Fish and Chips: On the Root Causes of Co-Located Website-Fingerprinting Attacks

Yusi Feng, Sioli O'Connel, Xin Zhang, Chitchanok Chuengsatiansup, Daniel Genkin, Yuval Yarom, Yinqian Zhang, *Senior Member, IEEE*, and Zhi Zhang, *Member, IEEE*

***Abstract*—Microarchitectural website-fingerprinting attacks use timing information to leak the browsing habits of a victim to co-resident attackers. Microarchitectural leakage in these attacks often comprises multiple sources. While most published attacks claim to identify the cause of leakage, these claims are not always well supported. Thus, so far the question of how to determine what leaks remains mostly unanswered. In this work, we develop a framework for identifying and measuring the contribution of leakage sources to the overall observations the attacker makes. Experimenting with three website-fingerprinting attacks in the literature, we qualitatively identify four main classes of leakage sources: core contention, interrupts, frequency scaling, and cache eviction. We demonstrate cases where we can completely mitigate leakage by controlling these sources. We then show that enabling each of the sources individually leaks enough to allow website-fingerprinting attacks. In the quantitative analysis, we use the correlation between events related to each source and the measured timing in the attacks as a metric to determine the relative contribution of each source to the specific attack. Our work provides insights into the leakage sources of coarse-grained microarchitectural attacks, aiding the design of secure processor systems as well as more effective attacks and defenses.**

Received 26 April 2025; revised 17 September 2025; accepted 28 September 2025. Date of publication 1 October 2025; date of current version 14 January 2026. This work was supported in part by the Air Force Office of Scientific Research (AFOSR) under Grant FA9550-24-1-0079, in part by the Alfred P. Sloan Research Fellowship, an ARC Discovery Project under Grant DP210102670, in part by the China Scholarship Council (CSC) under Grant 202104910308, in part by the Defense Advanced Research Projects Agency (DARPA) under Grant W912CG-23-C-0022, and in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA under Grant 390781972 and through Project 560392681, and in part by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China under Grant JYB2025XDXM114, and gifts from Cisco and Qualcomm. *(Corresponding authors: Yinqian Zhang; Yuval Yarom.)*

Yusi Feng and Yinqian Zhang are with the Research Institute of Trustworthy Autonomous Systems, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China (e-mail: fengys@sustech.edu.cn; zhangyq3@sustech.edu.cn).

Sioli O'Connel is with the University of Adelaide, Adelaide, SA 5005, Australia (e-mail: sioli.oconnell@adelaide.edu.au).

Xin Zhang is with the School of Software and Microelectronics, Peking University, Beijing 100871, China (e-mail: zhangxin00@stu.pku.edu.cn).

Chitchanok Chuengsatiansup is with the Hasso-Plattner Institute and the University of Potsdam, 14482 Potsdam, Germany (e-mail: chitchanok.chuengsatiansup@hpi.de).

Daniel Genkin is with the Georgia Institute of Technology, Atlanta, NW 30308 USA (e-mail: genkin@gatech.edu).

Yuval Yarom is with Ruhr University Bochum, 44780 Bochum, Germany (e-mail: yuval.yarom@rub.de).

Zhi Zhang is with the Department of Computer Science and Software Engineering, University of Western Australia, Perth, WA 6009, Australia (e-mail: zzhangphd@gmail.com).

Digital Object Identifier 10.1109/TDSC.2025.3617019

***Index Terms*—System security, side-channel attack, website-fingerprinting attack.**

I. Introduction

Over the last two decades, side-channel attacks that leak information through shared computational resources have become a threat to the security of computer systems. Multiple attacks have been designed, exposing secrets such as cryptographic keys [1], [2], [3], [4], [5], [6], address space layout information [7], [8], [9], proprietary hardware designs [10], [11], [12], keystrokes and their timing [13], [14], visited websites [15], [16], [17], [18], [19], [20], [21], data accessed during speculative execution [22], [23], and other secret of private information [24].

While most such attacks target specific components, such as memory caches [4], [6], [25], [26], [27], branch predictors [7], [28], [29], [30], [31], translation lookaside buffers [32], [33], [34], shared buses [35], [36], execution units [37], [38], [39], or GPUs [19], [20], [21], [40], [41], others extract leaked information from coarse-grained measurements of system performance [17], [18], [42], [43], [44], [45]. Such attacks have several advantages. In particular, they can be mounted without fully controlling the microarchitecture of a system or even without fully understanding it.

Similar to attacks that measure individual components, the attack designer models the behavior of the system and uses the model to devise software whose execution time depends on the victim's behavior. For example, Shusterman et al. [18] propose the cache-occupancy primitive as a method to perform website fingerprinting. The primitive repeatedly measures the time taken to iterate through a large buffer. The attacker can detect victim memory accesses by detecting corresponding slowdowns when accessing this large buffer. These slowdowns occur because victim memory accesses evict parts of the attacker's buffer. Then, when accessing the buffer, the attacker must wait for the memory to be served from the slower main system memory.

Typically, the success of the attack is taken as a validation of the model, and thus Shusterman et al. [18] attribute the leakage to cache activity. However, due to the course-grain nature of the measurements, it is not clear if the attribution is indeed justified. In particular, Cook et al. [42] cast doubt on this explanation. They argue that the slowdowns observed by the attacker are due to the operating system halting the current attacker program to handle interrupts caused by the victim's activities.

Such doubts in the understanding of the mechanisms that underlie an attack do not necessarily call into question the validity of the attack. The attack's success in recovering leaked

information proves that the attack is valid, irrespective of whether we understand it. However, such doubts call into question the validity of any countermeasures built on this understanding. For example, cache partitioning schemes are often suggested as a countermeasure for cache-based attacks [10], [18], [25], [27], [46]. If a system deploys such a scheme to mitigate an attack that uses the cache-occupancy primitive, the defense may be ineffective if the primitive recovers information through interrupts rather than through the cache. With an increasing deployment of side-channel countermeasures in widely used software, such as web browsers [47], [48], [49], compilers [50], and the Linux kernel [51], it has become increasingly important to ensure that the foundations of our defenses and theoretical leakage models accurately capture reality.

