical displays, and recognizing non-GUI applications, on two common graphics adapters of Nvidia's and Intel's. Our attack process periodically samples graphics interrupt counts and uses the pattern of its increment as a time-series signature to identify the target workload with the assistance of a trained machine learning model. Evaluations showed that our attacks demonstrated comparable effectiveness with the latest GPU side-channel attacks based on memory APIs and performance counters [21] in webpage fingerprinting, and an accuracy as high as 99.8% in GUI-application fingerprinting. As an interesting observation, we found our application fingerprinting attack being capable of identifying different types of graphics workload that presents the same visual perception. Experiments on this aspect demonstrated high accuracy in distinguishing different video players when playing the same video or detecting differences in playing the same video encoded with different codecs. Finally, we also demonstrated that graphics interrupts are valid leakage vectors for inferring workload of GPU general-purpose computing (GPGPU), such as the execution of cryptographic and neural network algorithms.

Last but not the least, we further explore potential defenses against the threat of graphics-interrupt-based GPU side-channel attacks. Specifically, we propose a mitigation scheme which, given a potential victim program, piggybacks a trivial but randomized graphics payload on it as a "mask". With the two components running simultaneously, interrupt patterns observed by side-channel attackers are blended with "noisy" interrupt counts caused by our mask payload, creating concealment to the signature of actual program behavior and therefore making the attacks difficult to succeed. Evaluations on a proof-of-concept implementation of our mitigation showed that using an 1-pixel redrawing operation as the mask, our mitigation effectively reduced the accuracy of graphics-interrupt-based webpage fingerprinting to lower than 57% while inducing negligible performance overhead.

In summary, the main contributions of this work include the followings[1].

- We demonstrate that it is possible to infer GPU related operations using temporal pattern of graphics interrupts, and argue that by exploiting such a lower-level and native source of leakage, a side-channel attack on GPU can be fully passive.
- We conduct a systematic study on the practicality and generality of the interrupt-based GPU side-channel attacks, showing that in various attack scenarios (not just the typical webpage fingerprinting) this attack could be equally effective as the previously proposed ones based on memory contentions.
- After digging into details of interrupt patterns of various types of GPU-related program operations, we find that the proposed attack in fact captures more information than the displayed visual effects, making it capable of identifying an even wider range of GPU workloads, e.g., the computing of cryptographic algorithms or deep learning models.
- We propose a defense scheme against the proposed attacks, which uses trivial graphics operations to introduce randomized interrupt requests into the overall interrupt statistics and therefore confuses the attacks. We have implemented a proof-of-concept demonstration of the proposed defense, and evaluated its effectiveness and cost.

1. This paper is an extended version of an earlier conference paper [17].

### 1.3 Paper organization

The rest of this paper is organized as follow. In Section 2.1 we briefly introduce the role of graphics interrupts in CPU-GPU communication, and illustrate the intuition on why such a character makes graphics interrupts a possible side-channel leakage vector. Next, in Sections 2.2, 2.3, 2.4, 2.5, and 2.6 we introduce the various attacking scenarios based on graphics interrupts, demonstrate the effectiveness of these attacks, and present detailed analyses on the rationale behind the results. Following that, in Section 3 we present our mitigation scheme of the proposed attacks, with a proof-of-concept implementation as well as a brief evaluation. Then in Section 4, we discuss the accuracy-efficiency trade-off involved in graphics-interrupt-based GPU side-channel attacks, as well as the capability of attackers with root privilege. Previous researches related to this paper are given in Section 5. And finally, we conclude the paper in Section 6.

## 2 User Behavior Snooping Using Graphics-Interrupt-Based GPU Side Channel

### 2.1 Overview

#### 2.1.1 *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 graphics 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 GPU engine, 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 contents being displayed. By the same token, the user interface of an application needs to be rendered and refreshed,

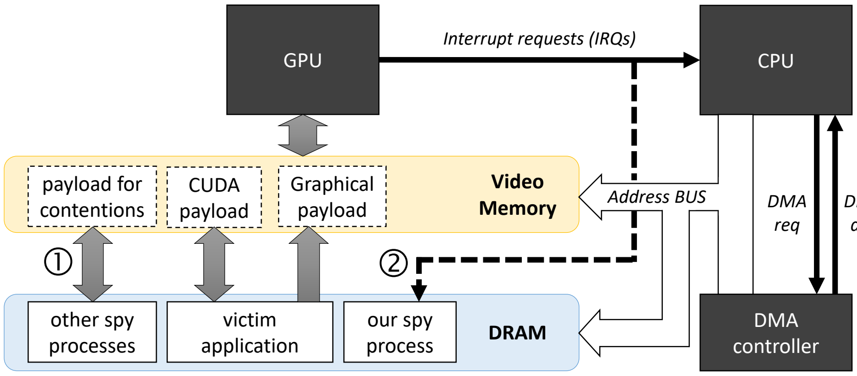


Fig. 1: Conventional GPU-related attacks and our attack strategy.

which could also be reflected on the corresponding graphics interrupts.

### 2.1.2 *Threat model and our idea*

Different from existing side-channel attacks on GPUs [14], [21], [25], [34], our proposal considers a lower-level interface which works completely in a passive manner by capturing only statistical interrupt information provided by the OS kernel. 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. As illustrated in Figure 1, our proposal differs from existing attacks which intrusively cause contentions in the GPU resource space (as highlighted by ① in the figure). Instead, our attack does not access any GPU resource at all, but only reads interrupt statistics from outside of the GPU (as highlighted by ②). Although such a spy process could potentially exploit other system side channels (e.g., CPU cache and network traffics) to launch data-driven leakage attacks, our investigation here focuses on the leakage of GPU-related information, which is more informative in inferring GUI of the application and might not be acquirable via data-driven leakage attacks.

### 2.1.3 *Challenges and experiments*

Although modern operating systems like Linux report graphics interrupt statistics to any non-privileged user process via the `proc` filesystem (procfs), the specific types of graphics interrupts (e.g., those reported in Table 1) are aggregated into a single count. It is therefore not clear whether such coarse-grained reporting is sufficient for revealing user privacy. In this paper, we evaluate the extent to which such aggregated graphics interrupt information masks or reveals workload 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 LTS 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 is 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) [10], [15].

## 2.2 Attack Scenario I: Document Fingerprinting

Our first attack implements document fingerprinting in which the victim is an application process with (part of) a document being graphically displayed, and the goal of the attack is to identify which document is being opened. We first consider webpage fingerprinting as it has been the target of many existing attack strategies (see Section 2.1). We then move on to the scenario of identifying (script-free) PDF documents to gain deeper insights, considering that most real-world webpages are script-heavy.

