Fantastic Interrupts and Where to Find Them: Exploiting Non-movable Interrupts on x86

Xin Zhang, Qingni Shen, Zhi Zhang, Yansong Gao, Jiajun Zou, Yi Yang, Zhonghai Wu

z ***Abstract*—While interrupts play a critical role in modern OSes, they have been exploited as a wide range of side channel attacks to break system confidentiality, such as keystroke interrupts, graphic interrupts and network interrupts. However, as previous attacks mainly focus on the exploitation of movable interrupts, they are required to determine which core is handling the target interrupts before their attack, which is non-trivial. The exploitability of non-movable interrupts, which cannot be reassigned by privileged softwares at will, remains unclear. In this paper, we conduct an empirical study on exploitable non-movable interrupts and their contribution to interrupt-based side-channel leakages in x86-based systems. We propose a dynamic analysis technique to investigate how various types of non-movable interrupts are influenced by different workloads. We then conduct a model fingerprinting attack as the benchmark to show that 7 types of non-movable interrupts are exploitable. To demonstrate the viability of these non-movable interrupts, we have created two concrete side channels, called ThermalScope and TimerScope. Specifically, ThermalScope exploits the thermal event interrupts that are triggered only when the CPU temperature exceeds a predetermined threshold, and TimerScope exploits timer interrupts that are activated regularly to enable the process schedule. Both techniques are adaptable to different attack scenarios, functioning regardless of whether the attacker and victim share the same core or reside on separate cores. Last, we successfully apply them to mount realistic case studies, ranging from constructing cross-core covert channels to breaking kernel address space layout randomization. We also demonstrate successful DNN model fingerprinting attacks under browser scenarios when the frequency scaling is disabled and attacker core is isolated from movable interrupts, where previous HertzBleed, ThermalBleed, and movable interrupt-based attacks are ineffective.**

***Index Terms*—Side channel attacks, Interrupts, x86 CPU, Computer architecture.**

I. Introduction

INTERRUPTS play a critical role in a modern operating system (OS), facilitating key functions such as efficient time-sharing, real-time processing, and exception handling. When an interrupt occurs, it triggers a context switch to the kernel, allowing the process scheduler to preempt the running process and execute the appropriate interrupt handler [1]–[4].

This work was supported by the National Key Research and Development Program of China (No.2022YFB2703301) and the Science and Technology Development Program of Two Districts in Xinjiang Province, China (No.2024LQ03004). Corresponding author: Q. Shen and Z. Zhang.

X. Zhang, Q. Shen, J. Zou, Y. Yang and Z. Wu are with the School of Software and Microelectronics, Peking University, Beijing 100871, China, also with the National Engineering Research Center for Software Engineering, Peking University, Beijing 100871, China, and also with the PKU-OCTA Laboratory for Blockchain and Privacy Computing, Peking University, Beijing 100871, China. E-mail: qingnishen@ss.pku.edu.cn.

Z. Zhang and Y. Gao are with the department of Computer Science and Software Engineering, University of Western Australia, Perth, WA 6009, Australia. E-mail: zzhangphd@gmail.com.

TABLE I
THE TAXONOMY OF X86-BASED INTERRUPT SIDE CHANNEL ATTACKS. OUR WORK, FOR THE FIRST TIME, CONDUCTS A SYSTEMATIC STUDY OF INFORMATION LEAKAGE INDUCED BY NON-MOVABLE INTERRUPTS.

	Interrupts	**Non-movable**
NDSS'17 [13], ESORICS'17 [12], ESORICS'23 [10], NDSS'24 [8]	Keystroke Interrupts	×
TDSC'21 [6]	Graphic Interrupts	×
SEC'22 [7], NDSS'24 [8], ESORICS'23 [10]	Network Interrupts	×
This Work	Thermal Event Interrupts, Timer Interrupts, etc	√

However, the execution of these handler functions leaves observable micro-architectural/architectural footprints that persist even after returning to userspace, making interrupts observable to an unprivileged attacker [5]–[10]. As interrupts are critical indicators of system activities, the attacker can infer the activities via observing interrupts.

In recent decades, various exploitable interrupts are identified, with attacks successfully compromising system confidentiality, as outlined in Table I. Specifically, keystroke interrupts are generated when a user presses keyboard. Since the interval between every two consecutive keystrokes correlates with user inputs, the inputs can be inferred via keystroke-based interrupts [11]–[13]. Besides, when GPUs process different workloads, they generate distinct patterns of graphic interrupts. With this observation, Ma et al. [6] exploit the graphic interrupts to identify different activities inside integrated/isolated GPUs. Last, network interrupts are triggered during the website visits. As different website access generates distinct network interrupts, they can be exploited to fingerprint websites [5], [7]–[10].

Limitations of State-of-the-art: However, all of the aforementioned interrupt side-channel attacks exploit *movable* interrupts, which can be dynamically reassigned to different CPU cores in a multi-core system and mitigated by software-level isolation. For instance, on x86-based Ubuntu systems, `irqbalance` can be enabled to dynamically balance the workload of interrupt processing across available cores. Besides, a privileged defender can also actively bind the movable interrupts via interrupt affinity. In such cases, the attacker has to first identify which core is handling the targeted interrupts. This can only be done by accessing */proc/interrupts* to monitor the interrupt count on the target core or by migrating their process to that core to observe any induced micro-architectural changes. However, The dynamic migration of interrupts between cores will reduce the accuracy of */proc/interrupts* sampling. Further, when */proc/interrupts* is disabled, it is non-trivial to consistently migrate the attack process to the interrupt-receiving core to observe interrupts via microarchitectural changes. This is because process scheduling is handled by the OS, and the unprivileged attacker lacks control over interrupt handling and core assignments.

In contrast, *non-movable* interrupts [14], which are bound to fixed cores at the hardware level and cannot be reassigned by the OS, have not been thoroughly investigated. To this end, we are interested in the following research questions:

Is there any non-movable interrupt that can be exploited for interrupt side channel attacks? If yes, what attacks can be mounted and what information can be leaked?

Our Work: In this paper, we conduct a system-level empirical study on non-movable interrupts in x86 and their potential as a side-channel. Specifically, we introduce a dynamic analysis technique that randomly switches between different computing workloads to investigate how various types of non-movable interrupts are influenced by these workloads. We then calculate the Point-Biserial correlation between each workload and the frequency of individual interrupts. Our results reveal 7 previously unreported exploitable interrupts.

To exploit the identified interrupts, we utilize these interrupts to fingerprint 36 models selected from 9 architecture families. Our experimental results show that model inference has an impact on each of the non-movable interrupts. Particularly, for either thermal event interrupts or timer interrupts, our fingerprinting attack has successfully achieved an accuracy of no less than 10 $\times$ of random guess.

Building on the dynamic analysis and fingerprinting results, we conclude that both thermal event interrupts and timer interrupts exhibit a strong correlation with CPU activity. Based on these findings, we build two effective side channels: ThermalScope and TimerScope. Both are non-movable interrupts identified in this paper, and we describe them below.

ThermalScope and TimerScope: ThermalScope leverages thermal event interrupts as a new side channel to monitor CPU temperature and ThermalScope alone has been published in the 61st Design Automation Conference (DAC) 2024 [15]. TimerScope exploits timer interrupts to observe the time of CPU being idle. Both of them fall within the scope of unprivileged exploitation of non-movable interrupts on x86-64 microarchitectures.

The key insight of ThermalScope is that workloads generate distinct heat signatures, which can be correlated with thermal event interrupts. When the CPU temperature exceeds a specific threshold during victim code execution, these interrupts serve as a side channel, potentially leaking sensitive information about the victim's activity. For TimerScope, it exploits another mechanism used for power optimization. We observe that timer interrupts are triggered and handled at fixed intervals when the CPU is active, but stop being triggered when the CPU enters an idle state to conserve power.

To demonstrate the viability of ThermalScope and TimerScope, we conduct three more case studies. First, we show that ThermalScope can establish a covert channel with a transmission rate of 0.1 b/s, while TimerScope-based covert channel achieves a transmission rate of 10 b/s. Second, we utilize both ThermalScope and TimerScope to derandomize kernel address-space layout randomization (KASLR). Our experimental results demonstrate that ThermalScope successfully identifies the correct offset in 8.2 hours, whereas TimerScope achieves this within just 1 minute. Lastly, we demonstrate successful DNN model fingerprinting attacks under browser scenarios when the frequency scaling is disabled and the attacker core is isolated from movable interrupts, where previous HertzBleed [16], ThermalBleed [17], and movable interrupt-based [8], [10], [14] attacks are ineffective.

Summary of Contributions: The contributions are summarized as follows:

- We conduct a comprehensive system-level empirical study on exploitable non-movable interrupts, significantly expanding the attacker's capabilities by eliminating the need to identify the specific core handling the target interrupts.
- We propose a dynamic analysis technique to investigate how various types of non-movable interrupts are influenced by different workloads. We then conduct a model fingerprinting attack as the benchmark to show that 7 types of non-movable interrupts are potentially exploitable.
- ThermalScope exploits the thermal event interrupts and can be used to infer CPU temperature (published in DAC 2024 [15]). For TimerScope, it exploits the timer interrupts and is used to infer CPU idle time. Both can bypass the aforementioned limitations of movable interrupts.

The remainder of this paper is organized as follows. Section II presents background on interrupt side channels, model fingerprinting, and non-movable interrupts. Section III covers our threat model, the design of dynamic analysis techniques and the identifying results. Section IV further demonstrates two side channels via thermal event interrupts and timer interrupts. Section V evaluates them with two case studies. Section VI discusses possible countermeasures and we conclude in Section VII.

II. Background and Related Work

A. Interrupt Side Channel Attack

Interrupts play a critical role as hardware resources in a modern operating system (OS), enabling OS process scheduler to preempt a running process and execute a corresponding interrupt handler [1]–[4]. As different interrupts are activated when the system handles different workloads, existing interrupt side channel attacks have exploited various types of interrupts, such as keystroke interrupts [11]–[13], graphic interrupts [6], and network interrupts [7], [14], to break system confidentiality.