Thankfully, several works have proposals that address this problem. Gülmezoglu [44] proposes a methodology to solve this problem in the context of website-fingerprinting attacks. They record both the behavior of the browser and its effect on the microarchitectural state of the processor using hardware performance counters. They train machine-learning models to distinguish websites based on these measurements and use machine-learning techniques to uncover which aspects of the measurements were deemed most important by the model. Ge et al. [52] and Cook et al. [42] both propose methodologies that rely on controlling microarchitectural channels to control leakage. Both works use this control to test and validate theoretical leakage models in the contexts of operating system process isolation and website fingerprinting, respectively.

While the work of Gülmezoglu [44] is effective at linking victim behavior to leakage, it does not reveal any insights into how information flows to the attacker. In contrast, the works of Ge et al. [52] and Cook et al. [42] do provide insights into how information flows to an attacker, but they were unable to completely control the flow and eliminate the flow of information. We aim to fill these gaps.

I. Our Contribution

In this work, we introduce a framework for analyzing coarse-grained microarchitectural channels. We demonstrate our framework, analyzing three primitives in the context of website-fingerprinting attacks: cache occupancy [18], loop counting [42], and mwait [53]. For each of the attacks, we identify the leakage source that contributes the most to the attack success and quantify the contribution of each source.

Our framework consists of two parts: qualitative and quantitative analysis. In the qualitative analysis, we first identify four possible major sources of leakage described in the literature and demonstrate: competition on CPU time [54], interrupt handling [42], cache eviction [25], [55], and frequency scaling [21], [56], [57], [58]. To validate that this list is comprehensive, we determine BIOS and operating system settings that allow us to manage each of the sources. Through controlling these settings, we identify cases where we can completely eliminate flow of information to the loop counting and mwait primitives, and to significantly reduce it in other scenarios we investigate. Being able to completely eliminate the leakage shows that all leakage sources contributing to an attack have been identified. We suspect that the remaining source of leakage in the cache-occupancy attacks may be the contention on the main memory [59], [60].

The second step in our qualitative analysis shows that each of the identified leakage sources indeed leaks information. We focus on the loop-counting attack, running on one of the machines where all leakage can be eliminated. We enable each of the leakage sources separately and test for evidence of leakage. We find that leakage through each of these sources alone is sufficient to mount a successful attack that is competitive with the baseline, demonstrating that no single channel is the sole root cause of website-fingerprinting attacks. As an additional contribution, we refine the investigation of leakage through interrupts, identifying that leakage depends on the interrupt type, with some interrupts causing more leakage while others do not. Therefore, the attacker must target cores that handle specific interrupts.

With all sources potentially leaking information, we perform a quantitative analysis of the contribution of each source to the overall leakage. To achieve this, we modify the attack programs to collect performance events related to each source from the operating system and processor. We then correlate the attack counts with the event measurements of each source, using the absolute value of the correlation as a metric for the source's contribution to the attacks. We find that the cache-occupancy attack most strongly correlates with events related to last-level cache activities. For the mwait attack, we identify system interrupts and frequency scaling as the primary leakage sources. Finally, the main contributor to the loop-counting attack is frequency scaling on some processor models, while on others it is a combination of system interrupts and frequency scaling. Additionally, we successfully mount website-fingerprinting attacks by directly using the event data related to each source.

Our results corroborate the claim of Ge et al. [52] that it is impossible to prevent the cross-core leakage in modern processor systems. We have only succeeded doing that for very limited attacks, and even that only on some processors. Moreover, we conclude that focusing on a single channel is not enough for designing system-level defenses. Every channel may leak enough for a successful attack, and consequently all channels need to be considered.

Moreover, some of the leakage effects are counterintuitive. An example is the observation that the loop-counting attack leaks via a cache channel. The loop-counting attack only counts the number of loop iterations that the attacker can perform within a set time, and does not measure any memory accesses. Thus, it could be expected that it will not be affected by cache leakage. Yet, our measurements show that the cache channel leaks enough information to enable a website-fingerprinting attack with a reasonable accuracy. Further investigation demonstrates that victim's activity can evict part of the attacker program from the cache, resulting in instruction cache miss when executing the attack code.

In summary, the contributions of this work are:

- We propose a framework for performing qualitative and quantitative analysis of the root causes of co-located website-fingerprinting attacks, which can also be used to

analyze other attacks based on coarse-grained microarchitectural measurements (Section III).
- As the first step of our qualitative analysis, we identify four major leakage sources and demonstrate how to control each source to eliminate leakage. To the best of our knowledge, this is the first work to completely eliminate leakage in a co-located website-fingerprinting attack by controlling each leakage source (Section IV).
- Our qualitative analysis further validates that all leakage sources contribute to website-fingerprinting attacks. Additionally, we find that leakage through interrupts depends on the type of interrupt handled by the attacker core. We also observe that attacks not involving direct memory accesses can still be influenced by instruction cache misses (Section V).
- In the quantitative analysis, we measure the contribution of each source to the overall leakage, identifying those that contribute the most to each website-fingerprinting attack. Additionally, we successfully mount website-fingerprinting attacks by directly collecting data from each source (Section VI).

II. Background and Related Work

A. Caches

Modern processors use caches, fast memory co-located with processor cores, to reduce the average latency of memory accesses. These caches are arranged into a hierarchy where typically lower-level caches are private to each core and higher-level caches are shared among cores.

Inclusivity: The caches of some processors exhibit an *inclusive* property. That is, the contents of lower-level caches is a strict subset of data stored in higher-level caches. To maintain this property, if data is evicted from a higher-level cache, it must also be evicted from all lower-level caches. Caches that do not exhibit this property are said to be *non-inclusive*.

Attacks on Caches: Since the cache is shared among different processes, the cache state of one process may be affected by the memory access behavior of a different process. Processes may unintentionally encode sensitive information within their memory access behavior which may be recoverable through the use of a cache-based side-channel attack. Several works have shown that cryptography keys [4], [6], [25], [61], keystrokes [13], [14], and address layout [8], [62] can be recovered using cache-based side-channel attacks. Several mitigations have been proposed from the perspective of hiding cache evictions or using intelligent noise injection [63], [64], [65].

B. Interrupts

Interrupts are signals from hardware or software that indicate when a process or event requires immediate attention. Hardware interrupts are triggered by devices such as mice, keyboards, and network cards. Interrupts can also be generated by the CPU itself. For example, processors often feature a timer that can trigger an interrupt after some elapsed time. This feature is often used to implement preemptive scheduling in the operating system. When an Interrupt Request (IRQ) is generated, the CPU will immediately halt the execution of the current thread and invoke the corresponding interrupt handler defined by the kernel. After the interrupt handler completes the execution, the halted program resumes its execution.