### 2.2.1 *Webpage Fingerprinting*

**Experimental design** We make a comparative study with one of the latest attacks using GPU side channels [21]. 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], [30]. 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 pre-processing the input data to extract feature vectors. Our ResNet model used the same hyperparameters as in the original proposal [30] with 3 residual blocks each

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). |

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 were, 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 loaded each webpage 100 times with a script, and logged the timestamp of each event. Upon each webpage loading, we picked 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 used 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 started 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 folder 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. [21] 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. [21]. 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`.

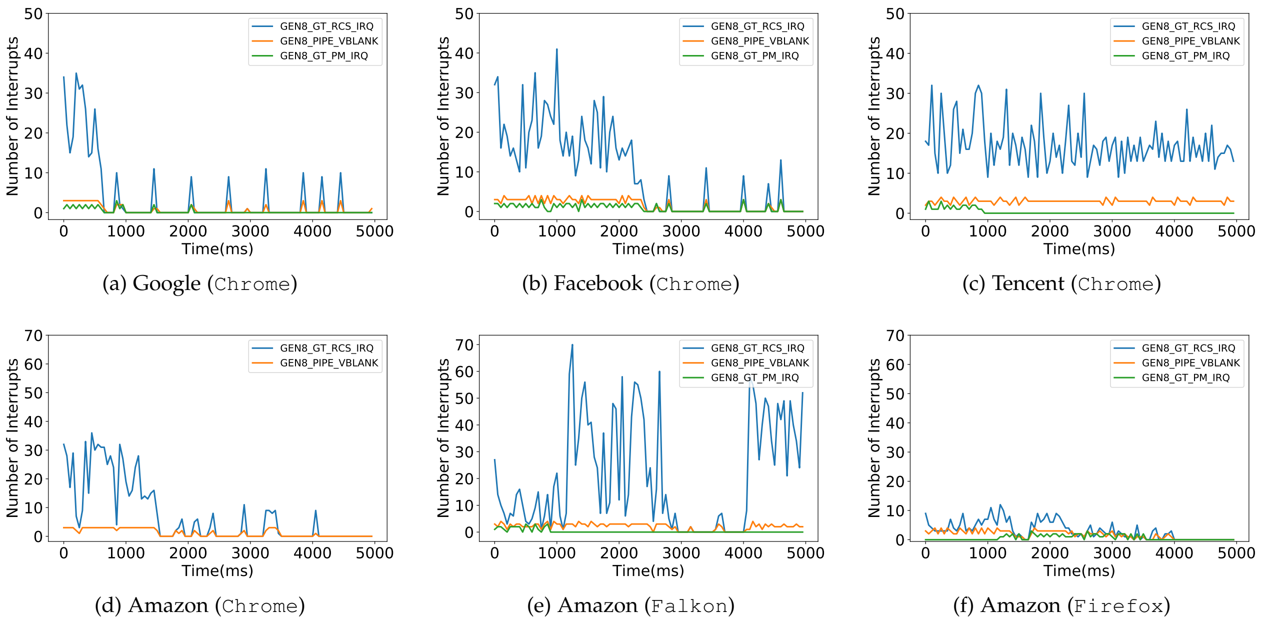
To better understand the results, we dive into the low-level details of the interrupt handling process by hooking the IRQ handlers of the drm/i915 driver to gain more detailed logs on the graphics interrupts captured, which enable us to investigate the interrupt counts for each IRQ listed in Table 1. 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, 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 what happened to the other tested webpages. These confirm our intuition (see Section 2.1) that graphics interrupts reflect layouts and objects of the display.

Second, we found that on opening the same webpage, different browsers result 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. 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 compared with `Falkon` and `Chrome`. This also suggests that graphics

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. [21] (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) |



(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.

interrupts could not only be used to fingerprint data (e.g., webpages) processed, but also to fingerprint applications; see Section 2.3.

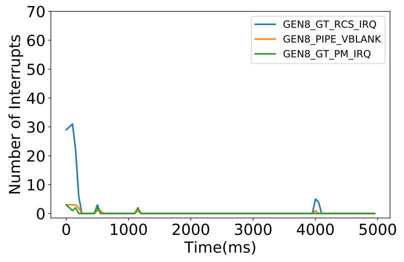
We also note that modern web browsers utilize the GPU to accelerate their rendering processes. Many webpages now contain optimized frontend/backend code to take advantage of it [19]. As a result, different webpages can adopt different acceleration techniques including server- and client-side rendering, rehydration, and prerendering, which lead to differences in their resulting graphics interrupt patterns. To confirm this intuition, we used an open-source prerendering tool, `pre-render`[2] to convert a simple `Vue` webpage into its pre-rendered variant[3], 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.

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

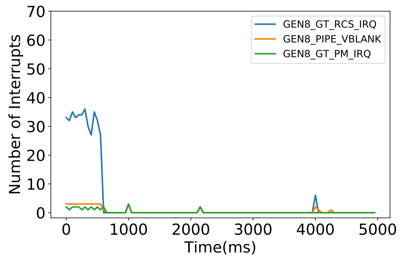
### 2.2.2 *PDF document fingerprinting*

Although our experiments with the Alexa top 200 websites show good results in general webpage fingerprinting, we do not have control on the exact layout of the webpages and therefore could not tell precisely what kind of documents are easier or harder to fingerprint. In addition, webpages are usually script-heavy, making their resulting GPU interrupts arguably code dependent (due to the scripts embedded) rather than purely data driven. Therefore, here we design a more controlled set of experiments on PDF document fingerprinting, in which we intentionally select some similar and distinct script-free PDF documents to experiment with. Given that the previous experiment has already demonstrated the superior capability of deep learning over conventional machine learning methods in webpage fingerprinting, in this experiment, we choose to use ResNet only as the classifier.

**Experimental design** We selected 20 PDF documents consisting of 4 categories of research papers (RP), magazines (Mag), newspapers (News), and fiction books (Book); see the Appendix A for snapshots of these 20 documents. Each category contains documents with distinct characteristics.



(a) Standard version



(b) Prerendered version

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

For example, all research papers are from the same security conference published in the same year under the same layout and typesetting requirements, and are seemingly more difficult to be differentiated. Documents from different categories are easier to be told apart, e.g., all magazines have significantly more photos. Each of these documents was opened with `evince` to display its first page as well as a preview of other pages. Our spy process collected aggregated graphics interrupt counts during these processes, and performed classification to infer which document was opened. Note that in this experiment, we first launched `evince` (with nothing to be opened), and then waited for 10 seconds before starting collecting data to prevent the interrupt pattern of application starting from kicking in and interfering with the results.