First, when a victim presses the keyboard to input her secret, keystroke interrupts are activated to invoke the corresponding interrupt handler. Because the context needs to switch into kernel, the timestamp observed by users will become discontinuous. An unprivileged attacker can use the timing side channel to monitor the keystroke interrupts to infer the victim's input [8], [10]–[12]. Besides, when the corresponding interrupt handler code is loaded, the cache state changes. With this observation, KeyDrown [13] proposes to use cache side channel to detect the memory accesses to the interrupt handler code, thereby monitoring keystroke interrupts. To find targeted cache sets, they need the physical addresses of the keystroke interrupt handler.

Second, graphic interrupts are activated by a GPU when there is a need to handle specific tasks related to graphics processing, indicating significant events such as the completion of a graphics command or reporting a hardware error. The timing interval between two consecutive graphic interrupts varies depending on the specific workloads being processed by the GPU. Ma et al. [6] exploit graphic interrupts as a separate side channel to leak the activities inside the integrated and isolated GPUs. They successfully mount several side-channel attacks under various scenarios, including fingerprinting documents, distinguishing applications, and recognizing non-GUI applications.

Last, the activation of network interrupts is related to network activities. For example, when data packets arrive at the network interface, network interrupts need to be activated to handle the reception and processing of the incoming data. Different websites can trigger different network activities, including distinctive network interrupts. An attacker can use these interrupts to fingerprint websites [7], [10], [14], whose attack typically consists of two phases: an *offline preparation phase* and an *online classification phase*. In the *offline phase*, the attacker collects a number of interrupt traces and utilizes them to train a classifier. In the *online phase*, while the victim is opening a website, the attacker employs the pre-trained classifier to fingerprint which website the victim is visiting.

Movable Interrupts: We summarize that all existing interrupt-based side-channel attacks rely on movable interrupts, which face two key limitations in practice. First, modern operating systems (such as x86-based Ubuntu systems) support interrupt load balancing, which frequently randomizes the assignment of movable interrupts across cores by enabling `irqbalance` [18], making it difficult for attackers to reliably infer or consistently co-locate on the interrupt-receiving core. A privileged defender can also bind the targeted movable interrupts on a different core with the attacker core via interrupt affinity [8], [9], [14]. Second, because both interrupt delivery and the `/proc/interrupts` interface are managed at the OS-level, if the victim and attacker are located in different virtual machines (VMs) of the same hardware, all previous attack will not work at all, as both the */proc/interrupts* and co-location of interrupt-receiving core are not feasible, except in special cases like KVM-based virtualization [8].

Compared to the above works, both ThermalScope and TimerScope remain effective under the first scenario where interrupt load balancing or interrupt affinity obfuscates movable interrupt behavior. Furthermore, because thermal event interrupts are triggered at the hardware level across all cores, ThermalScope can potentially bypass OS-level isolation, extending the attack surface to cross-VM settings.

B. Model Fingerprinting Attack

Given the considerable investment in developing DNNs, they are treated as valuable intellectual property by AI providers, who often monetize these models through licensing [19], query-based fees [20], or deployment on edge devices [21], [22]. This business model is threatened by model extraction attacks, where adversaries aim to steal the DNN models [23], [24]. Such attacks not only lead to direct economic losses by circumventing monetization strategies and enabling unauthorized distribution but also pose security risks. Stolen models can be used for further malicious activities such as model inversion attack [25]–[27] that recovers sensitive training data, or adversarial attack [28], [29] that tricks the model to produce incorrect outputs, thereby raising significant privacy, ethical, and legal concerns.

To achieve this attack goal, side-channel attacks have become a potent means of determining the architecture of DNN models by exploiting indirect information leaks during model execution [30]–[33]. These attacks monitor various side-channel signals that occur during the model inference phase, including cache usage patterns [34], memory access sequences [35], [36], power consumption profiles [33], and electromagnetic emissions [22], [31]. Since DNNs involves complex matrix multiplication operations inherent in neural network computations, it results in different types of leakages in these side channels. By analyzing these signals, an adversary can infer the sequence of computational operations, deduce layer types, and extract architectural parameters without direct access to the model's internal structure.

One of the prominent techniques is side-channel assisted model fingerprinting, which utilizes side-channel data to identify a DNN model by matching it against a pre-established database of known architectures [37]–[39]. This attack operates under a closed-world assumption, where the victim's model is selected from a well-known set of architectures. The attacker formulates the attack as a classification problem, training a classifier on side-channel traces gathered from the known models. During the online phase, the attacker collects side-channel traces from the victim's model and uses them to classify and identify the architecture. Although this method may face challenges when distinguishing between architectures with similar characteristics, model fingerprinting remains a significant threat by allowing attackers to accurately classify and identify models from popular libraries. In practice, victims often select models from widely used DNN libraries [37], making this attack highly feasible in real-world scenarios.

C. Non-movable Interrupts

Non-movable interrupts are interrupts that are dedicated to handle critical system events and cannot be reassigned to different CPU cores during execution [14], [40], [41]. This design ensures reliable and timely handling of critical events, such as hardware timer interrupts and thermal event interrupts, which require consistent, predictable processing. In contrast to movable interrupts, which can be managed by one or more cores through privileged software configurations, non-movable interrupts have fixed core assignments and cannot benefit from CPU load balancing.

Figure 1 illustrates a partial snapshot of interrupt statistics from `/proc/interrupts`. Movable interrupts are identified by numeric identifiers, followed by the name of the corresponding device responsible for generating the interrupt

```
name       Core0     Core1     Core2   ...   Core10    Core11
   1:         92         0         0   ...       64         0  IR-IO-APIC    1-edge      i8042
   8:          0         0         0   ...        0         0  IR-IO-APIC    8-edge      rtc0
   9:       1784       387         0   ...        0         0  IR-IO-APIC    9-fasteoi   acpi
  17:       7467     10878         0   ...       50      3342  IR-IO-APIC   17-fasteoi   idma64.1, i2c_designware.1
  51:     108226         0     78802   ...        0         0  IR-IO-APIC   51-fasteoi   MSFT0001:02
 162:         14         0         0   ...        0         0  IR-PCI-MSI 56623104-edge      rtsx_pci
 163:      24191         0      9181   ...   104478     87088  IR-PCI-MSI 57147392-edge      iwlwifi:default_queue
 177:         10         7         0   ...        0         0  IR-PCI-MSI 57147405-edge      iwlwifi:exception
 178:          0         0      2048   ...        0         0  IR-PCI-MSI 514048-edge      snd_hda_intel:card0
 NMI:         38      2951       320   ...      104       137  Non-maskable interrupts
 LOC:    1494092   4660484   1361822   ...  1079992    412680  Local timer interrupts
 SPU:          0         0         0   ...        0         0  Spurious interrupts
 PMI:         38      2951       320   ...      104       137  Performance monitoring interrupts
 IWI:      51545     15687     17677   ...    16880     13212  IRQ work interrupts
 RTR:          0         0         0   ...        0         0  APIC ICR read retries
 RES:       9782      6951      7160   ...     6412      7260  Rescheduling interrupts
 CAL:     204497    107199     95661   ...    87191     82569  Function call interrupts
 TLB:      29054     26963     25718   ...    22560     18894  TLB shootdowns
 TRM:    6320738   6320742   6320741   ...  6320739   6320741  Thermal event interrupts
```

Fig. 1. An example of interrupt statistics provided by `/proc/interrupts`. The movable interrupts, identified by numerical values, are consistently associated with the type of external devices (e.g., i8042 or rtc0). On the other hand, the non-movable interrupts, denoted by abbreviations (i.e., NMI, LOC, etc), are simply described based on their function. Note that some entries have been omitted to save space.

(e.g., network cards or storage controllers). In contrast, non-movable interrupts are labeled with descriptive names (e.g., local timer interrupts or thermal event interrupts) and are triggered by corresponding hardware events specific to a given core. These events must be handled on the same core where they originated, as their nature is tied to the core's function. To further elucidate the working principle of non-movable interrupts, we introduce thermal event interrupts and timer interrupts as two pertinent examples.

Thermal Event Interrupts: In x86 CPUs, thermal throttling is implemented through a package-level thermal sensor (MSR 0x19c), which monitors the overall temperature of the CPU package [40]. When the sensor detects that the package temperature reaches a pre-determined threshold, thermal event interrupts are triggered across all CPU cores, initiating context switches to OS kernel. As a response, the kernel logs the thermal event and takes actions by adjusting cooling devices (e.g., increasing the fan speed, reducing the CPU frequency, etc). To mitigate the risk of hardware failure due to excessive heat generated by high-intensity workloads in userspace, it is crucial for the CPU to distribute interrupts across all cores. Consequently, thermal event interrupts do not support interrupt affinity, which would otherwise direct specific interrupts to designated cores.

Timer Interrupts: Timer interrupts are managed by the Advanced Programmable Interrupt Controller (APIC) on each CPU core, enabling the operating system to invoke kernel operations at fixed intervals [40], [42]–[44]. Thus, when there is a need to reschedule one or two processes on a specific core, the timer interrupts must be activated and cannot be assigned to another core. Additionally, to minimize inefficiencies, as many timer interrupts are unnecessary when no tasks require attention, the APIC hardware allows users to configure the interrupt generation policy. By default, timer interrupts are only triggered when the CPU has active tasks to process. This configuration reduces unnecessary interrupts during idle periods, thereby optimizing power consumption.

III. Identifying Exploitable Non-movable Interrupts

In this section, we first present the threat model and assumptions. Second, we propose a dynamic analysis technique to investigate how various types of non-movable interrupts are influenced by different workloads. Last, we conduct a model fingerprinting attack as the benchmark to show that some non-movable interrupts are indeed exploitable.

A. Threat Model and Experimental Setup

In our threat model, we assume an unprivileged attacker. Thus, the attacker cannot make any modifications to a victim x86-based system. In the *native* scenario, the attacker can access local resources (e.g., system interfaces). In the *browser* scenario, the attacker is allowed to run their code in a sandbox.