Cores can also send interrupts to other cores. For example, when memory mappings are updated within a multi-threaded application, the core that triggers the update will send an interrupt to other cores to invalidate their Translation Look-aside Buffers (TLBs), which is known as a TLB shootdown.

Linux contains two interfaces for querying the number of interrupts that have occurred on the machine: `/proc/interrupts` and `/proc/softirqs`. Both interfaces break down interrupts by the core that received the interrupt and the type of interrupt that was received.

C. Website-Fingerprinting Attacks

Website-fingerprinting attacks use statistical analysis to distinguish which website a victim is visiting. The conventional on-path attack model observes network packets traveling between the user's computer and the network [66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77]. The attacker fingerprints websites from patterns in size and timing of network packets. Corresponding defenses [78], [79], [80], [81], [82] therefore focus on injecting random delays and spurious cover traffic to disrupt these patterns.

Co-located website-fingerprinting attacker executes on the same machine as the victim. Most attacks exploit the contention of shared microarchitecture resources such as cache [15], [16], [17], [18], [19] and GPU [19], [20], [21]. In this paper, we investigate the microarchitectural causes of three recent website-fingerprinting attacks: cache-occupancy [18], loop-counting [42] and mwait [53] attacks.

Cache-Based Website Fingerprinting: Shusterman et al. [18] propose the cache-occupancy channel and demonstrate a website-fingerprinting attack that exploits it. The cache-occupancy channel is based on the observation that when a victim accesses memory, the amount of memory inadvertently evicted from the cache is proportional to the number of different addresses accessed. The authors measure cache-occupancy by repeatedly measuring the time to access a large buffer and demonstrate a website-fingerprinting attack that achieves high accuracy on the Alexa Top 100 websites.

Interrupt-Based Timing Side Channels: Cook et al. [42] extend previous work by proposing that the timing in the cache-occupancy attack [18] is primarily influenced by the time spent handling interrupts, rather than the cache activity itself. To substantiate this claim, they first modify the measurement primitive by eliminating memory accesses, thereby transforming the attack into one that only measures the number of iterations of an empty loop. Subsequently, they demonstrate that a website-fingerprinting attack based on this modified primitive still achieves high accuracy. We refer to this attack as the loop-counting attack.

Zhang et al. [53] exploit recently added *user wait* instructions to mount website-fingerprinting attacks and we refer to this attack as mwait attack. The `umwait` instruction puts a core into an idle state to conserve power and waits for a write to an

address range specified by the `monitor` instruction. The core wakes when either a write to the monitored address is performed, a timeout is reached, or an interrupt occurs. The authors exploit this behavior to measure the number of interrupts within a 10-millisecond window by counting the number of `umwait` instructions that can be executed within the window. Similarly, Rauscher et al. [83] exploit the `tpause` instruction, which transitions the core to the new idle states C0.1 and C0.2. This capability allows for monitoring system activities, particularly interrupts, to facilitate website-fingerprinting attacks.

D. Software-Based Power Side Channels

Traditional power analysis attacks measure the energy consumption of hardware circuits to leak secrets from embedded devices such as smart cards [84], [85]. They require physical measurement with specialized equipment and as such require physical proximity to the victim device.

In contrast, software-based power analysis attacks use software to perform energy consumption measurements. These attacks operate under a different attack model that better matches attacks on mobile phones, desktop computers, or servers. This model trades the requirements of physical proximity and specialized equipment for code execution on the device. Several works have mounted software-based power analysis attacks using the *Running Average Power Limit* (RAPL) interface on modern x86 CPUs to mount website-fingerprinting attacks [86], extract AES and RSA keys [87], and to break KASLR [88]. These attacks have been mitigated by removing access to the RAPL interface from unprivileged users [89], [90].

In addition to directly measuring the energy consumption, several works have proposed techniques that exploit *Dynamic Voltage and Frequency Scaling* (DVFS) to indirectly measure the energy consumption. DVFS is a power management technique that dynamically adjusts processor frequency to keep the processor within its power and thermal limits. Wang et al. [56], [57] were the first to exploit DVFS behavior to measure CPU energy consumption and to distinguish between different instructions and operands. They use this capability to break several cryptographic systems, to measure the energy consumption of other devices in the system, and to reveal parts of rendered webpages. Furthermore, DF-SCA [91] was the first work to exploit frequency scaling for website fingerprinting attacks by reading frequency values from unprivileged access to the `cpufreq` interface. In addition, Taneja et al. [21] demonstrated the widespread presence of frequency- and power-based side channels. They mount several attacks, including website fingerprinting, on a wide range of devices.

III. Experimental Framework

Before presenting the details of our experiments and analysis, we first outline the experimental framework employed to assess the sources of leakage in co-located website-fingerprinting attacks. As illustrated in Fig. 1, our framework is divided into two components: qualitative and quantitative analysis. In the qualitative analysis, we identify the main leakage sources of each attack. We begin by identifying the main potential sources of leakage and demonstrate how to control and eliminate each source. We successfully mitigate leakage in several scenarios (Section IV). Subsequently, we validate that each identified source indeed contributes to information leakage (Section V). For the quantitative analysis, we measure the contribution of each leakage source to the overall leakage in each attack (Section VI).

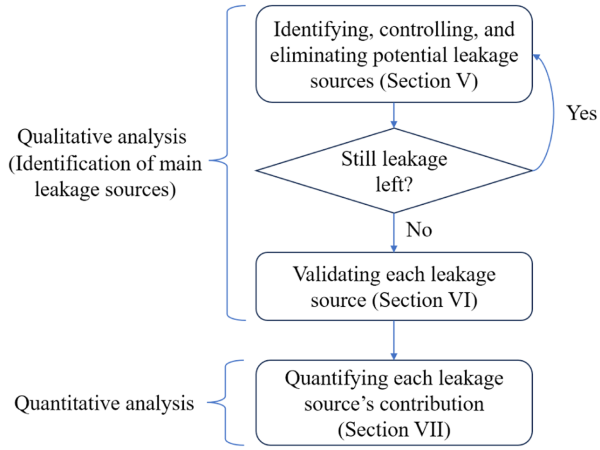


Fig. 1. Our proposed framework for systematically identifying, validating, and quantifying leakage sources.