**Result** Figure 4 shows the heatmaps for classification results on the two GPUs tested. Again, we find the accuracy of our attack on the Intel GPU (86.0%) to be higher than that on the Nvidia's (58.97%). Specifically, our attack on the Intel GPU is effective for in-group and cross-group fingerprinting for magazines, newspapers, and books. However, low accuracy is observed in classifying research papers, which is mainly due to the misclassification within the same category. Intuitively, this is because that our selected research papers from the same conference proceeding are all of the similar layout (formatted using the same template). It's worth mentioning that when reducing the granularity of classification from individual file level to the category level, the accuracy of our attacks increases to 96.8% and 73.2%, respectively, on the Intel and the Nvidia GPUs. This suggests that graphics interrupts could at least be exploited to reveal the flavor of documents being displayed.

TABLE 3: Subjects for our application fingerprinting attack.

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

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

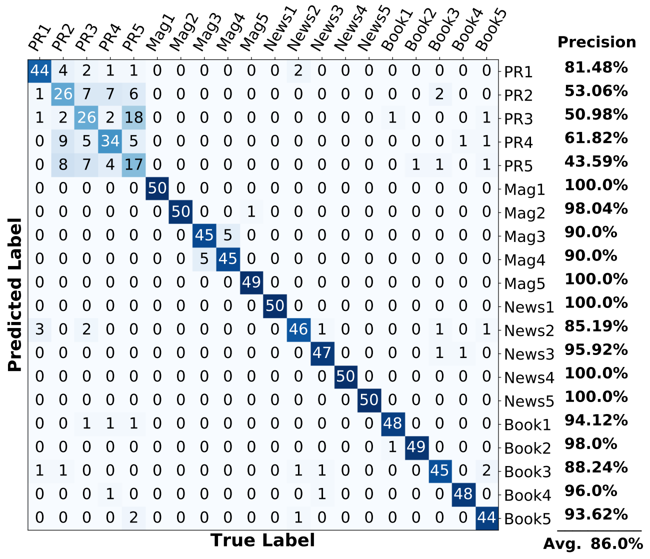
| | | F-Measure | Precision | Recall |
|---|---|---|---|---|
| **Intel** | NB | 98.7% (0.26) | 98.8% (0.19) | 98.7% (0.26) |
| | KNN-3 | 91.4% (3.53) | 91.9% (2.99) | 91.5% (3.51) |
| | RF | 99.6% (0.07) | 99.7% (0.06) | 99.7% (0.07) |
| | ResNet | 99.5% (1.09) | 99.5% (0.91) | 99.6% (1.11) |
| **Nvidia** | 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) |

## 2.3 Attack Scenario II: GUI Application Fingerprinting

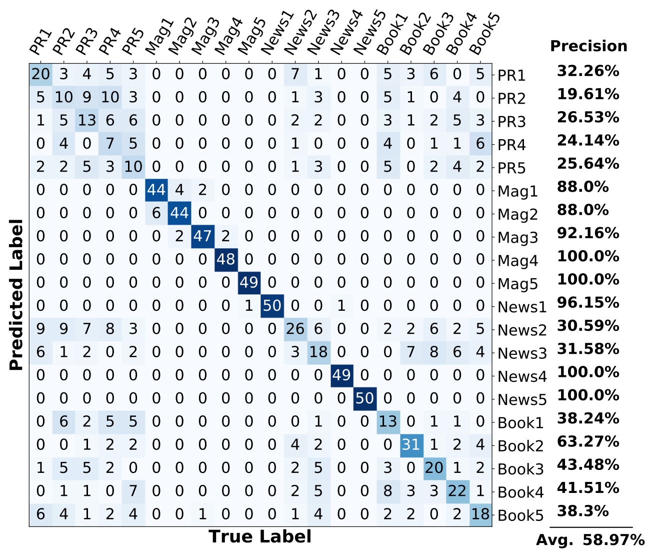
Our second attack attempts to fingerprint GUI applications with the same spy process monitoring graphics interrupts. Application fingerprinting has implications not only on revealing end user activities (e.g., which application is launched), but also on picking the best machine learning model for document fingerprinting. This is especially important because as shown in Section 2.2.1, different browsers cause different interrupt patterns even for displaying the same webpage. As such, with an effective application fingerprinting, it could be possible for an adversary to first identify the specific browser being used and then pick the suitable machine learning model to achieve optimized accuracy in subsequent webpage fingerprinting attacks.

**Experimental design** 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.

**Result** Our attack on application fingerprinting demonstrated very high accuracy with all tested machine learning models on both Nvidia and Intel GPUs — more than 99% for random forest and ResNet; see 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.



(a) on Intel HD Graphics 520



(b) on Nvidia GeForce GTX 760M

Fig. 4: Fingerprinting PDF files (research papers (RP), newspapers (News), magazines (Mag), and fiction books (Book).

## 2.4 Attack Scenario III: Beyond Visual Perception

In our attack scenarios I and II, document and application fingerprinting are both targeting objects that present unique visual perception to human users. The idea is that each unique GUI or view of the documents correspond to unique workload on the GPU, resulting in classifiable patterns of graphics interrupts. In this section, we investigate a more challenging problem in using aggregated graphics interrupts to differentiate objects with the same visual perception, i.e., can we differentiate something even a human being cannot differentiate with visual inspection? Such 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.

**Experimental design** 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[4] 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.