To fingerprint DNN model architectures, we assume that the victim selects models from PyTorch vision DNN architectures[1]. To build covert channels, two user processes collude with each other. To break KASLR, the victim system does not have any software bugs or vulnerabilities that enable the attacker to acquire a mapped kernel address.

Machine settings: Our evaluation is conducted under three different systems as shown in Table II, including different Intel-based CPUs. Unless otherwise stated, we use the default system configuration.

TABLE II
System configurations.

Machine	CPU	Kernel	OS
Xiaomi Air 13.3	Intel Core i5-8250U	5.15.0	Ubuntu 20.04.5
Lenovo Savior R9000	Intel Core i7-9750H	5.8.0	Ubuntu 22.04.1
Supermicro X10DRG (Server)	Intel Xeon E5-2680 v4	5.15.0	Ubuntu 22.04.1

[1]https://github.com/pytorch/vision

B. Dynamic Analysis Technique

Here, we propose a dynamic analysis technique to investigate if there are any interrupts having a strong correlation with the CPU activities rather than specific external device. To this end, we rely on `stress-ng` [45] to create various activities and then calculate the correlation between each type of interrupts and each CPU state. Specifically, we alternate the CPU between high-load and idle states by controlling the CPU's operational states at fixed predetermined time intervals. For the idle states, we utilize the `sleep` operation across all CPU cores, except for the attack core. In the high-load state, we employ the following tasks:

- **Computation**: This setting induces CPU load by performing square root computations on randomly generated numbers.
- **L1-Cache Hit**: This setting specifically targets the L1 cache by having all active cores access data that entirely fits within the L1 cache size (i.e., 32 KB).
- **LLC Hit**: The setting continuously accesses data that fits within the Last-level cache (LLC), which is much larger than the L1 cache, thereby enforcing cache lookups and accesses at this level. This stress test ensures that the LLC, shared among multiple cores, is highly utilized.
- **Memory Read**: This setting emphasizes memory read operations by allocating 1 GB of memory and directing each active core to read 64-bit values from it.
- **Memory Write**: This setting focuses on memory write operations. Each active core is allocated 1 GB of memory and continuously writes 64-bit values to it.
- **Disk I/O**: This setting generates disk I/O stress by executing synchronous write operations on the hard disk. Each active core performs I/O operations concurrently, simulating a high-load disk scenario.

To investigate the variations in different types of interrupts under varying conditions, we run each setting separately. In each setting, we randomly choose the running states between a high-load and idle states every 30 s and record the number of interrupts generated at both the beginning and the end of each iteration, thereby establishing a foundation for subsequent analysis. We repeat the above procedure 1,000 times for each setting. Following data collection, we concentrate on evaluating the relationship between interrupt behavior and changes in running states.

We employ two metrics to analyze the correlation between different interrupt types and CPU activity, aiming to identify interrupt categories that may be influenced by CPU operations. First, we calculate the average increment of each type of interrupt when different workloads are activated. This serves as an indicator of observability, as a higher frequency of interrupts enhances their detectability. Second, we compute the Point-Biserial Correlation Coefficient [46] for each setting. This coefficient is well-suited for this analysis as it measures the relationship between a binary variable, representing the running state (idle or active), and a continuous variable, the number of interrupts observed. These correlation coefficients range from [-1, 1], with absolute values close to 1 indicating a strong correlation. As we randomly alternate CPU workloads between idle and a specific active state for each setting, this coefficient provides a quantitative measure of the extent of information leakage by determining how well interrupt patterns can differentiate between the two states. A higher correlation indicates a stronger association between workloads and specific interrupt behavior, thereby identifying potential information leakage risks.

Experimental Results: Table III lists the results of 7 non-movable interrupts that have correlation with the tested activities. Note that some types of interrupts are ignored because their number is always 0 during the experiments. Besides, we bolded the values with correlations exceeding 0.9 to highlight strong correlations.

First, our experiments demonstrate that certain types of non-movable interrupts can be reliably triggered by specific system activities. For instance, thermal event interrupts are activated in response to high-load computations, L1 cache reads, memory reads, and memory writes. However, these interrupts are not triggered by disk I/O, even when the same CPU cores are stressed during these operations. Besides, even interrupts that show only a small number of activations can still reflect important system activities. For example, performance monitoring interrupts (PMIs) and non-maskable interrupts (NMIs), which often occur together, are instrumental in tracking CPU performance metrics such as instruction counts, cache misses, and branch prediction failures. Linux kernel enables `nmi_watchdog` by default, which tracks the instruction counts that overflow periodically and trigger PMIs in non-maskable mode. During each PMI, kernel will check a high-resolution timer (hrtimer) to avoid hard lockups. Since these PMIs are configured to be non-maskable and no other NMIs are generated, we consistently observe the same counts for NMIs and PMIs across all tested machines.

Observation 1: Some non-movable interrupts are closely tied to specific system activities and thus potentially exploitable.

Second, we emphasize that the mere presence of a large number of interrupts does not necessarily indicate exploitability. For example, we observed a significant number of TLB shootdowns and function call interrupts during copy-on-write operations. However, TLB shootdowns depend on the need to create new TLB entries, and thus, these interrupts only occur at the beginning of the test. This illustrates that the timing and frequency of interrupts must be carefully considered when assessing their exploitability. Moreover, we found that most interrupts are bound to individual logical cores, while thermal event interrupts are synchronized across all cores. Therefore, understanding the behavior and characteristics of each type of non-movable interrupt is critical before leveraging them in an attack.

Observation 2: Different non-movable interrupts exhibit varying levels of observability and responsiveness to system activities. Their behavior must be carefully analyzed to assess exploitability.

TABLE III
RESULTS OF OUR DYNAMIC ANALYSIS. WE SHOW THE AVERAGE OF EACH TYPE OF INTERRUPTS UNDER ALL RUNNING STATES AND MARK THE CORRELATION IN THE BRACKETS.

Type	Idle	Computation	L1-Cache Hit	LLC Hit	Memory Read	Memory Write	Disk I/O
Thermal Event Interrupts	0	**10338.8 (0.974)**	**7629.4(0.979)**	420.7(0.678)	4639.7(0.876)	**3750.7(0.977)**	0
Local Timer Interrupts	0	**7355.9 (0.979)**	**7270.3 (0.974)**	**3602.2 (0.948)**	**7302.6(0.976)**	**7303.9(0.951)**	3551.3(0.878)
Non-maskable Interrupts	0	**3.77 (0.961)**	**2.5 (0.948)**	**2.2 (0.949)**	**4.1 (0.972)**	**4.1 (0.971)**	0.08(0.212)
Rescheduling Interrupts	0.40	8.1(0.850)	**143.1 (0.974)**	9.8(0.751)	2.4(0.657)	241.3(0.323)	1.25(0.387)
TLB Shootdowns	0.05	0	0.14(0.134)	0.2(0.105)	0	22711.7(0.308)	0.11(0.176)
Function Call Interrupts	0	6.1(0.102)	3.5(0.276)	0.2(0.111)	0.3(0.007)	22219.9(0.308)	651.9(0.862)
Performance Monitoring Interrupts	0	**3.77 (0.961)**	**2.5 (0.948)**	**2.2 (0.949)**	**4.1 (0.972)**	**4.1 (0.971)**	0.08(0.212)

C. Model Fingerprinting Attack

To evaluate the information leakage of non-movable interrupts, we perform model fingerprinting attack as a significant benchmark. Our key insight is that a number of matrix operations used by a DNN model inference result in a distinctive pattern of system activity and then trigger different non-movable interrupts. We can use their fingerprinting results to quantize the leakages.

Aligned with previous work [33], [37]–[39], [47], our model fingerprinting attack has two phases: an *offline preparation* phase and an *online classification* phase. In the *offline preparation* phase, we collect the non-movable interrupt traces to build a series of well-trained classifiers, which can translate a trace to its corresponding model architecture. In the *online classification* phase, we query a black-box target model running on the victim machine to trigger its model inference and collect our side channel traces. Then we use the offline-trained classifiers to fingerprint the model architectures.

Since the fingerprinting attack is essentially a 36-class classification task, we choose 4 classic ML models as classifiers for the attack, aligned with previous fingerprinting attacks [39], [48], including Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RForest), and k-Nearest Neighbor (k-NN). We do not deploy deep neural networks (e.g., BERT) as the classifier by the need for interpretability, computational efficiency, and reduced training data requirements. For the SVM classifier, we choose a linear kernel function, and the soft margin constant is set to 10. For LR classifier, the range of penalty parameter C is 1, 10, 100, 1000. For the RForest classifier, we set the number of trees as 100 and the maximum depth as 32. For the k-NN classifier, we set the number of nearest neighbors as 10. For each classifier, we perform the 10-fold cross-validation during the evaluation, where nine folds are used as the training data, and the remaining one fold is retained as the testing data.

Experimental Setup: We carry out our experiments in all the tested machines. The victim model uses PyTorch 1.13.0 with Python version 3.9.13 as its underlying deep learning framework. The attacker process and the victim process run on separate physical cores. Aligned with [37], [39], the victim models are selected from the PyTorch vision DNN architectures, including 36 architectures over 9 diverse architecture families. All the pretrained models expect the input images to be $224 \times 224 \times 3$ and normalized in the same way. We

TABLE IV
THE 36 PYTORCH VISION ARCHITECTURES WHICH WERE USED IN EVALUATING OUR FINGERPRINTING ATTACK.

Model Family	Model Architecture
VGG	VGG11, VGG13, VGG16, VGG19 VGG11_bn, VGG13_bn, VGG16_bn, VGG19_bn
ResNet	ResNet18, ResNet34, ResNet50, ResNet101, ResNet152 Wide_ResNet50_2, Wide_ResNet101_2, ResNext50_32x4d, ResNext101_32x8d
SqueezeNet	SqueezeNet1_0, SqueezeNet1_1
DenseNet	DenseNet121, DenseNet161, DenseNet 169, DenseNet201
ShuffleNet	ShuffleNet_v2_x0_5, ShuffleNet_v2_x1_0 ShuffleNet_v2_x1_5, ShuffleNet_v2_x2_0
MnasNet	MnasNet0_5, MnasNet0_75, MnasNet1_0, MnasNet1_3
MobileNet	MobileNet_v2, MobileNet_v3_large, MobileNet_v3_small
AlexNet	AlexNet
GoogleNet	GoogleNet

run inference on the ImageNet ILSVRC Test set images. The victim runs a DNN model in series for 20 seconds. [2] We pin the attack process and the victim process to separate cores to avoid scheduling contention.