In the qualitative analysis, website-fingerprinting attacks are used as a proxy to gauge the level of leakage, where high classification accuracy reflects significant leakage capability. In the quantitative analysis, we collect data associated with each leakage source during the execution of the attacks and examine the correlation between the attack data and data from each leakage source. In this section, we first outline our experimental setup (Section III-A) and describe the website-fingerprinting attacks employed in this paper (Section III-B), followed by an description to our framework, including both qualitative (Section III-C) and quantitative (Section III-D) analyses.

A. Experimental Setup

Table I summarizes the hardware and software configurations of the computers we use in our experiments. Experiments in Section V are performed only on our i9-12900KF and i7-7700 machines as we can reliably remove leakage from these machines but not from our i5-8259 U machine.

B. Attack Descriptions

In this section, we investigate the microarchitectural causes of three recent website-fingerprinting attacks: cache-occupancy [18], loop-counting [42], and mwait [53]. We begin by describing the threat model of these co-located website-fingerprinting attacks, followed by the presentation of the pseudo-code. Finally, we explain how we collect and report the measurements for these attacks.

Threat Model: Co-located website-fingerprinting attackers execute on the same machine as the victim and exploit contention for shared microarchitectural resources. The attack can occur

TABLE I
SYSTEM CONFIGURATIONS

CPU	Intel i9-12900KF	Intel i5-8259U	Intel i7-7700
Memory	4× 16 GB DDR5 4800 MT/s	2× 16 GB DDR4 2400 MT/s	1× 4 GB DDR3 1333 MT/s
Motherboard	MSI Z690-A WIFI	NUC8BEB	ASUS B150M-A D3
Storage	480 GB Western Digital SATA SSD	256 GB Silicon Power NVMe SSD	120 GB Patriot Burst SATA SSD
OS	Ubuntu 22.04.1 LTS	Ubuntu 22.04.2 LTS	Ubuntu 22.04.1 LTS
Kernel	Linux 6.2.0	Linux 6.2.12	Linux 6.1.38
Browser	Chrome 113.0.5672.63	Chrome 114.0.5735.198	Chromium 109.0.5414.74
Python	Python 3.10.12	Python 3.10.12	Python 3.10.12
Selenium	Selenium 3.141.0	Selenium 3.141.0	Selenium 3.141.0
C Compiler	GCC 11.4.0	GCC 11.4.0	GCC 11.5.0

```
linked_list = create_linked_list()
for each sample {
  counter = 0
  current = linked_list->head

  begin = time()
  // Traverse each cache line
  while (time() - begin < 5 ms) {

    current = current->next
    counter = counter + 1
  }
  samples[sample] = counter
}

```

Listing 1. Pseudo-code of the cache-occupancy attack.

```

for each sample {
  counter = 0


  begin = time()

  while (time() - begin < 5 ms) {


    counter = counter + 1
  }
  samples[sample] = counter
}

```

Listing 2. Pseudo-code of the loop-counting attack.

either when the user accesses an attacker-controlled website containing malicious JavaScript or WebAssembly code, or when a normal attacker program runs, attempting to learn which other sensitive sites the user is visiting simultaneously. To gather more microarchitectural data, we execute the attacker program written in C on the victim's machine. Furthermore, our attack operates in a closed-world scenario, where we assume knowledge of the start time for each website browsing session.

Pseudo-Code: We present the pseudo-code for the cache-occupancy, loop-counting, and mwait attacks used in our paper in Listings 1, 2, and 3, respectively. We implement each using a mix of inline assembly and C. As shown in Listing 1, for the cache-occupancy primitive, we use a linked list, which covers a memory buffer with the size of the last-level cache. Each element in this list is a double pointer that points to the address of a different cache line. This setup creates a chain where each cache line's content holds the address of the next cache line to be accessed. We use a pointer chasing to traverse the memory buffer

```

for each sample {
  counter = 0


  begin = time()

  while (time() - begin < 5 ms) {
    UMONITOR;
    UMWAIT;
    counter = counter + 1
  }
  samples[sample] = counter
}

```

Listing 3. Pseudo-code of the mwait attack.

of the last-level cache size. The traversal order is randomized, and we link the last element of the list back to the first to form a circular list. In each measurement period of five milliseconds, we traverse along the list accessing each of the elements.

We deviate from previous works which iterate through the entire list for every loop iteration [17], [18], [42]. We instead access a single element per loop iteration. This avoids the artificial limit on the precision of the attack, which would otherwise be limited to the time required to traverse the list.

Recording a Trace: To record a trace, we launch and navigate a browser to a randomly chosen website. Concurrently, we execute the code of one of the attacks. We let the attack collect a total of 3 000 samples, with a delay of five milliseconds between successive samples, for a total collection time of a little over 15 seconds. The extra time is due to the overhead of collecting samples.

To guarantee that we capture the entire page load, we start the attack and wait for 50 milliseconds, after which we navigate to the target website. We ensure that the browser always starts from the same configuration by visiting `example.com` before we navigate to the real website.

Collecting Traces: Before collecting any traces, we first visit each of the considered websites, allowing them to completely load and display for a period of 30 seconds. This ensures that every trace is collected with the same browser cache state and is more representative of a real-world scenario where a victim is unlikely to be visiting a website for the first time.

We collect 100 traces for each of the Alexa Top 100 websites, thus a total of 10 000 traces. Since Amazon has shut down Alexa ranking sites, we use the same list of sites as Cook et al. [42]. Six of the websites have been shut down or had their domain names

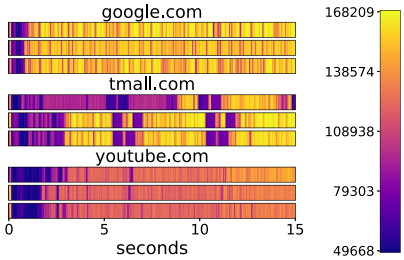


Fig. 2. A heatmap of cache-occupancy attack on a baseline system (i7-7700 machine, website classifier accuracy: 95.6±1.0%).