4. The video used can be found at https://shorturl.at/zCQTX

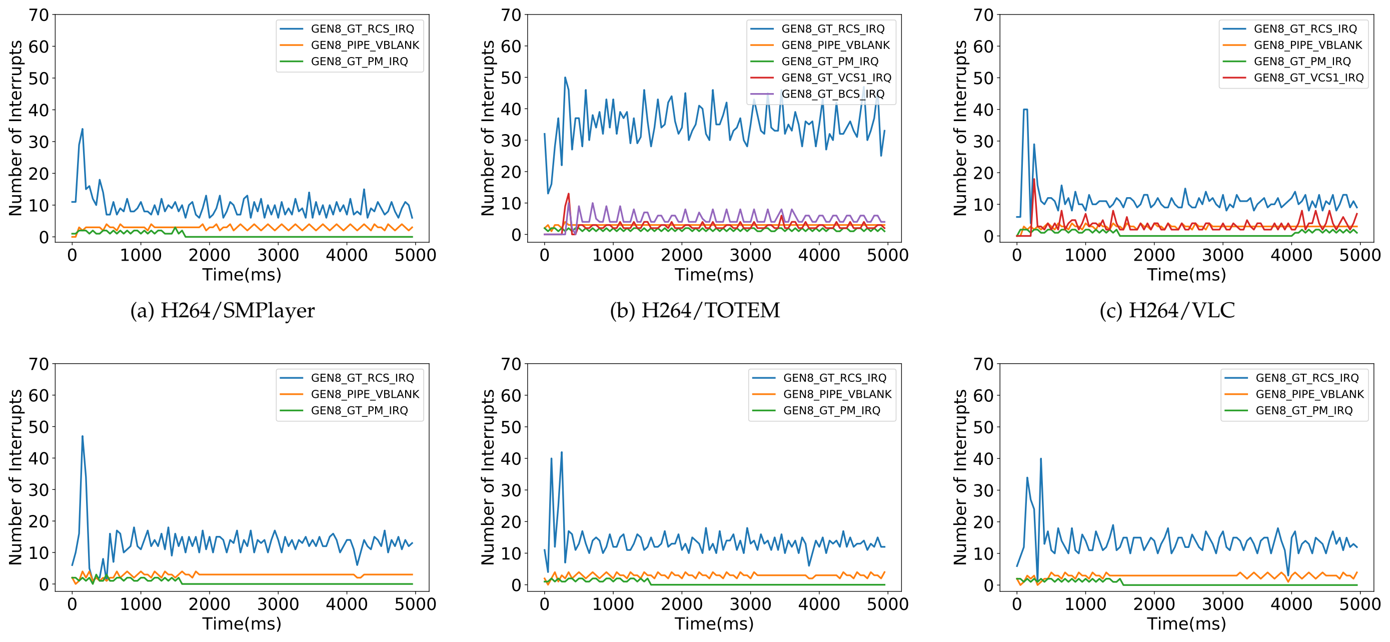
TABLE 5: Distinguishing video playback events: average and standard deviation.

| | | 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) |

**Result and analysis** Table 5 shows the performance of our attack in the two settings listed above. We find that in the scenario of distinguishing different video players, our attack works perfectly without a single misclassification. While in the scenario of distinguishing different codec, the attack on the Nvidia GPU outperforms that on the Intel (86.3% vs. 70.2%). Note that the classification here has only four potential classes due to the specific application scenario (i.e., we do not have 200 different video players or 200 different codecs to play with), and therefore the interpretation of results need to be adjusted accordingly. We remind readers that this is a much more difficult task than document or application fingerprinting, though, due to the lack of differences in the graphics display.

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 2.2). Figure 5 demonstrates such 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 that 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 interesting observations worth noting.

First, we find that different video player engines use different rendering techniques. Figure 5.a, 5.b, and 5.c show that when playing the same FLV video, `VLC`, `SMPlayer`, and



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

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

Fig. 5: Interrupt patterns (Intel) of playing the same video using different video players and codec. Missing lines correspond to zero readings of the IRQ types.

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 resorted 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, when playing the XVID, MPEG4, and WMV videos, VLC only involved the RCS engine with pure software-level decoding. To demonstrate how such implementation details affect the 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 6. 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.

## 2.5 Attack Scenario IV: Inferring Non-Graphical GPGPU Workload

In the last attack scenario, we move from eavesdropping visible events to fingerprinting program activities where GUI displaying is not involved. In this attack, the victim process is assumed to be executing some general-purpose computing workload on the GPU, while a spy process tries

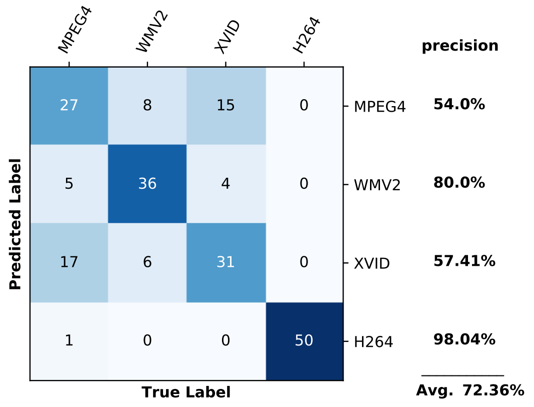
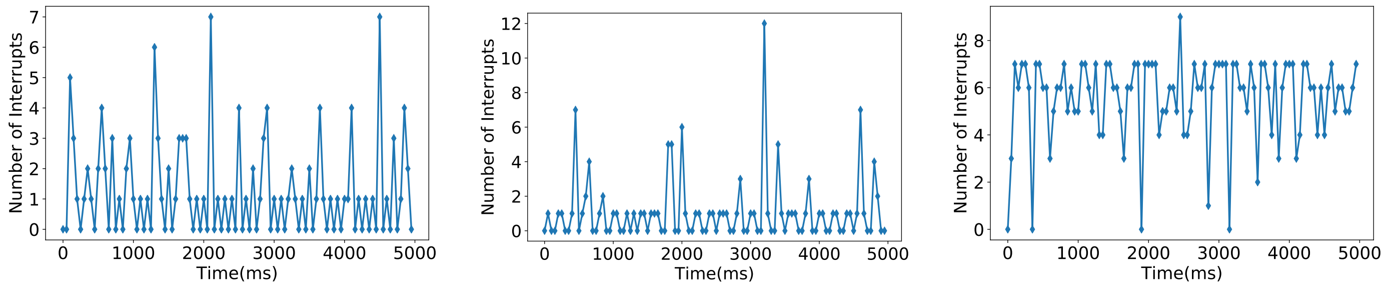


Fig. 6: Classifying (using ResNet) video playback of different codec using the same video player.

to identify the algorithmic type of that workload. General-purpose computing workloads on GPU may include sensitive executions such as cryptographic and machine learning algorithms. Therefore, though the attack considered here may not be fine-grained enough to directly reveal critical sensitive data such as the secret key for en/decryption, being able to identify the nature of GPGPU workload is still significant in that it could provide key information to subsequent attacks that cause more severe privacy leakage.

**Experimental design** For the test subjects of this attack scenario, we selected the CUDA-implementation of 3 well-known algorithms, namely AES[5], SHA-256[6], and the Py-

5. https://github.com/allenlee820202/Parallel-AES-Algorithm-using-CUDA.
6. https://github.com/Horkyze/CudaSHA256.



(a) AES (b) SHA-256 (c) ALEXNET

Fig. 7: Graphics interrupt patterns (Nvidia) caused by different GPU general-purpose computing payloads.