We sequentially sample the increment of each type of non-movable interrupts from `/proc/interrupts` by default at a fixed interval of 50 ms and collect 1,000 points for each attack. For each setting, we record 100 traces for each of the 36 model architectures. We feed the collected side channel traces without any preprocessing into the classifiers.

Experimental Results: Figure 2 shows the fingerprinting results of 7 different non-movable interrupts, with the best classifier for each interface highlighted. Among the 7 identified interfaces, only thermal event interrupts and timer interrupts can achieve an accuracy of no less than 10 $\times$ of random guess (i.e., 27%), indicating a strong correlation between sampled interrupt counts and model inference activities. Besides, the results of thermal event interrupts achieve a high accuracy of over even 90%. This is reasonable because DNN model inferences would create heat, thereby activating distinctive thermal event interrupts. We further investigate their exploitation in Section IV.

IV. THERMALSCOPE AND TIMERSCOPE

In this section, we propose two instances that respectively exploit thermal event interrupts and timer interrupts as a standalone side channel, followed by their characterization.

[2]Aligned with [39], we run the inferences in series (not in a batch) because an attacker would not have control over the batch size input to the victim model.

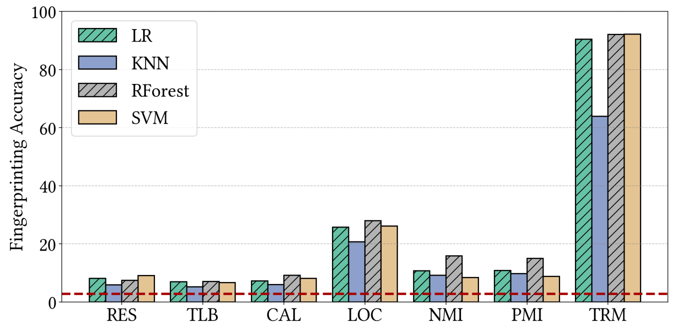

Fig. 2. The fingerprinting accuracy of the fingerprintng results under different settings. The red dashed line denotes a baseline of random guess (i.e., 2.7%).

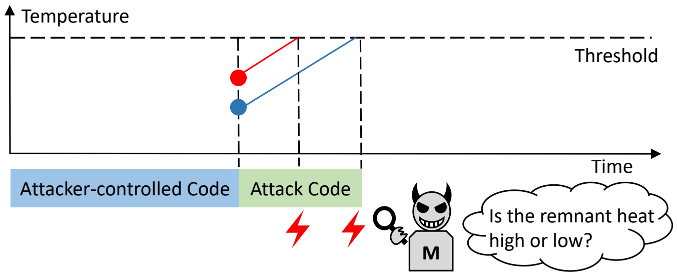

Fig. 4. An illustration of using *ThermalScope-Probe* to exploit thermal event interrupts. Two pieces of code are executed sequentially on the same core. The second piece intentionally generates heat until a thermal event interrupt is triggered, whose execution time is used to observe temperature.

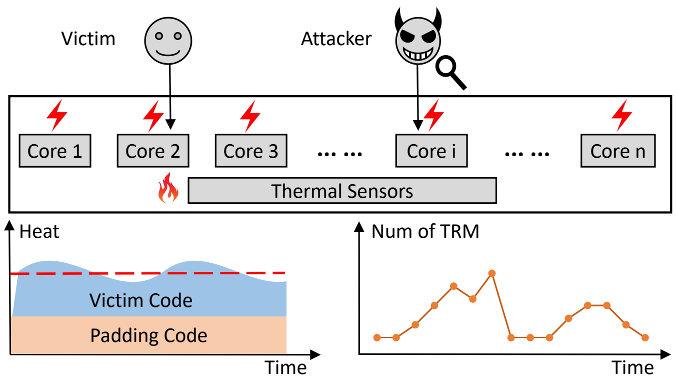

Fig. 3. An illustration of using *ThermalScope-Monitor* to exploit thermal event interrupts (TRM). As thermal event interrupts are consistently activated on all cores, the attacker can obtain temperature information without relying on direct access to the thermal sensor on an arbitrary core.

A. ThermalScope

As introduced in Section II-C, the thermal event interrupts are triggered only when CPU temperature exceeds a predetermined threshold. We aim to fully exploit the potential of thermal event interrupts to build ThermalScope. Based on the attacker's core configuration, ThermalScope consists of two variants for observing temperature changes. The first variant, *ThermalScope-Monitor*, monitors temperature changes on a victim core different from the attacker's core. The second variant, *ThermalScope-Probe*, observes temperature fluctuations on the same core as the attacker, where a number of sensitive instructions have been executed to induce temperature variations correlated with the secret data.

ThermalScope-Monitor: On a multi-core system, the CPU package temperature is determined jointly by the workloads of all cores. To amplify the temperature increments, the attacker can execute another compute-intensive code (called *padding code*) concurrently with the victim code. The padding code loops to execute the same instructions to generate constant heat, which is used to lift the package temperature to exceed the threshold to activate thermal event interrupts. Because the heat caused by the padding code is stable, the variation of thermal event interrupts can be attributed to the victim code. In this way, an attacker can infer victim activity by observing the induced thermal event interrupts.

As shown in Figure 3, to construct *ThermalScope-Monitor*, the attacker needs to run her padding code on a different core with the one occupied by the victim. By selecting an appropriate padding code, the package temperature is amplified near the threshold, causing distinctive thermal event interrupts during the execution of the victim program. They can infer the activity of the victim code via the activation of thermal event interrupts.

ThermalScope-Probe: Since the temperature increments caused by the computing task can be observed even after it is stopped, these increments can expose secret information to the following processes, especially when they share the same physical core [49]. This remnant heat allows the process to infer sensitive information from its predecessor, thereby violating time division. Unlikely other types of shared system resources that can be reset before context switching (e.g., CPU registers), as the decrease in temperature requires a significant amount of time, the remnant heat is hard to eliminate.

In this section, we leverage the remnant heat to construct *ThermalScope-Probe*, a new variant of ThermalScope that obtains temperature observations by triggering featured thermal event interrupts. To achieve this, the attacker is required to execute their code on the same CPU core as the victim's code. As shown in Figure 4, two pieces of code are executed sequentially. One piece is attacker-controlled code (e.g., a code snippet that accesses a kernel address in breaking KASLR) that executes a computational task related to user's secret. According to the instructions and data executed, the attacker-controlled code will cause different heat (i.e., the red dot is higher than the blue dot). The other piece contains ThermalScope. After the execution of the attacker-controlled code, ThermalScope-based code loops to execute a specific operation until a thermal event interrupt is detected. Even if there is heat loss caused by the cooling device, the heat generated by the attacker code should be negatively correlated with the remnant heat. Considering that the heat generated by the attack code depends on the number of executed instructions, when starting from a higher temperature (the red dot), the attack code requires fewer instructions to execute. We use the number of executed instructions to guide our attack.

Observing Thermal Event Interrupts: In the native scenario, the `/proc/interrupts` interface provides statistics of various types of interrupts, which are available to unprivileged processes in Linux. Previous works have exploited the interface to observe targeted device interrupts, such as GPU [6]. Similar to them, we can also leverage this interface to retrieve the thermal event interrupts. To mitigate existing interrupt side channels, the interface can be disabled for unprivileged users. To bypass the mitigation, the `/sys/devices/system/cpu/cpuX/thermal_throttle` interface has been identified to provide information about our targeted thermal event interrupts. Specifically, this interface allows us to obtain the number of thermal events, whose logging depends on whether there is a thermal event interrupt occurring within a certain time interval.

In the browser scenario, the two interfaces above become inaccessible. However, we observe that thermal event interrupts are always triggered on all cores, resulting in a reduction in user-mode time. To this end, we use a loop-counting program to observe interrupts that have occurred, including thermal event interrupts. Specifically, the program loops to increase a counter value and sample its increment at fixed time intervals set by a timer. If the running core receives any interrupt (e.g., thermal event interrupts), context switch occurs, slowing down the counter's increment and thus signaling the occurrence of an interrupt. As high-resolution timers are crippled in browser scenario, the loop-counting program uses a low-resolution timer with a granularity of millisecond (e.g., 1 ms for Chrome and Firefox) to monitor these interrupts.

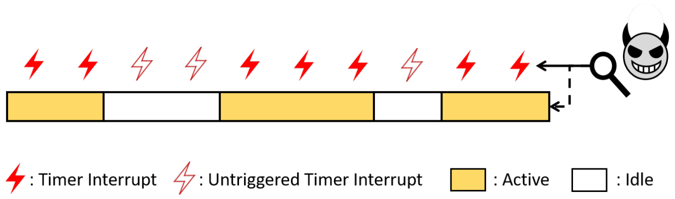

Fig. 5. An illustration of using *TimerScope-Monitor* to exploit timer interrupts to infer victim activities.

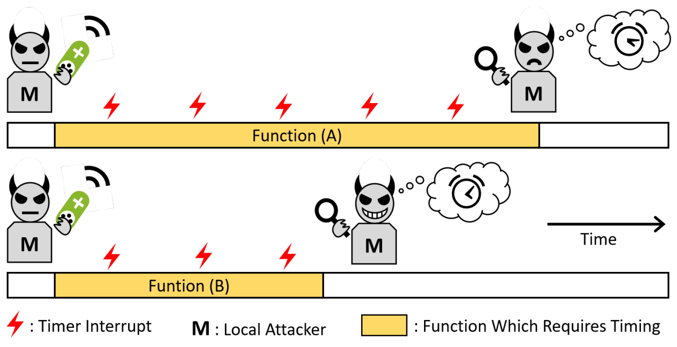

Fig. 6. An illustration of using *TimerScope-Probe* to exploit timer interrupts to time the execution of victim code. The number of generated timer interrupts can be used as an alternative timing source to monitor the elapsed time.