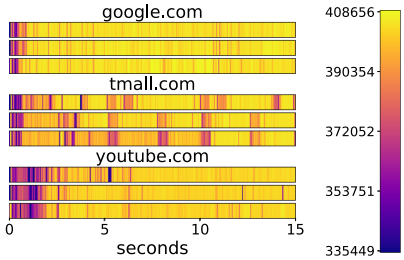


Fig. 3. A heatmap of loop-counting attack on a baseline system (i7-7700 machine, website classifier accuracy: 94.3±0.8%) .

changed. We therefore replace these with the six non-sensitive websites following the top 100 in Alexa's list. To prevent bias due to collection order, we randomize the order of websites we visit. We automate the collection of these traces with a Python program that uses Selenium to directly control the web browser and Python's `threading` library to invoke the attacker.

Training a Classifier: We assume the ability to mount a website-fingerprinting attack as a proxy regardless of whether a system has information leakage or not. As Cook et al. [42], we use a website-fingerprinting attack as a proxy for the level of leakage. We implement the same Long Short-Term Memory (LSTM) model featuring 32 units, consistent with Cook et al. [42] and Shusterman et al. [18], and maintain identical hyper-parameters. Our evaluation process involves dividing the data into ten folds, designating one fold as the test set while distributing the remaining data into an 81% training set and a 9% validation set. We execute this procedure iteratively for each fold and calculate the average accuracy over all ten folds to determine the final model accuracy.

Measuring Time: We measure time using the `rdtscp` instruction, which measures *reference cycles*. These cycles occur at a fixed rate irrespective of the actual frequency of the processor [92, Vol. 3B §18.17] and are ideal for measuring time with high precision. We convert five milliseconds to a fixed number of reference cycles ahead of time and wait for that number of reference cycles to elapse.

Heatmaps: Throughout this paper, we visualize the data collected using one-dimensional heatmaps in the style of Shusterman et al. [18]. We use a linear color map with the minimum and maximum values set at the 2.5% and 97.5% percentiles of the data. We note that while this removes outliers and makes the visual features of each heatmap easier to identify, it also means that color scales are different in different figures.

Figs. 2, 3, and 4 show examples of heatmaps for each of the attacks. The horizontal axis shows samples over time. The color of each position corresponds to the number of iterations of the inner loop. Lighter colors indicate more iterations while darker colors indicate fewer iterations.

Hardware Performance Counters: Hardware Performance Counters (HPCs) measure processor performance parameters such as instruction cycles, cache hits, cache misses, branch mispredicts, and more [92, Vol. 3B §20]. Users can directly access these counters from user space using the `rdpmc` instruction. In some of our experiments, we program these counters to measure a performance parameter then we record the contents of these counters before and after each five-millisecond measurement period.

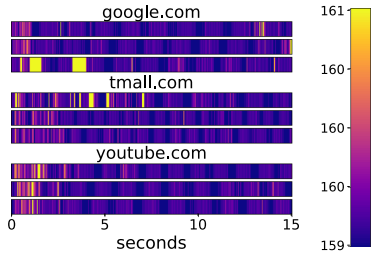


Fig. 4. A heatmap of mwait attack on a baseline system (i9-12900KF machine, website classifier accuracy: 47.6±7.1%).

C. Qualitative Analysis

The aim of the qualitative analysis is to identify the main leakage sources for each attack. First, we identify four potential leakage sources by reviewing existing literature. To validate whether these sources are indeed the main contributors, we control and eliminate each leakage source based on insights from technical blogs and Intel white papers, ensuring that all leakage is eliminated, at least in specific scenarios. In cases where leakage persists after eliminating several sources, we hypothesize new potential leakage sources and conduct experiments to validate these assumptions. Next, leveraging the ability to eliminate all leakage in specific scenarios, we apply the control variable method by configuring the system to eliminate all other leakage sources while leaving one intact. This enables us to test whether each source actually leaks information.

Following Cook et al. [42], we use a website-fingerprinting attack as a proxy for the level of leakage, with high recognition accuracy indicating strong leakage capability. For the first step of our qualitative analysis, we begin with four known channels: competition with co-resident threads for CPU time [54], interrupt handling [42], frequency scaling [21], [56], [57], [58], and eviction from the cache hierarchy [25]. We control each leakage source by modifying BIOS and operating system settings, successfully eliminating information leakage for the loop-counting and mwait attacks on both i7-7700 and i9-12900KF machines. This means that after eliminating these sources, the attacks achieve almost random recognition accuracy. In the second step, we start with a system configuration from the first step that

completely eliminates all leakage and then enable each channel individually. We find that with each channel enabled, the loop-counting attack can still be launched successfully, indicating that all identified channels contribute to the loop-counting attack.

D. Quantitative Analysis

Identifying the main leakage sources enables the measurement of each source's contribution. The aim of the quantitative analysis is to quantify the relative contribution of each channel to the overall attack, which aids in better understanding the root causes of the leakage observed by the attacker.

Methodology: To quantify the leakage, we conduct website-fingerprinting attacks to collect attack data. Simultaneously, we record several system and processor performance events associated with each leakage source. We then compute the Pearson correlation coefficient between the attack data and each leakage source. This coefficient indicates the contribution of each leakage source to the total leakage observed by the attacker. We report the average and standard deviation of the coefficient across all 10,000 traces.

We modify the inner loop of each attack to record system and processor performance events using kernel interfaces and hardware performance counters. For each five-millisecond sample, we record the number of loop iterations, the number of interrupts delivered to the attacker core, the total length of handling interrupts in execution experienced by the attacker, the attacker core frequency, and the number of cache misses of the attacker core. We perform all of these measurements on an unmodified system. In the following, we list the system and processor performance events used to monitor each leakage source.

Measuring Interrupt Count: We measure the number of interrupts delivered to the attacker core using the `/proc/interrupts` and `/proc/softirqs` interface. We measure before and after each five-millisecond sample and compute the difference.

Measuring Interrupt Length: While we can easily measure the total number of interrupts, there are no readily available interfaces to measure the total handling time of interrupts with fine granularity. Instead, we adopt the technique from Zhang et al. [93]. The idea is based on the observation that when an interrupt is handled by the kernel and control returns to the attacker program with no special privileges, it causes two observable effects. First, the processes can notice a large gap between consecutive clock reads. Second, the CPU clears the data segment registers (e.g., `DS`, `ES`, `FS`, and `GS`). Therefore, we set the `GS` register to 1 and modify the inner loop of each attack to compare the previous clock measurement with the current clock measurement. If we observe a time difference exceeding 500 nanoseconds but less than 10 microseconds, and the `GS` register is reset to 0, we classify the time difference length as interrupt handling time.