Torch[7] implementation of ALEXNET [13]. We applied the cryptographic algorithms on a randomly generated binary stream of 1.5MB on each run. As for the neural network algorithm, we performed network training on the MNIST training set[8] and let the process run for 1 minute on each run. Graphics interrupt counts were collected as in previous experiments, and we added the time series corresponding to these algorithms into the dataset of webpage and GUI application fingerprinting (see Section 2.2 and 2.3) for retraining the ResNet models. The reason of this experimental choice is that a spy process that is only capable of performing classification among a finite group of general-purpose computing algorithms has limited capability, since the dominating activities on an end-user device would most likely belong to attack scenarios given in one of the previous sections. Therefore, we believe that general-purpose computing identification is more meaningful if it can be integrated into the previous spy processes.

**Result** Surprisingly, we find that the retrained ResNet models could successfully identify each of the three general-purpose computing algorithms with 100% accuracy. Meanwhile the F-measure of the retrained models in identifying other GUI applications and webpages are 99.7% and 88.5%, respectively (which is consistent with results in the previous experiments).

As shown in Figure 7, each of the selected algorithms caused a unique pattern (which we find to be highly consistent across multiple executions of the same algorithm), and such patterns are also very different from those of graphical payloads such as webpages or application UIs. Unfortunately, because the Intel drm/i915 driver does not support general-purpose computing, we can only observe the aggregated interrupt patterns on the Nvidia GPU without further details. That said, this result still suggests that graphics interrupts could be a highly effective leakage source for inferring non-graphical (and executable) workload in GPUs.

## 2.6 Putting All Attacking Scenarios Together

With different results observed in the various attack scenarios, a question is thus raised: are these observations leading to a potentially consistent explanation? After analyzing underlying details of the graphical and non-graphical events being fingerprinted and inferred in all our simulations, we believe *a strong correlation exists between the accuracy of our attacks and the programmatic differences behind the targeted events*. More specifically, aggregated graphics interrupts are capable of capturing both the data plane and control plane of computer displays, where data plane refers to the graphical representation of the display (static objects in documents, videos, and application GUI) and control plane refers to program semantics responsible for the dynamic generation of the display (scripts within webpages, codec used in videos, executable code that generates application GUI, and general purpose GPU workload). Therefore, accuracy of our attacks is in fact influenced respectively by the presence of data and control planes.

With this high-level understanding, we go into our specific scenarios again to recognize their respective data-plane and control-plane involvement.

- In fingerprinting webpages, GUI applications, and non-Graphical GPGPU Workloads, the graphics interrupt patterns of the subject events are heavily affected by their corresponding execution routines (control plane).
- In recognizing video players and their codecs, there might be comparatively more control-plane involvement when identifying the different video players. We note that codecs also contribute to control-plane executions, although we argue that their determining effects on graphics-based interrupts are probably not as strong as the software implementation of different video players.
- Finally, the influence of programmatic activities in fingerprinting PDF documents is negligible due to our use of script-free PDF documents.

We now put together the average detection accuracy of all attacking scenarios and see if we could provide a potential explanation; see Figure 8. The results show that fingerprinting different video players is much easier than differentiating various codec used. Likewise, recognizing which PDF document is being opened results in much lower accuracy than telling apart different webpages.

This potentially suggests that graphics interrupts reveal more control-plane-related signatures of the targeted events than data-plane-related ones. This property could also make this side channel a useful tool in certain defensive scenarios of software/web security. To give an intuitive example, graphics interrupts might enable malicious webpage de-

7. https://pytorch.org/.
8. http://yann.lecun.com/exdb/mnist/

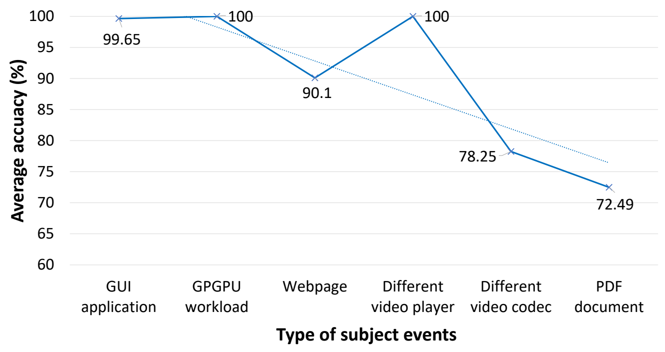


Fig. 8: An (illustrative) summary on the accuracy of our attacks against different types of subject events.

tection systems to overcome evasive techniques aiming to prohibit code analysis (e.g., obfuscation), since the side channel profiles a webpage's runtime behaviors. In our future work, we plan to carry out an in-depth study on using graphics-interrupt-based GPU side channel to support malicious webpage detection and other defensive security techniques.

Note that a systematic comparison among all the different attacking scenarios is unlikely to be fruitful considering other contributing factors, e.g., the number of possible classes in the classification experiments. Therefore, our analysis above is restricted to comparing the fingerprinting of PDF documents and webpages, or comparing identifying video player or codec used.

## 3 Defending Against Graphics-Interrupt-Based GPU Side Channel: A Mitigation

### 3.1 A Motivating Observation

As mentioned in Section 2.1.2, the graphics interrupt statistics retrieved in our attacks are aggregated measurements. As such, attacks based on such information could be interfered by other events which trigger screen refreshing or redrawing. A typical example of such noise sources is the movement of the mouse cursor, in which areas at the past and present locations of the cursor must be redrawn.

We first performed a preliminary evaluation on 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 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:

- **Separate model:** We train two ResNet models for the "noisy" and "clean" (free of noise) data, respectively. The application of such a strategy is under the assumption that the existence of noise (mouse movement or movie playing) could be effectively detected by the attacker (e.g., by observing mouse movement interrupts or by monitoring other side channels like CPU utilization); and
- **Mixed model:** We train only one model with both noisy and clean data mixed. Such classification will be useful when the existence of noise cannot be effectively detected.

We found that when classifying noisy data samples, the F-measure of both tested strategies decreased to only slightly over 54% for mouse movements as the noise and around 68% with video playing as the noise. Meanwhile, the F-measure of classifying clean data using the *Mixed model* is 3% less than that with *Separate models*. Figure 9 shows the interrupt patterns of loading Google's homepage and launching `Libreoffice` with and without mouse movements, demonstrating how the additional and continuous occurring interrupts caused by redrawing of the cursor had changed the patterns of the corresponding events.