B. *TimerScope*

On x86 architectures, timer interrupts are generated by the Advanced Programmable Interrupt Controller (APIC) at fixed intervals. By configuring the APIC, privileged software (e.g., the OS kernel) can precisely control the generation of these interrupts. Previous work has primarily focused on privileged exploitation, where attackers inject timer interrupts to periodically pause the execution of victim enclave programs, subsequently using various side channels to compromise their confidentiality. To date, the only unprivileged exploitation is SegScope [5], which leverages timer interrupts as clock edges to build a fine-grained timer. In this work, we aim to demonstrate that timer interrupts themselves can be exploited as a side channel to leak information about CPU activities. Same as ThermalScope, TimerScope also contains two variants to enable a cross-core/same-core attacker, namely *TimerScope-Monitor* and *TimerScope-Probe*.

The main idea behind `TimerScope` is that the APIC hardware in x86 CPUs stops generating timer interrupts to save energy when the core becomes idle and no context switch is needed [40], [41]. By default, Linux-based systems support the *tickless* mode, which allows the operating system to suppress periodic timer interrupts once the target core enters `C-state 1` or deeper C-states. In contrast, when the core remains active, timer interrupts continue at a fixed rate, providing a stable and accessible timing source for unprivileged processes.

TimerScope-Monitor: Our first primitive is to exploit timer interrupts to infer whether victim core is idle or not, which relies on the `/proc/interrupts` interface to obtain interrupt statistics of that core. As shown in Figure 5, the attacker runs their code on a different core while monitoring the timer interrupts of the victim core. Timer interrupts are only triggered when the victim core is active. If the victim program requests specific resources (e.g., webpage rendering) and then relinquishes the CPU for a certain period, timer interrupts will pause. By monitoring these interrupts, the attacker can infer the victim's activities.

TimerScope-Probe: The second variant of TimerScope focuses on the fixed time interval between two consecutive interrupts. Our goal is to use the number of triggered timer interrupts as a timing primitive to measure the duration of a specific function. As shown in Figure 6, *TimerScope-Probe* operates with a piece of attacker-controlled code (e.g., a snippet that accesses a kernel address to break KASLR) that needs to be timed. The attacker queries the number of timer interrupts before and after the execution of the code. By calculating the difference in the number of interrupts, the attacker can derive an estimated execution time, with the granularity of this measurement determined by the interval between consecutive clock interrupts, which is typically in the level of milliseconds.

Observing Timer Interrupts: As mentioned above, TimerScope relies on `/proc/interrupts` to collect statistics of either cross-core or same-core interrupts. We note that a recent work has proposed a new technique for monitoring cross-core interrupts via microarchitectural changes [9]. However, this approach depends on IPI virtualization features available only on Intel Sapphire Rapids and Arrow Lake processors, which were commercially released in 2023 and November 2024, respectively. These platforms are not included in our experimental setup. Our objective is to demonstrate the feasibility of exploiting timer interrupts as a leakage source, and we leave the application of state-of-the-art microarchitectural

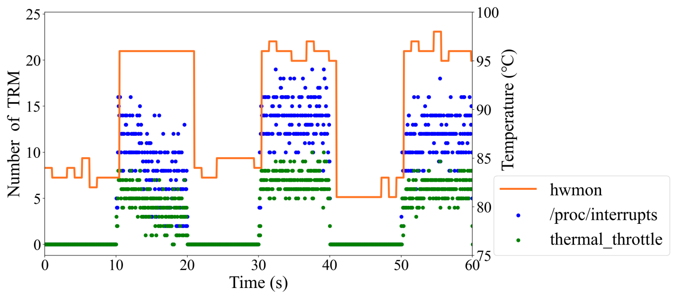

Fig. 7. The relationship between the number of thermal event interrupts (TRM), the number of package thermal events, and temperature (through `hwmon`). Thermal event interrupts are activated only when the CPU package temperature exceeds a specific threshold, and the thermal events are logged.

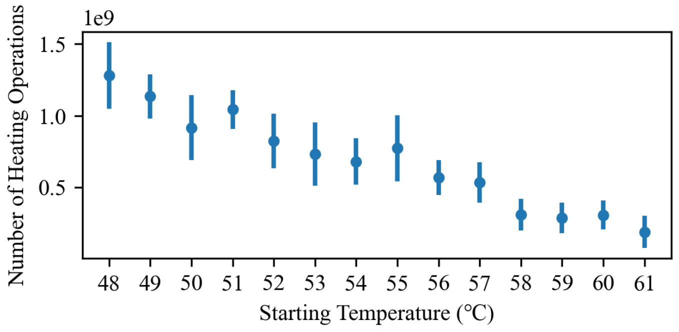

Fig. 8. An approximate linear relationship between the heating time and starting temperature (through `hwmon`).

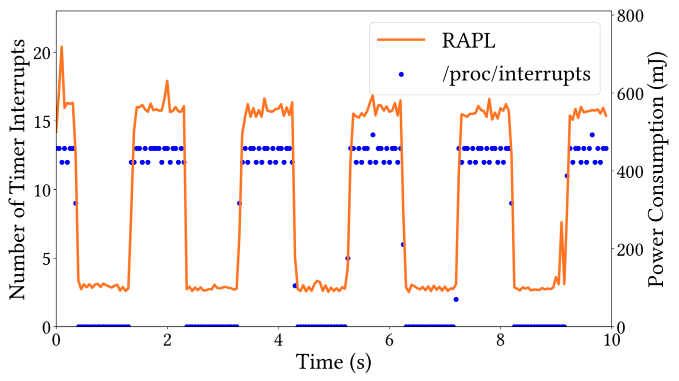

Fig. 9. The relationship between the number of timer interrupts and power consumption (through `RAPL`). Only when the CPU is active, the timer interrupts are activated and the power consumption are increased.

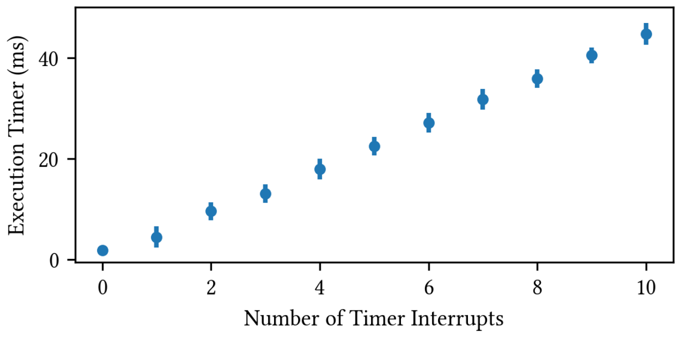

Fig. 10. An approximate linear relationship between the number of timer interrupts and execution time (through `rdtsc`).

techniques for observing timer interrupts to future work.

C. Characterizing ThermalScope and TimerScope

Characterizing ThermalScope: To show the effectiveness of ThermalScope in measuring temperature changes, we compare ThermalScope results with the temperature variation reported by the `hwmon` interface. `hwmon` is a mechanism for measuring and controlling the temperature of individual components of Linux-based machines. It provides a real-time measurement of the temperature for each CPU core, based on high resolution sampling of the integrated thermal sensors inside the CPU.

In this experiment, `/proc/interrupts` is used to obtain the interrupt statstics and `thermal_throttle` is used to obtain the number of thermal events. Aligned with previous work [6], we choose a sampling period of 50 ms. To cause temperature increments, we use a certain number of *sqrt* operations to calculate the square root of a non-negative random number. We separately evaluate the two variants of ThermalScope on the Lenovo machine.

First, we mount *ThermalScope-Monitor* that executes *sqrt* operations for 10 seconds and sleeps for 10 seconds in turns. At the same time, we use another process running on a separate core to record the temperature via `hwmon`, the number of thermal event interrupts, and the number of thermal events every 50 ms. In this way, we can directly observe the relationship between the activation of thermal event interrupts and CPU package temperature.

Second, we apply *ThermalScope-Probe* to observe the relationship between the number of needed instructions and CPU starting temperatures. To achieve this, we have two pieces of code to execute. The first piece of code (i.e., attacker-controlled code) performs a random number of *sqrt* operations to cause different starting temperatures. Then, the second piece of code starts loops to execute *sqrt* operations and records the number of executed operations. At the same time, its child thread running on another core is used to monitor the activation of thermal event interrupts and notify the heating process to break their loops if a new interrupt is observed. We run the above steps for 1000 times.

Experimental Results: As shown in Figure 7, only when the temperature reaches a threshold (e.g., 80 °C in our Xiaomi machine and 90 °C in our Lenovo machine), the thermal event interrupts are activated and the thermal events are logged. Besides, the number of thermal events is less than thermal event interrupts, because the logging frequency of thermal events is limited by kernel. If there are two thermal events in a logging cycle, the kernel will only log them once. Figure 8 shows the relationship between heating time and starting temperature. These two values approximately satisfy a linear relationship. The lower the starting temperature is, the more *sqrt* operations are executed. So the attacker can use the number of executed operations to infer the starting temperature that depends on the system activity.

Characterizing TimerScope: We then characterize ThermalScope by using the activation of timer interrupts to infer victim activity and measure the execution time of the victim code. Our aim is to confirm that TimerScope can be used to determine whether the victim core is active or idle. Aligned with HammerScope [50], we test TimerScope under two extremes. To support this comparison, we leverage `RAPL` [51], [52] to capture the energy consumption of whole CPU package, and `rdtsc` to obtain precise CPU cycle counts which are then translated into real time. Since all other cores remain idle during this experiment, variations in the measured `RAPL` values primarily reflect the power consumption of the victim core.