Measuring Frequency: We measure the `CPU_CLK_UNHALTED.THREAD_ANY` and `CPU_CLK_UNHALTED.REF_TSC` performance events. The former counts the actual cycles that the core executes, whereas the latter measures wall-clock time as a count of nominal cycles ticking at a model-specific frequency. These events have similar functions to the `APERF` and `MPERF` Model Specific Registers (MSRs) but can be accessed directly from the user space. The ratio between the two registers (or between the two performance events) gives the average frequency of the processor over a given period of time [92, Vol. 3B §14.2].

Measuring Cache Activity: We measure the activity of the data and instruction caches separately by measuring the `CYCLE_ACTIVITY_CYCLES_L3_MISS` and `FRONTEND_RETIRED.L1I_MISS` performance events. The former measures the number of cycles that a load instruction is stalled due to a data cache miss, while the latter measures the number of instructions that miss in the L1I cache.

IV. Identifying, Controlling, and Eliminating Potential Leakage Sources

In this section, we present the first step of the qualitative analysis—identifying the main potential channels that contribute to information leakage in website-fingerprinting attacks and demonstrating that we can control them. We start with four known channels, namely: competition with co-resident threads on CPU time [54], interrupt handling [42], frequency scaling [21], [56], [57], [58], and eviction from the cache hierarchy [25]. We identify the levers and controls that the BIOS and the operating system provide for controlling each of the channels. Last, we evaluate the efficacy of these controls in preventing flow of information through the channel for each of the attacks we investigate. Our controls completely eliminate information leakage for the loop-counting and mwait attacks on i7-7700 and i9-12900KF machines, and significantly reduce the leakage for the cache-occupancy attack.

A. CPU Time Contention

To eliminate the contention on the CPU time, we isolate the attacker from every other program on the machine. Specifically, we designate one core as the attacker core and only allow the attacker to execute on this core; all other processes are forced to execute on the remaining cores. We perform this isolation via the `isolcpus` Linux kernel parameter. This parameter prevents the kernel from automatically scheduling any thread on the specified core. We then use the `taskset` utility to explicitly schedule the attacker onto this core.

To further avoid the contention on core resources, we disable hyperthreading. Combined with core isolation, this ensures that the victim never executes on the same core as the attacker. Hence, the attacker cannot observe the effects of the victim execution on the microarchitectural state of the core.

B. Interrupts

Isolating cores with `isolcpus` will additionally move the interrupt handling to non-isolated cores. However, in several cases throughout the paper we need to manually move interrupts. To do so, we use the `irqbalance` utility and the `/proc/irq/[id]/smp_affinity` interface. While many

interrupts can be moved, some cannot. We now discuss how to remove the remaining interrupts.

Tickless: To remove timer interrupts, we put the kernel into a so-called *tickless* mode. When running under this mode, timer interrupts are disabled if the core has only a single thread scheduled for execution. To use this mode, we compile the kernel with the `CONFIG_NO_HZ_FULL` compile time option and then use the `nohz_full` kernel parameter to specify the list of cores that can enter the tickless mode.

Tickless Constraints: Using tickless mode to 'disable' timer interrupts has three constraints on the attack that we must consider. First, we must never execute a system call that would cause the kernel to directly issue a timer interrupt. In particular, we must never issue a system call to sleep. Second, our attack implementation must be limited to a single thread. Finally, we must never allow another thread to be scheduled onto the attacker core. This includes our experimentation automation infrastructure. The last two conditions are required because without them there would be multiple processes scheduled on the core, preventing the tickless mode.

Verifying Constraints: We verify the first constraint through manual code inspection. We ensure that the attack never executes anything that could cause it to go to sleep. In cases where we might need to sleep, we replace the call with a busy loop that uses `rdtscp` to detect the elapsed time. We verify the second and third constraints through the use of the `/sys/kernel/debug/sched/debug` interface, ensuring that we observe only a single thread on the attacker core.

RCU Callback Offloading: Read-Copy-Update (RCU) is a synchronization mechanism that allows many Linux kernel threads to concurrently read from a shared data structure. Callbacks can be registered with the kernel to be executed when a thread releases its reference to the structure. To minimize the latency of the operation that releases the last reference, the callback may be offloaded to another core. To remove the interrupt associated with this offloading, we use the `rcu_nocbs` kernel parameter to specify the list of cores that cannot be used for RCU callback offloading.

Verifying Absence of Interrupts: We verify that we have eliminated the effect of interrupts through the use of the `/proc/interrupts` and `/proc/softirqs` interfaces. These interfaces provide the total number of interrupts that have been handled by the kernel since power on. The reported interrupts are separated by types of interrupts and cores that handle the interrupts. We access these interfaces at the start and end of each trace, and compute the difference to obtain the number of interrupts during the attack. In addition, we use the `CPU_CLK_UNHALTED.RING0_TRANS` performance counter to monitor the number of transitions from user to kernel mode.

We find that both methods almost always report zero interrupts and zero transitions from user to kernel mode. In rare cases, 0.48% of experiments, we observe some interrupts. However, we do not observe many interrupts, and they do not appear to be correlated with the victim behavior. For these reasons, we consider the effects of interrupt handling to have been eliminated from our experiments.

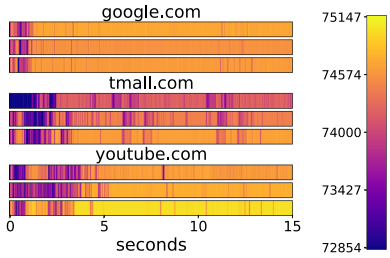


Fig. 5. A heatmap of cache-occupancy attack after removing leakages (i7-7700 machine, website classifier accuracy: 93.9±0.5%).

C. Frequency Scaling

Modern processors try to maintain the balance between frequency, heat dissipation, and power budget. The main tool for maintaining this balance is a mechanism for changing the operating frequency and voltage, known as Dynamic Voltage and Frequency Scaling (DVFS). Recent works have shown how frequency scaling can be exploited to leak information [21], [56], [57], [58], [91]. As before, we aim to control leakage through frequency scaling by demonstrating that we can completely remove the effect of DVFS on the attacker core.