### 3.2 Idea and Implementation

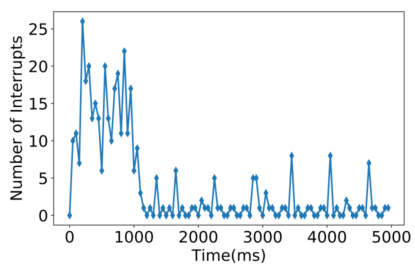
Motivated by the observation above, we propose a mitigation scheme to fight against privacy leakage via graphics-interrupt-based GPU side channel. Our main idea can be seen in Figure 10. Given an application carrying our mitigation module, whenever a privacy-related graphics/GPGPU event (henceforth we call this the *protected event*) is to be launched, our module actively generates an ultra-lightweight but highly randomized GPU payload, called the *mask payload*, and attaches it to the protected event. The mask payload is designed to cause only a negligible visual effect or no visual effect at all so that it has no substantial impact on the user experience of the protected event. Meanwhile, when under the observation of a spy process exploiting the graphics-interrupt side channel, the dynamic pattern of the protected event will be blended with that of the mask payload, hopefully to the extent that it is no longer identifiable.

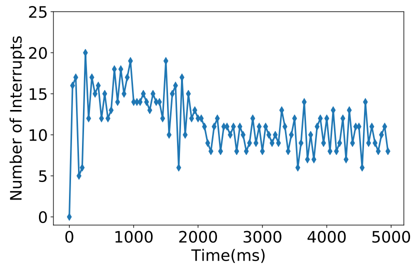
Implementing such a mitigation module is not hard:

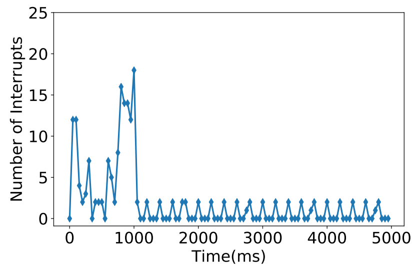
- For web browsers, the mitigation module can be built as a browser addon that actively embeds dynamic elements (in the form of small JavaScript pieces) into webpages upon opening them. This, rather than changing the side-channel signature of the application, masks that of the webpages displayed, instead.
- For GUI applications, the mitigation module can be built as a binary re-writing toolkit which, given a subject application, creates a series of dynamic displayable elements and embeds at least one of them into each of the application's GUI windows. When the rewritten application window is displayed, the embedded displayable elements cause GPU to launch extra IRQs and thus alter the application's overall graphics interrupt signature.

We have implemented a proof-of-concept demonstration of the browser-based side-channel mitigation module, which is built as a small `Chrome` addon[9]. When the Chrome browser opens a webpage, our addon embeds a piece of JavaScript payload into the HTML body to create a dynamic element which involves GPU operations, and runs this extra element together with the webpage. Currently, our `Chrome` browser addon is designed to create a single repeatedly

9. Our `Chrome` addon for the demonstration of GPU side-channel mitigation can be found at https://github.com/IanWE/PixelDefense


(a) Google homepage (clean)


(b) Google homepage (with mouse movement)


(c) `Libreoffice` (clean)

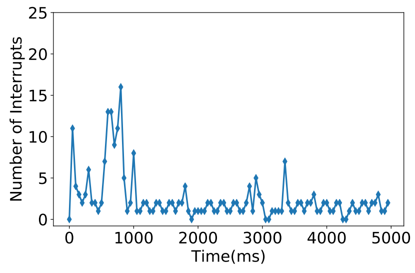

(d) `Libreoffice` (with mouse movement)

Fig. 9: Graphics interrupt patterns (Nvidia) with and without mouse movement noise.

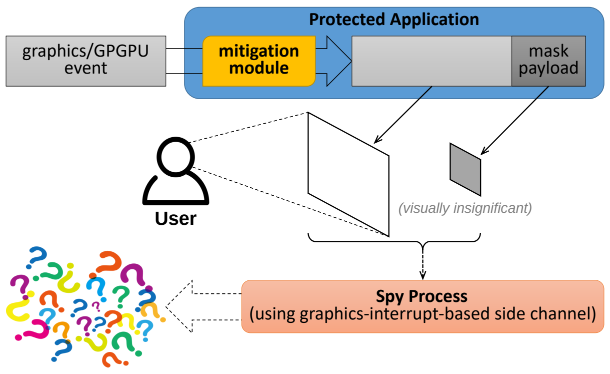


Fig. 10: General idea of our mitigation scheme.

redrawn pixel as its dynamic element (using no more than 5 lines of JavaScript code), which is deployed at the edge of the webpage opened such that the redrawing of this pixel can be kept active in all viewing modes of the webpage (e.g., windowed and full-screen). Given that the purpose is to mask the graphics interrupt signature of webpages, we avoid changing the color of the pixel in order to minimize the interference to the original visual effect of webpages. The embedded JavaScript operates iteratively according to a just-in-time configured refreshing rate, which is chosen uniformly random within the range of 1 to 100 ms.

Another potential option of implementing the dynamic elements is to use GPGPU workloads. However, our preliminary tests showed that using GPGPU workloads may not be an effective mitigation scheme because doing so caused a relatively heavy CPU occupation. The reason of this is that unlike the usual GPGPU cases which tend to batch processing large amount of computing in parallel, our mitigation module requires frequently sending small workloads into the GPU memory space and arranging operations on them, which then results in a large volume of CPU-GPU communications.

## 3.3 Evaluation

We conducted another round of webpage fingerprinting attack to demonstrate the effectiveness of our mitigation scheme on graphics-interrupt-based GPU side channel. We selected this specific attack scenario because manipulating webpages at a large scale can be done efficiently without causing any correctness issues. We believe that this setup does not undermine the generality of our experimental result or conclusion.

**Experimental design** The experiment with our `Chrome` addon is carried out on the same device as in all attack simulations demonstrated in this paper. Similar to what we did in Section 2.2.1, we tested our GPU side channel attack on the Alexa top 200 websites using ResNet as the attack-side classifier, given that this model resulted in the best performance in all tested machine learning models in the original attack simulation. Note that as a software-level defense scheme, we must consider a threat model where an adversary can obtain our side-channel mitigation toolkit and train her attacking deep-learning models with training samples embedded with our *mask payload*, hoping to gain as much adaptability against our mitigation as possible. Therefore, this time we configure each test webpage to be automatically loaded 200 times with our `Chrome` addon running at the background and embedding the pixel redrawing payload into them, creating "noisy" data only. Furthermore, to obtain better understanding of the effectiveness of our mitigation, we again tested both classification strategies mentioned in Section 3.1 — the *Separate model* and *Mixed model* strategies. For both strategies, we used data collected in the simulation of webpage fingerprinting attack (Section 2.2.1) as the "clean" portion of training data set, while the "noisy" portion consisted exclusively of data collected with our `Chrome` addon running. Like in all other experiments, we used 10-fold cross validation to measure the accuracy of the resulting models, except this time all test samples used in the validation were selected from the newly collected "noisy" data given that this would be the case in actual attacks intervened by our mitigation. Finally, we also measure the performance overhead caused by our `Chrome` addon in terms of CPU/power usage increment as well as extra GPU memory consumption.