First, we run a piece of victim code that executes *sqrt* operations for 1 second and sleeps for 1 second in turns. At the same time, we use another process running on another separate core to record the increment of energy consumption and the number of timer interrupts every 50 ms. In this way, we can directly observe the relationship between the activation of timer interrupts and CPU activities. To highlight the strong correlation between the timer interrupts and power consumption, we compute the Pearson correlation coefficient (ρ) and regression coefficients (p). Pearson correlation coefficient is used to assess the strength and direction of the linear association, while the regression coefficient measures the impact of these two values in a linear regression model. We note that we use privileged execution to apply `RAPL` to obtain power consumption.

Second, we apply *Timer-Probe* to examine the relationship between the number of timer interrupts and the execution time of the victim code. To do this, we run attacker-controlled code that performs a random number of *sqrt* operations, generating varying execution times. We query both timer interrupts before and after each execution of the victim code. This procedure is repeated 1000 times to ensure statistical significance.

Experimental Results: As shown in Figure 9, both RAPL and the TimerScope measurements can accurately capture the CPU activity. The RAPL and TimerScope measurements are highly correlated (ρ = 0.984, $p < 0.001$). Figure 10 shows the relationship between execution time and increment of timer interrupts. These two values approximately satisfy a linear relationship. The more the execution time is, the more timer interrupts are triggered. So the attacker can use the number of timer interrupts to time the victim code.

V. Case Studies

A. Building Covert Channels

A covert channel [16], [49], [50], [53]–[55] requires a pair of colluding sender and receiver, which are usually not allowed to communicate over normal channels. We assume that a sender actively executes *sqrt* operations on random numbers to stress its CPU cores. Depending on the sender's intention to send bit '0' or bit '1', the child threads execute for different time. Meanwhile, the receiver operates on a separate core and utilizes either *ThermalScope-Monitor* or *TimerScope-Monitor* to intercept the covertly transmitted bits. To evaluate the efficacy of *ThermalScope-Monitor*, the receiver leverages the `thermal_throttle` interface to capture thermal event interrupts. For *TimerScope-Monitor*, the receiver monitors timer interrupts from the sender's core through `/proc/interrupts`.

Experimental Setup: For each test, we transmit 1 kB of random data between two unprivileged processes running on different cores of the Lenovo machine. The sender controls the execution number of *sqrt* operations to control the activation of non-movable interrupts. For the ThermalScope-based covert channel, a '1' is transmitted by executing *sqrt* operations for 5 seconds, followed by a 5-second sleep period. Conversely, a '0 is transmitted by sleeping for 10 seconds. Additionally, to ensure the activation of thermal event interrupts when transmitting a '1', the sender spawns a separate process on another core that continually increments a counter. For the TimerScope-based covert channel, a '1' is transmitted by executing *sqrt* operations for 100 milliseconds, while a '0' is transmitted by sleeping for 100 milliseconds. The receiver captures the increments of thermal event interrupts or timer interrupts at consistent intervals via `/proc/interrupts` or `thermal_throttle`, and by comparing these increments against a predefined threshold, it determines the transmitted bits as either '0' or '1'.

Experimental Results: For the ThermalScope-based covert channel, our proof-of-concept (PoC) implementation achieves a transmission rate of 0.1 bit/s with a bit error rate of 0.2% when using the `/proc/interrupts` interface and 0.7% when using the `thermal_throttle` interface (averaged across 10 trials) on our Lenovo machine. On the same platform, the TimerScope-based covert channel achieves a transmission rate of 10 bit/s with a bit error rate of 0%. While the transmission rate of our covert channel is lower in contrast to other state-of-the-art covert channels [56], [57], our covert channel has the benefit that it does not rely on high-resolution timers and exploits a different mechanism (i.e., non-movable interrupts). And we believe that this case study can serve as a significant benchmark for our side channel's bandwidth.

B. Breaking KASLR

In Linux, KASLR is implemented to randomize the base address of the text segment at every boot time, aligning it with a 2 MB boundary and mapping it to an address within a range of 1 GB [17], [58]–[61]. If an attacker wants to determine the text base address, a maximum of 512 times of guessing is needed. However, prior work [5], [17], [58] has shown that a side channel leakage occurs when executing *prefetch* instructions on a memory address, depending on whether that address is mapped or not. Our key idea of breaking KASLR is that the activation of non-movable interrupts also depends on the execution of *prefetch* instructions. We can use *ThermalScope-Probe* or *TimerScope-Probe* to distinguish the *prefetch* instructions to mapped address from the *prefetch* instructions to unmapped address.

Experimental Setup: For the *ThermalScope-Probe*, the attacker-controlled code executes 3 billion *prefetch* instructions at each address. Following this, the attack code, which time-shares the same physical core as the attacker-controlled code, repeatedly performs *sqrt* calculations on random numbers until a thermal event interrupt is triggered. The attacker distinguishes whether an address is mapped or not based on the number of executed *sqrt* operations. To allow for

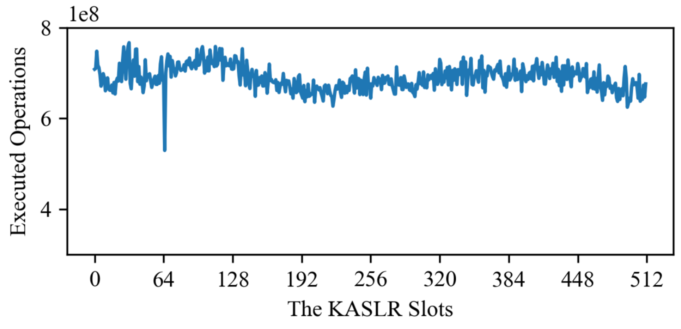

Fig. 11. Number of executed *sqrt* operations when we apply *ThermalScope-Probe* on every possible base address. The X-axis represents the candidate address for the kernel text segment.

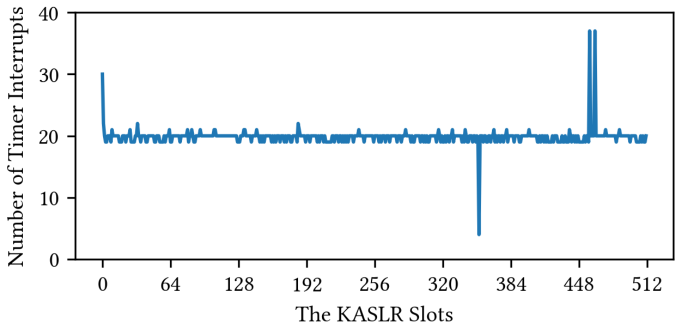

Fig. 12. Number of timer interrupts when we apply *TimerScope-Probe* on every possible base address.

adequate cooling, a 30-second pause is introduced before each attempt. For the *TimerScope-Probe*, the attacker-controlled code executes 10 million *prefetch* instructions at each address, while the number of timer interrupts during this period is recorded. If an address in the *ThermalScope-Probe* requires significantly fewer operations to trigger the thermal event interrupt compared to others, we conclude that the address is mapped. Similarly, if the number of timer interrupts in the *TimerScope-Probe* is notably smaller for a particular address, we also infer that it is the mapped address.

Experimental Results: Figure 11 presents the results of applying *ThermalScope-Probe* to break KASLR, which is conducted on the Lenovo machine. The entire attack process takes 8.2 hours. When prefetching a mapped memory, the execution of *prefetch* instruction is faster and thus the starting temperature in *ThermalScope-Probe* is higher than others. As a result, fewer *sqrt* operations are needed to make the package temperature reach the fixed threshold. Obviously, the slot 64 corresponds to the mapped address. Figure 12 presents the results of applying *TimerScope-Probe* to break KASLR. When prefetching a mapped memory, the execution of *prefetch* instruction is faster and thus the increments of timer interrupts are lower than others. As *TimerScope-Probe* achieves a granularity of millisecond (see Section IV-C), the entire attack process only takes 51.6 seconds on our Xiaomi machine.

TABLE V
THE IMPACT OF THE SAMPLING OF INTERRUPTS (RFOREST IS USED AS THE CLASSIFIER).

Setting	Xiaomi	Lenovo	**Average**
Loop-counting (Native)	0.933	0.923	0.938
Loop-counting (JavaScript)	0.589	0.459	0.524
Thermal Event Interrupts	0.964	0.982	0.973
Overvall Interrupts	0.927	0.941	0.934
Other Interrupts	0.418	0.407	0.418

C. Browser-based DNN Model Fingerprinting Attack

Here, we demonstrate a practical cross-core attack using the techniques discussed in Section IV.A, which does not rely on `/proc/interrupts` and thus can work under the *browser* scenario [12], [62]–[67]. To achieve this, we leverage heat padding to trigger thermal event interrupts and use a loop-counting program to continuously increment a counter value. We aim to show that thermal event interrupts can serve as a reliable cross-core leakage source for browser-based DNN model fingerprinting attack. In our experiments, we sample this counter at a fixed interval of 10 ms, collecting 5,000 points for each attack. Since the loop-counting code measures instruction throughput, which can be affected by processor frequency scaling, we fix the CPU frequency using the Linux `cpufreq-set` command. Besides, we apply `isolcpus` to prevent network and other movable interrupts from being handled on the sampling cores. We implement the loop-counting functionality using native C code and JavaScript code within Firefox v135.0 on the Xiaomi machine and Firefox v125.0 on the Lenovo machine. For the heat padding code, we use one or more tabs to perform computations using JavaScript code.

Ablation Study: We also test our method in other settings including observing thermal event interrupts, overall interrupts, and the other interrupts under the native scenario, where we sample the increment of thermal event interrupts from `/proc/interrupts` by default at a fixed interval of 50 ms and collect 1,000 points for each attack. For the other settings, We separately use `/proc/interrupts` to obtain the sum of the overall interrupts on the attack core and the sum of interrupts other than thermal event interrupts and at a fixed interval of 50 ms.

As our research focus is not on the deep learning algorithm, we feed the collected side channel traces without any preprocessing into the RForest classifier presented in Section III.C. For each experiment, we record 100 traces for each of the 36 model architectures shown in Table IV. For each classifier, we perform the 10-fold cross-validation during the evaluation, where nine folds are used as the training data, and the remaining one fold is retained as the testing data.