We start by disabling two BIOS features that allow the processor to change its frequency, TurboBoost and SpeedStep. We then instruct the operating system to fix the frequency to a low enough value, ensuring that the processor always remains within its allocated operating power budget and that the processor will never exceed its maximum thermal threshold. We achieve that using the `cpufreq-set` utility to set the minimum and maximum allowed frequencies of all cores to half of the base frequency of the processor (1.8 GHz). Further, we set the power governor to `performance` and write a value 0 to the `/dev/cpu_dma_latency` interface to prevent the processor from entering any low power states.

D. Cache

We eliminate the possibility of the victim evicting attacker memory by using Intel Cache Allocation Technology (CAT) to partition the last-level cache. Specifically, we assign half of the ways to the attacker core and the other half to the remaining cores. Partitioning the last-level cache in this way prevents the victim from unintentionally evicting attacker memory from the cache, severing the channel between the victim and the attacker. However, as we describe below, this does not completely close all memory-based channels, and some leakage remains.

E. Validating Leakage Elimination

We now move to measuring the effects of using our techniques on the attacks.

Loop Counting and Mwait: We succeed in completely eliminating leakage on our i7-7700 and i9-12900KF machines, with similar results on both machines. Figs. 6 and 7 show heatmaps recorded using the loop-counting and mwait attacks after we apply the steps to eliminate leakages. For loop-counting, most

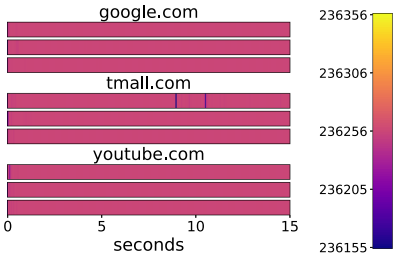


Fig. 6. A heatmap of the loop-counting attack after removing leakages (i7-7700 machine, website classifier accuracy: 3.1±0.7%).

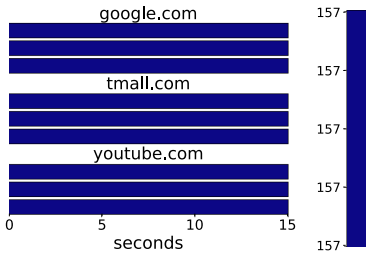


Fig. 7. A heatmap of mwait attack after removing leakages (i9-12900KF machine, website classifier accuracy: 1.0±0.0%).

of the data changes almost only by 1, from 236255 to 236256. For the mwait, we observe that the heatmaps consist of a single color. This is not an artifact of the visualization; all samples within these traces are completely identical, suggesting that we have completely eliminated leakage from the loop-counting and mwait attacks. This is in stark contrast to Figs. 3 and 4, where we can clearly distinguish websites just by looking at the heatmaps.

As described in Section III-B, we first conducted the above experiments with the attacker program written in a mix of inline assembly and C, which provides better isolation and allows us to gather more microarchitectural data. We then validated whether our isolation mechanisms also eliminate leakage in the browser environment by running the attacker as malicious JavaScript code hosted on an attacker-controlled website. Following Cook et al. [42], while the victim was accessing websites, we launched a separate browser that visited the attacker-controlled site executing the loop-counting attack. The only difference is that we constrained the attacker browser to the isolated core. Fig. 8 shows traces that appear entirely random across websites and samples on the i9-12900KF. This randomness arises because, even in isolation, the browser spawns multiple processes, and context switches with their associated interrupts introduce noise. The resulting accuracy is 4.3±0.4%. For comparison, running the same experiment without isolating any leakage source on the same machine yields an accuracy of 88.5±1.5%. These results indicate that our isolation mechanisms effectively eliminate leakage from the loop-counting attack in the browser environment, thereby broadening the applicability of our conclusion.

i5-8259 U: With our i5-8259 U machine, we are still able to classify websites with 71.5±2.2% accuracy using the loop-counting attack. We believe this is because our control over each channel was ineffective. We note that although we have partitioned the last-level cache using CAT, we still observed

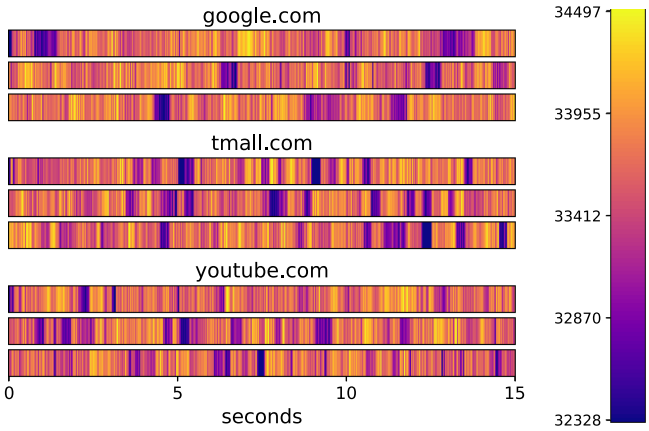


Fig. 8. A heatmap of the loop-counting attack collected in the browser environment using JavaScript after removing leakages (i9-12900KF machine, website classifier accuracy: 4.3±0.4%)

cache misses. Further, we note the processor frequency can still change despite setting a fixed frequency.

Cache Occupancy: Unfortunately, we cannot completely eliminate the leakage measured by the cache-occupancy attack. Fig. 5 shows a heatmap for the cache-occupancy attack when executed on the same system configuration with leakage elimination. We note that we can still clearly distinguish different websites from the obtained traces. Furthermore, when we train a classifier on these traces, we achieve an accuracy of 91.9% in identifying websites. This suggests that the cache-occupancy attack measures some additional channels that cannot be measured by the loop-counting attack.

F. Remaining Leakage Sources

Having identified that a leakage remains detectable in the cache-occupancy attack, we now seek to provide more insights into the cause of this leakage. Several works have shown that an attacker can measure contentions in inter-core interconnects [35], [36], [94], last-level cache slice accesses [35], [36], [94], memory controllers [95], and DRAM [59].