**Result** As shown in Table 6, we found that with pixel redrawing events as the mask payload, our mitigation effectively reduces the effectiveness of graphics-interrupt-based webpage fingerprinting attacks. Specifically,

- the "clean" model trained for the *Separate model* strategy was completely put out of action, with its accuracy dropped to 1.63±0.12%;
- the "noisy" model of the *Separate model* strategy was also significantly compromised, which only achieved an accuracy of 43.3±6.53%;
- although the performance of the *Mixed model* strategy ended up slightly better, it still only achieved an accuracy of 56.69±4.73%.

Meanwhile, the performance overhead indexes showed that when working with the pixel redrawing payload, our `Chrome` addon induced only an additional CPU usage of 2.1% and an extra power consumption of 5W, while no observable GPU memory cost was produced. These together suggest that with proper randomization process, introducing intended GPU workloads to disguise user-privacy-related events could indeed be a viable strategy against the GPU side-channel attacks studied in this paper.

One additional observation we found particularly interesting is that, comparing the performances of the two strategies we have tested, it was actually shown that mixing "clean" and "noisy" data up could be a better training configuration for GPU side-channel attackers (when test samples are noisy) than using either type of data exclusively (although the resulting "improvement" is still not enough to make successful attacks). We are not able to reveal the exact reason behind this given that behavior of deep neural networks during training and classification is notoriously hard to explain. Our insight into this observation is that due to the generalization capability of neural networks, being able to learn both clear and consistent signature from the "clean" data as well as compromised patterns from the "noisy" data had made the resulting model more robust against the interference of our mitigation. That said, we also believe that the above observation actually further confirms the effectiveness of our mitigation scheme, because it suggests that it could be very hard for a GPU side-channel attacker to gain more advantage by adopting alternative strategies or extra tricks than moving to train a *Mixed model*. To give an example, an attacker may try to further add a clean/noisy feature to samples (which we did not do in our experiment) used in the training of a *Mixed model* with the suspicion that doing so could improve the resulting model. However, having such a binary feature would most likely end up emphasizing the correlation between samples within the "clean" and "noisy" groups, thus creating the same effect as using the *Separate model* strategy and leading to worse accuracy as the result.

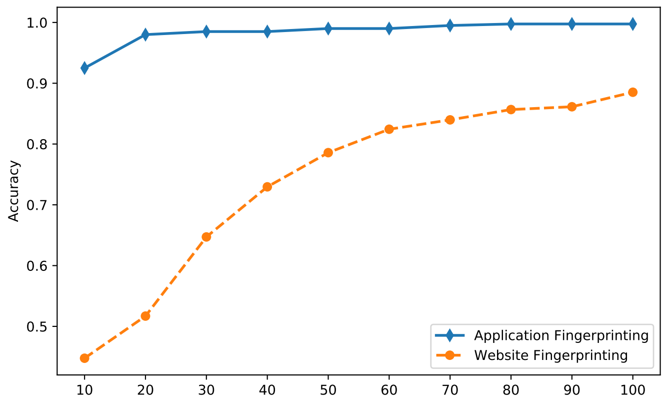


Fig. 11: Webpage and application fingerprinting with shorter interrupt time series.

## 4 Discussion

### 4.1 Tradeoff between Accuracy and Timeliness of The Graphics-Interrupt-Based Side Channel

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 2.1, 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 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 varied the length of subsequences with 10 different settings to train new machine learning models and observed the accuracy of them. Note that the sampling rate remains at 50ms to minimize workload of our spy process. As presented in Figure 11, 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 from the very beginning while loading webpages takes a slightly longer period. It also 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.

### 4.2 Side-Channel Effectiveness under Root Privilege

Recall in Section 2.1 that an unprivileged user process could only observe the aggregated reading of the different types of graphics interrupts. Should an adversary somehow manage to gain root privilege, his spy process would be able to analyze more detailed vectors of GPU interrupt statistics by hooking the system's kernel driver. This therefore raises

TABLE 6: Performance of webpage fingerprinting with our mitigation demo activated: average and standard deviation.

| | | F-Measure | Precision | Recall |
|---|---|---|---|---|
| **without mitigation** | | 88.2% (0.51) | 89.9% (0.31) | 88.3% (0.44) |
| **Separate model** | **"clean" model** | 1.63% (0.12) | 1.63% (0.08) | 1.64% (0.08) |
| | **"noisy" model** | 43.3% (6.53) | 43.3% (6.54) | 43.2% (5.92) |
| **Mixed model** | | 56.69% (4.73) | 56.68% (4.33) | 56.69% (4.55) |

a question: what advantage could root privilege bring to our attack? We admit here that an attack with root privilege could do more than monitoring interrupts, but still believe that it is an interesting question to assess the potential additional accuracy that could be gained. To understand this, we dissected the Intel interrupt data for webpage fingerprinting (see Section 2.2) into multi-dimensional time series consisting of separate readings of the specific interrupt types to see if the ResNet model trained with such vectors performs better. Compared to the result in Table 2, we found that F-measure of the new model only improved for about 1% on average. This suggests that when using a state-of-the-art deep learning model, the additional advantage from obtaining root privilege in our attack is negligible.

## 5 Related Work

### 5.1 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], [24]. The relationship between webpage loading and graphics displaying 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], [29]. Kotcher et al. [12] found that after applying CSS filters to a framed document, its rendering time becomes dependent on its content.

### 5.2 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. [27] 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 [28]. There were also researches suggesting that attackers could establish covert channels based on the CPU time used for handling interrupts [6], [18]. 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.

### 5.3 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 [32], TCP sequence numbers [26], and user identities [33].

## 6 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 possible attack scenarios in which graphics interrupt patterns were leveraged to respectively profile webpage opening, GUI application starting, and GUI tasks with the same graphics perception. On top of that, we further studied other potential attacks where graphics interrupts are exploited to infer program-independent non-executable GPU workloads as well as behaviors of GPGPU tasks that result in no visual presentation. 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. Finally, we proposed a practical mitigation scheme, which piggybacks GPU workloads with mask payloads that generates interfering graphics interrupt signals, such that signature of the original GPU workload can no longer be correctly observed and identified via the graphics-interrupt-based side channel. Using the scenario of webpage fingerprinting, we have demonstrated that the proposed mitigation strategy could effectively reduce accuracy of graphics-interrupt-based side-channel from 88.2% to 56.7%, while the additional performance overhead is negligible.