Experimental Results: Table V reports the classification accuracy of our RForest model under different settings. Our loop-counting attack remains effective even when the attacker core is isolated. When executed within the browser using JavaScript where there is more system noise, the fingerprinting accuracy inevitably decreases compared to native execution; however, the achieved accuracy of 52.4% still significantly exceeds random guessing, which yields only 2.8% across 36 architectures. Besides, without thermal event interrupts, using the sum of other interrupts can only achieve a low success rate of 41.8%. Last, the average success rate of overall interrupts and thermal event interrupts setting is 93.4% and 97.3%, respectively, which validates how thermal event interrupts contribute to our browser-based attack. The reason for the slightly higher success rate of our native loop-counting attack is that the loop-counting program essentially observes the handling time of overall interrupts, rather than the number, thus capturing more information.

VI. Discussion

In this section, we present the future work and then discuss the possible mitigation.

A. Future Work

Besides ThermalScope and TimerScope, there might also exist other unprivileged exploitation of non-movable interrupts except model stealing attack. As discussed in Section III, we identified 5 additional non-movable interrupts that have not yet been fully investigated in this paper. For instance, TLB shootdowns could potentially be exploited to probe whether new TLB entries are created on the same or different cores. Similarly, with performance monitoring interrupts, an unprivileged attacker could deliberately manipulate one of the performance monitoring counters to approach its overflow threshold. By then triggering or waiting for the victim's code to execute and observing whether a performance monitoring interrupt is activated, the attacker could infer whether a specific performance event (e.g., cache miss and branch prediction failure) occurred. Further evaluation of these instances with real-world case studies is left for future work.

B. Mitigation Strategies

Eliminating the Information Leakage: Since non-movable interrupts cannot easily be isolated to a dedicated core that is inaccessible to attackers, mitigating the associated information leakage requires tailored defenses for each type of non-movable interrupt. For thermal event interrupts, one potential mitigation strategy is to improve the system's cooling devices to minimize the occurrence of thermal event interrupts, thereby reducing the granularity of information that can be leaked through temperature fluctuations. For instance, two of our tested machines are laptops, which exhibit poorer cooling performance compared to typical desktop or server systems. In contrast, our desktop machine, which is equipped with a dedicated cooling system, experiences significantly fewer thermal events than them. This observation underscores that stronger cooling mechanisms can effectively mitigate ThermalScope-style attacks and highlights the importance of addressing information leakage risks, particularly in mobile devices.

In the case of timer interrupts, defenders could recompile the kernel to enable `tickless-full` mode, which eliminates periodic timer interrupts on cores running a single process. This approach, coupled with assigning the victim process to a dedicated core, can significantly reduce the exposure to timer-based side-channel attacks by limiting the availability of exploitable interrupts on the core where the attacker resides.

Restricting Unprivileged Observations: Our high-resolution observation method leverages interrupt counts available through system files such as `/proc/interrupts` (and `thermal_throttle` specifically for thermal event interrupts), which are accessible to unprivileged users. By monitoring these counts, attackers can directly acquire the number of targeted non-movable interrupts, thereby creating a side channel for data leakage without requiring elevated privileges. To mitigate this risk, we recommend restricting unprivileged access to these files by modifying file permissions or enforcing stricter access controls. However, this approach only partially addresses the issue, as it does not prevent timing-based [9], [12] or microarchitectural observations [5], [7], [8], which extract interrupt information through execution timing or microarchitectural state changes.

Injecting Artificial Noise to Interrupts: For users, we also discuss some possible mitigation to prevent the side channel leakage of non-movable interrupts. Specifically, to mitigate TimerScope, the defender can co-locate a process with the attacker process. This process takes up a random amount of time on the CPU each time. By time-sharing the CPU, this will introduce random noise to the core that the attacker is monitoring. Similarly, to mitigate ThermalScope, a random computation process can also be used to generate noise to thermal event interrupts. While this strategy cannot entirely eliminate a side channel, it effectively reduces its signal strength [8], [9].

VII. Conclusion

In this paper, we conducted an empirical study on exploitable non-movable interrupts and their contribution to interrupt-based side-channel leakages in x86-based system. We proposed a dynamic analysis technique to examine how different types of non-movable interrupts are affected by various workloads. Using a model fingerprinting attack as a benchmark, we demonstrated that seven types of non-movable interrupts are exploitable. Building on these findings, we introduced two practical side channels, ThermalScope and TimerScope, which can infer victim activities when the attacker and victim share the same core or reside on separate cores. Last, we applied them to real-world case studies, successfully constructing cross-core covert channels and breaking kernel address space layout randomization (KASLR). Our work aims to emphasize the need to tackle information leakage at its source by conducting a comprehensive analysis of the risks posed by non-movable interrupts.

References

[1] Y. Wang, F. Gao, L. Wang, T. Yu, J. Zhao, and X. Li, "Automatic detection, validation, and repair of race conditions in interrupt-driven embedded software," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 346–363, 2022.

[2] C. Ye, Y. Cai, and C. Zhang, "When threads meet interrupts: Effective static detection of Interrupt-Based deadlocks in linux," in *USENIX Security Symposium*, 2024, pp. 6167–6184.

[3] Y. Lee, C. Min, and B. Lee, “ExpRace: Exploiting kernel races through raising interrupts,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2363–2380.

[4] A. Tai, I. Smolyar, M. Wei, and D. Tsafrir, “Optimizing storage performance with calibrated interrupts,” *ACM Transactions on Storage*, vol. 18, no. 1, 2022.

[5] X. Zhang, Z. Zhang, Q. Shen, W. Wang, Y. Gao, Z. Yang, and J. Zhang, “Segscope: Probing fine-grained interrupts via architectural footprints,” in *High Performance Computer Architecture*, 2024.

[6] H. Ma, J. Tian, D. Gao, and C. Jia, “On the effectiveness of using graphics interrupt as a side channel for user behavior snooping,” *IEEE Transactions on Dependable and Secure Computing*, pp. 3257–3270, 2021.

[7] R. Zhang, T. Kim, D. Weber, and M. Schwarz, “(M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels,” in *USENIX Security Symposium*, 2023.

[8] F. Rauscher, A. Kogler, J. Juffinger, and D. Gruss, “Idleleak: Exploiting idle state side effects for information leakage,” in *Network and Distributed System Security Symposium*, 2024.

[9] F. Rauscher and D. Gruss, “Cross-core interrupt detection: Exploiting user and virtualized ipis,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2024.

[10] D. Weber, F. Thomas, L. Gerlach, R. Zhang, and M. Schwarz, “Indirect meltdown: Building novel side-channel attacks from transient-execution attacks,” in *ESORICS*, 2023, pp. 22–42.

[11] J. Trostle, “Timing attacks against trusted path,” in *IEEE Symposium on Security and Privacy*, 1998, pp. 125–134.

[12] M. Lipp, D. Gruss, M. Schwarz, D. Bidner, C. Maurice, and S. Mangard, “Practical keystroke timing attacks in sandboxed javascript,” in *European Symposium on Research in Computer Security*, 2017, pp. 191–209.

[13] M. Schwarz, M. Lipp, D. Gruss, S. Weiser, C. Maurice, R. Spreitzer, and S. Mangard, “Keydrown: Eliminating software-based keystroke timing side-channel attacks,” in *Network and Distributed System Security Symposium*, 2018.

[14] J. Cook, J. Drean, J. Behrens, and M. Yan, “There’s always a bigger fish: A clarifying analysis of a machine-learning-assisted side-channel attack,” in *International Symposium on Computer Architecture*, 2022, p. 204–217.

[15] X. Zhang, Z. Zhang, Q. Shen, W. Wang, Y. Gao, Z. Yang, and Z. Wu, “Thermalscope: A practical interrupt side channel attack based on thermal event interrupts,” in *Design Automation Conference*, 2024.

[16] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, “Hertzbleed: Turning power Side-Channel attacks into remote timing attacks on x86,” in *USENIX Security Symposium*, 2022, pp. 679–697.

[17] T. Kim and Y. Shin, “Thermalbleed: A practical thermal side-channel attack,” *IEEE Access*, vol. 10, pp. 25 718–25 731, 2022.

[18] L. community, “Irqbalance,” https://github.com/Irqbalance/irqbalance, 2015.

[19] Y. Li, Z. Zhang, B. Liu, Z. Yang, and Y. Liu, “Modeldiff: testing-based dnn similarity comparison for model reuse detection,” in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, p. 139–151.

[20] E. Debenedetti, G. Severi, N. Carlini, C. A. Choquette-Choo, M. Jagielski, M. Nasr, E. Wallace, and F. Tramèr, “Privacy side channels in machine learning systems,” in *USENIX Security Symposium*, 2024, pp. 6861–6848.

[21] C. Gongye, Y. Luo, X. Xu, and Y. Fei, “Side-channel-assisted reverse-engineering of encrypted dnn hardware accelerator ip and attack surface exploration,” in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2023, pp. 1–1.

[22] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, “Deepem: Deep neural networks model recovery through em side-channel information leakage,” in *Hardware Oriented Security and Trust*, 2020.

[23] W. Jiang, H. Li, G. Xu, T. Zhang, and R. Lu, “A comprehensive defense framework against model extraction attacks,” *IEEE Transactions on Dependable and Secure Computing*, pp. 685–700, 2024.

[24] L. Gao, W. Liu, K. Liu, and J. Wu, “Augsteal: Advancing model steal with data augmentation in active learning frameworks,” *IEEE Transactions on Information Forensics and Security*, pp. 4728–4740, 2024.

[25] M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2015, p. 1322–1333.

[26] M. Khosravy, K. Nakamura, Y. Hirose, N. Nitta, and N. Babaguchi, “Model inversion attack by integration of deep generative models: Privacy-sensitive face generation from a face recognition system,” *IEEE Transactions on Information Forensics and Security*, pp. 357–372, 2022.