## Acknowledgment

We would like to express our appreciation to the anonymous reviewers and the associate editor for providing valuable feedbacks, which have helped a lot in improving this paper.

This work is supported by the Singapore National Research Foundation under the National Satellite of Excellence in Mobile Systems Security and Cloud Security (NRF2018NCR-NSOE004-0001), the National Key R&D Program of China(2018YFA0704703), the National Natural Science Foundation of China(61972215, 61972073), and the Natural Science Foundation of Tianjin(20JCZDJC00640).

## References

[1] Alexa, "The top 500 sites on the web," 2019. [Online]. Available: https://www.alexa.com/topsites

[2] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it: UI state inference and novel android attacks," in *Proc. of the 23rd USENIX Security Symposium*, 2014, pp. 1037–1052.

[3] W. Diao, X. Liu, Z. Li, and K. Zhang, "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*. IEEE, 2016, pp. 414–432.

[4] H. I. 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.

[5] E. W. Felten and M. A. Schneider, "Timing attacks on web privacy," in *Proc. of the 7th ACM conference on Computer and communications security*. ACM, 2000, pp. 25–32.

[6] R. Gay, H. Mantel, and H. Sudbrock, "An empirical bandwidth analysis of interrupt-related covert channels," *International Journal of Secure Software Engineering*, vol. 6, no. 2, pp. 1–22, 2015.

[7] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *Proc. of the 25th USENIX Security Symposium*, 2016, pp. 1187–1203.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[9] W.-M. W. Hwu, *GPU computing gems emerald edition*. Elsevier, 2011.

[10] inaz2, "Abusing interrupts for reliable windows kernel exploitation," 2015. [Online]. Available: https://www.slideshare.net/inaz2/abusing-interrupts-for-reliable-windows-kernel-exploitation-en

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

[12] R. Kotcher, Y. Pei, P. Jumde, and C. Jackson, "Cross-origin pixel stealing: timing attacks using css filters," in *Proc. of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 1055–1062.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of Advances in neural information processing systems 25*, 2012, pp. 1097–1105.

[14] S. Lee, Y. Kim, J. Kim, and J. Kim, "Stealing webpages rendered on your browser by exploiting gpu vulnerabilities," in *Proc. of the 2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 19–33.

[15] D. Lukan, "Hooking the system service dispatch table (ssdt)," 2014. [Online]. Available: https://resources.infosecinstitute.com/hooking-system-service-dispatch-table-ssdt

[16] C. Luo, Y. Fei, P. Luo, S. Mukherjee, and D. Kaeli, "Side channel power analysis of a gpu aes implementation," in *Proc. of the 2015 33rd IEEE International Conference on Computer Design*. IEEE, 2015, pp. 281–288.

[17] H. Ma, J. Tian, D. Gao, and C. Jia, "Walls have ears: Eavesdropping user behaviors via graphics-interrupt-based side channel," in *Proc. of the 23rd Information Security Conference*. Springer, 2020.

[18] H. Mantel and H. Sudbrock, "Comparing countermeasures against interrupt-related covert channels in an information-theoretic framework," in *Proc. of the 20th IEEE Computer Security Foundations Symposium*. IEEE, 2007, pp. 326–340.

[19] J. Miller and A. Osmani, "Rendering on the web," 2019. [Online]. Available: https://developers.google.com/web/updates/2019/02/rendering-on-the-web

[20] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on gpgpus," in *Proc. of the 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2017, pp. 354–366.

[21] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2139–2153.

[22] Nvidia, "Security notice: Nvidia response to "rendered insecure: Gpu side channel attacks are practical" - november 2018," 2018. [Online]. Available: https://shorturl.at/efJO6

[23] L. E. Olson, J. Power, M. D. Hill, and D. A. Wood, "Border control: Sandboxing accelerators," in *Proc. of the 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2015, pp. 470–481.

[24] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale," in *Proc. of the Network and Distributed System Security Symposium 2016*, 2016.

[25] R. D. Pietro, F. Lombardi, and A. Villani, "Cuda leaks: a detailed hack for cuda and a (partial) fix," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 1, p. 15, 2016.

[26] Z. Qian, Z. M. Mao, and Y. Xie, "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*. ACM, 2012, pp. 593–604.

[27] X. Tang, Y. Lin, D. Wu, and D. Gao, "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*. ACM, 2018, pp. 212–223.

[28] J. Van Bulck, F. Piessens, and R. Strackx, "Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic," in *Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 178–195.

[29] T. Van Goethem, W. Joosen, and N. Nikiforakis, "The clock is still ticking: Timing attacks in the modern web," in *Proc. of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1382–1393.

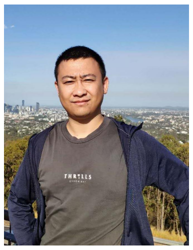
[30] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Proc. of the 2017 international joint conference on neural networks*. IEEE, 2017, pp. 1578–1585.

[31] Z. Yao, Z. Ma, Y. Liu, A. Amiri Sani, and A. Chandramowlishwaran, "Sugar: Secure gpu acceleration in web browsers," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 519–534.

[32] K. Zhang and X. Wang, "Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems." in *Proc. of the 18th USENIX Security Symposium*, vol. 20, 2009, p. 23.

[33] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, "Identity, location, disease and more: Inferring your secrets from android public resources," in *Proc. of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 1017–1028.

[34] Z. Zhou, W. Diao, X. Liu, Z. Li, K. Zhang, and R. Liu, "Vulnerable gpu memory management: towards recovering raw data from gpu," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 2, pp. 57–73, 2017.



**Haoyu Ma** received his Ph.D. degree in Computer Science and Technology from Nankai University in 2016. He joined School of Cyber Engineering, Xidian University in 2016, and is currently serving as a research engineer at Secure Mobile Centre, Singapore Management University. Haoyu focuses his research on operation system security as well as software security issues related to mobile computing and web applications.

**Jianwen Tian** received his master's degree in Computer Technology from Nankai University in 2019. He is currently a research engineer at Secure Mobile Centre, Singapore Management University.

**Debin Gao** is currently an Associate Professor from School of Computing and Information Systems, Singapore Management University. Having obtained his Ph.D degree from Carnegie Mellon University in 2006, Debin focuses his research on software and systems security. In recent years, Debin also actively participated in research of mobile security, cloud security, and human factors in security. Debin received the best paper award from NDSS in 2013.

**Chunfu Jia** got his Ph.D. in Engineering from Nankai University in 1996, and then finished his post doctoral research in University of Science and Technology of China. He is now a professor from College of Cyber Science, Nankai University, China. His current interests include computer system security, network security, trusted computing, malicious code analysis, etc.