[27] T. Zhu, D. Ye, S. Zhou, B. Liu, and W. Zhou, “Label-only model inversion attacks: Attack with the least information,” *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 991–1005, 2023.

[28] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against machine learning,” in *Asia Conference on Computer and Communications Security*, 2017, p. 506–519.

[29] L. G. Hafemann, R. Sabourin, and L. S. Oliveira, “Characterizing and evaluating adversarial examples for offline handwritten signature verification,” *IEEE Transactions on Information Forensics and Security*, pp. 2153–2166, 2019.

[30] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Faruque, “Leaky DNN: Stealing deep-learning model secret with gpu context-switching side-channel,” in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2020, pp. 125–137.

[31] H. T. Maia, C. Xiao, D. Li, E. Grinspun, and C. Zheng, “Can one hear the shape of a neural network?: Snooping the gpu via magnetic side channel,” in *USENIX Security Symposium*, 2022.

[32] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque, “Stealing neural network structure through remote fpga side-channel analysis,” *IEEE Transactions on Information Forensics and Security*, pp. 4377–4388, 2021.

[33] Y. Gao, H. Qiu, Z. Zhang, B. Wang, H. Ma, A. Abuadbba, M. Xue, A. Fu, and S. Nepal, “Deeptheft: Stealing dnn model architectures through power side channel,” in *IEEE Symposium on Security and Privacy*, 2024.

[34] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, “Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories,” in *IEEE Symposium on Security and Privacy*, 2022, pp. 1157–1174.

[35] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, “Deepsniffer: A dnn model extraction framework based on learning architectural hints,” in *Architectural Support for Programming Languages and Operating Systems*, 2020, p. 385–399.

[36] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, “Hermes attack: Steal dnn models with lossless inference accuracy,” in *USENIX Security Symposium*, 2021.

[37] K. Patwari, S. M. Hafiz, H. Wang, H. Homayoun, Z. Shafiq, and C.-N. Chuah, “Dnn model architecture fingerprinting attack on cpu-gpu edge devices,” in *European Symposium on Security and Privacy (EuroS&P)*, 2022, pp. 337–355.

[38] C. Liu, D. Wang, Y. Lyu, P. Qiu, Y. Jin, Z. Lu, Y. Zhang, and G. Qu, “Uncovering and exploiting amd speculative memory access predictors for fun and profit,” in *High Performance Computer Architecture*, 2024, pp. 31–45.

[39] J. O. Weiss, T. Alves, and S. Kundu, “Ezclone: Improving dnn model extraction attack via shape distillation from gpu execution profiles,” *arXiv preprint*, 2023.

[40] Intel, Inc., “Intel 64 and IA-32 architectures software developer’s manual combined volumes: 1, 2a, 2b, 2c, 3a, 3b and 3c,” 2019.

[41] AMD, Inc., “Amd64 architecture programmers manual volume 2: System programming,” 2019.

[42] J. Van Bulck, F. Piessens, and R. Strackx, “Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2018, p. 178–195.

[43] J. Cui, J. Z. Yu, S. Shinde, P. Saxena, and Z. Cai, “Smashex: Smashing sgx enclaves using exceptions,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2021, p. 779–793.

[44] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, “Zombieload: Cross-privilege-boundary data sampling,” in *ACM SIGSAC Conference on Computer and Communications Security*, 2019, p. 753–768.

[45] K. Colin, “Stress next generation,” 2023. [Online]. Available: https://github.com/ColinIanKing/stress-ng

[46] R. F. Tate, “Correlation between a discrete and a continuous variable. point-biserial correlation,” *The Annals of Mathematical Statistics*, vol. 25, no. 3, pp. 603–607, 1954.

[47] X. Zhang, Y. Yang, J. Zou, Q. Shen, Z. Zhang, Y. Gao, Z. Wu, and T. Carlson, “AmpereBleed: Exploiting On-chip Current Sensors for Circuit-Free Attacks on ARM-FPGA SoCs,” in *ACM/IEEE Design Automation Conference (DAC)*, 2025.

[48] T. Ni, G. Lan, J. Wang, Q. Zhao, and W. Xu, "Eavesdropping mobile app activity via Radio-Frequency energy harvesting," in *USENIX Security Symposium*, 2023, pp. 3511–3528.

[49] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S. Capkun, "Thermal covert channels on multi-core platforms," in *USENIX Security Symposium*, 2015.

[50] Y. Cohen, K. S. Tharayil, A. Haenel, D. Genkin, A. D. Keromytis, Y. Oren, and Y. Yarom, "Hammerscope: Observing dram power consumption using rowhammer," in *ACM SIGSAC Conference on Computer and Communications Security*, 2022, p. 547–561.

[51] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," in *IEEE Symposium on Security and Privacy*, 2021, pp. 355–371.

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

[53] D. B. Bartolini, P. Miedl, and L. Thiele, "On the capacity of thermal covert channels in multicores," in *European Conference on Computer Systems*, 2016.

[54] D. Cotroneo, L. D. Simone, and R. Natella, "Timing covert channel analysis of the vxworks mils embedded hypervisor under the common criteria security certification," *Computers & Security*, 2021.

[55] X. Zhang, J. Zou, Y. Yang, Q. Shen, Z. Zhang, Y. Gao, Z. Wu, and T. Carlson, "LeakyDSP: Exploiting Digital Signal Processing Blocks to Sense Voltage Fluctuations in FPGAs," in *ACM/IEEE Design Automation Conference (DAC)*, 2025.

[56] C. Chen, J. Cui, G. Qu, and J. Zhang, "Write+sync: Software cache write covert channels exploiting memory-disk synchronization," *IEEE Transactions on Information Forensics and Security*, p. 8066–8078, 2024.

[57] Y. Yarom and K. Falkner, "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack," in *USENIX Security Symposium*, 2014, pp. 719–732.

[58] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing smap and kernel aslr," in *ACM SIGSAC Conference on Computer and Communications Security*, 2016, p. 368–379.

[59] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss, "KASLR: Break it, fix it, repeat," in *Asia Conference on Computer and Communications Security*, 2020, p. 481–493.

[60] R. Hund, C. Willems, and T. Holz, "Practical timing side channel attacks against kernel space aslr," in *IEEE Symposium on Security and Privacy*, 2013, pp. 191–205.

[61] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss, "Kaslr: Break it, fix it, repeat," in *Asia Conference on Computer and Communications Security*, 2020, p. 481–493.

[62] J. Kim, S. van Schaik, D. Genkin, and Y. Yarom, "Ileakage: Browser-based timerless speculative execution attacks on apple devices," in *ACM SIGSAC Conference on Computer and Communications Security*, 2023, p. 2038–2052.

[63] H. Xiao and S. Ainsworth, "Hacky racers: Exploiting instruction-level parallelism to generate stealthy fine-grained timers," in *Architectural Support for Programming Languages and Operating Systems*, 2023.

[64] H. Kim, K.-D. Kang, G. Park, S. Lee, and D. Kim, "Brokensleep: Remote power timing attack exploiting processor idle states," in *High Performance Computer Architecture*, 2025, pp. 409–422.

[65] L. Giner, R. Czerny, C. Gruber, F. Rauscher, A. Kogler, D. D. A. Braga, and D. Gruss, "Generic and automated drive-by gpu cache attacks from the browser," in *Asia Conference on Computer and Communications Security*, 2024, p. 128–140.

[66] S. Gast, J. Juffinger, L. Maar, C. Royer, A. Kogler, and D. Gruss, "Remote scheduler contention attacks," in *International Conference on Financial Cryptography and Data Security*. Springer, 2024, pp. 365–383.

[67] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard, "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript," in *Financial Cryptography and Data Security*, 2017, pp. 247–267.

Xin Zhang received the B.Sc. degree in information security from Hunan University in 2022. He is currently pursuing the Ph.D. degree with Peking University, Beijing, China. His research interests include system security, computer architecture and side channel attack.

Qingni Shen received the Ph.D. degree from the Institute of Software Chinese Academy of Sciences, Beijing, China, in 2006. She is currently a professor with the School of Software and Microelectronics and the director of PKU_OCTA Lab of Blockchain and Privacy Computing, Peking University, Beijing, China. Her research interests include operating system security, cloud security and privacy, blockchain and privacy computing, and trusted computing. She is a Distinguished Membership of CCF and an ACM/IEEE Membership. She is serving as the reviewer for TOMPECS, JPDC, FGCS, Computers & Security, Information Sciences, the Computer Journal, ICICS, TrustCom, ACNS, ACM MultiMedia, ICASSP, and so on.

Zhi Zhang received the Ph.D. degree from The University of New South Wales. He is a Lecturer at The University of Western Australia. His current research interests include hardware security, system security, and their intersections with AI security. He was a recipient of USENIX SECURITY 2024 Distinguished Paper Award and ASIACCS 2023 Distinguished Paper Award. He serves as an Associate Editor for IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING. He also serves on the Program Committees for ASPLOS, DSN, ASIACCS.

Yansong Gao (Senior Member, IEEE) is a Lecturer at the University of Western Australia. He received his M.Sc degree from the University of Electronic Science and Technology of China and a Ph.D. degree from the University of Adelaide, Australia. His current research interests are AI security and privacy, system security, and hardware security. He serves as an Associate Editor of IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.

Jiajun Zou received the B.Sc. degree in software engineering from Hunan University in 2024. He is currently pursuing the Master degree with Peking University, Beijing, China. His research interests include system security and side channel attack.

Yi Yang received the B.Sc. degree in Information Security from Chongqing University of Posts and Telecommunications in 2020. He is currently pursuing the Master degree with Peking University, Beijing, China. His research interests include system security and side channel attack.

Zhonghai Wu received the PhD degree from Zhejiang University, Hangzhou, China, in 1997. Previously, he worked as a postdoctoral researcher with the Institute of Computer Science and Technology, Peking University, Beijing, China. Since then, he has been involved both in research and development of distributed system, service, and security. He is currently the dean of the School of Software and Microelectronics, Peking University, Beijing, China. His research interests include cloud computing, data intelligence, and system security.