Experiment Design: While controlling all cross-core channels is outside the scope of this work, we perform the following experiment to clarify whether the remaining channel is formed through the execution of memory access instructions themselves or whether it is formed through the contention in these off-core components serving the memory accesses. We modify the cache-occupancy attack such that all of its memory fits within the private cache of the attacker core. Specifically, we change the size of the linked list to 28 KB, which is slightly smaller than the size of the L1 cache. We preload the linked list into the cache before mounting the cache-occupancy attack.

Results: We obtain an accuracy of 1.43±0.3% when we mount the attack under this configuration. This result suggests that the remaining leakage can be entirely explained by off-core memory accesses contending for off-core resources. We leave the task of enumerating, controlling, and investigating these off-core channels to future work.

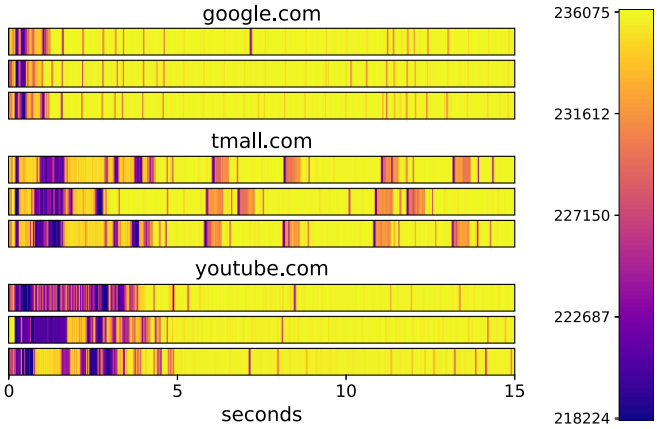


Fig. 9. Traces recorded on i7-7700 with a configuration that allows leakage from time contention (website classifier accuracy: 95.4±0.5%).

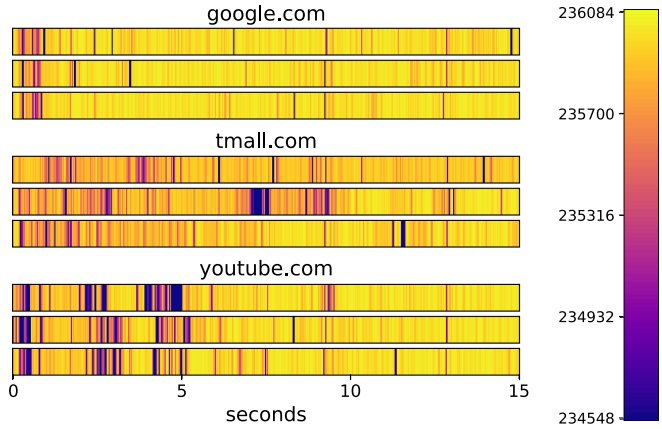


Fig. 10. Traces recorded on i7-7700 with a configuration that allows leakage from interrupts (website classifier accuracy: 95.3±0.8%).

V. Validating Each Leakage Source

The ability to eliminate leakage enables us to investigate the second step of our qualitative analysis: testing which channels actually leak information. We start with a system configuration obtained from Section IV that completely eliminates all leakage, and enable each channel individually. That is, for each channel, we configure the system so that leakage can only occur through that channel. For that, we apply the steps to control the leakage from all other channels, but do not control the leakage from the evaluated channel. We then mount a website-fingerprinting attack as described in Section III-B with this system configuration; if we can distinguish websites, we conclude that the channel does leak information. We find that all identified channels are measurable by the loop-counting attack. Monitoring these events related to each channel does not require privileges.

To avoid non-inclusive caches, which interfere with our validation of cache leakage, we focus on the i7-7700 machine. This also limits us to only using the loop-counting attack because the architecture does not support the `umwait` instruction that underlies the mwait attack.

A. CPU Time Contention

First, we test whether time and core resources contention is a channel that leaks information. We skip the steps taken in Section IV-A and manually move all interrupts away from the attacker core. We then mount a website-fingerprinting attack under this configuration.

Fig. 9 shows the results of this experiment. We achieve an accuracy of 95.4±0.5% when we train a classifier on this configuration.

B. Interrupt Leakage

Next, we test whether the interrupt handling is a channel that leaks information. We skip the steps taken in Section IV-B and then mount the website-fingerprinting attack. Since we are still isolating the core with `isolcpus`, which moves interrupts away from isolated cores, we measure no leakage in this configuration. Instead, we manually move all interrupts to the attacker core.

When we perform this experiment and train a classifier on this altered configuration, we achieve an accuracy of 95.3±0.8%. Fig. 10 shows a heatmap of the resulting traces in which we can clearly distinguish websites from each other.

C. Per-Device Interrupt Leakage

We note that the default configuration on our system balances the load of interrupts across each of the cores. That is, interrupt requests from each device on the system are handled by separate cores. For example, the operating system may arbitrarily assign the first core to handle all interrupt requests from the network interface and the second core to handle all interrupt requests from the sound interface.

The combination of this behavior along with the results of our previous experiments suggest that the choice of core to mount an attack on may be important. If the attacker mounts an attack on a core that does not handle any interrupts, they will not see any leakage. Similarly, if the attacker mounts an attack on a core that only handles interrupts from devices that are uncorrelated with victim behavior, they may see little to no leakage.

Experiment Design: We start with a system configuration obtained from Section IV that completely eliminates all leakage. Next, we move all interrupts from a specific device to the attacker core and move all other interrupts to another core. We then mount a website-fingerprinting attack as described in Section III-B. We repeat this experiment for each device in our system: the USB controller (USB), NVMe storage drive (Storage), networking, graphics, sound, and the system management engine (Management).

Results: Table II shows the results of this experiment. We find that the accuracy varies significantly depending on which device interrupts requests are delivered to the attacker core. For example, interrupt requests from the network achieve the highest accuracy at 69.1%, followed by interrupts from the storage device at 53.1%. The relatively low accuracy for sound-related

TABLE II
ACCURACY OF THE LOOP-COUNTING ATTACK ON OUR I7-7700 MACHINE WHEN WE DELIVER INTERRUPTS FROM A SPECIFIC DEVICE TO THE ATTACKER CORE

Interrupt	Accuracy
Networking	$69.1 \pm 1.3\%$
Storage	$53.1 \pm 1.1\%$
Graphics	$51.6 \pm 1.0\%$
Sound	$4.9 \pm 0.5\%$
Management	$1.6 \pm 0.3\%$
USB	$1.5 \pm 0.3\%$

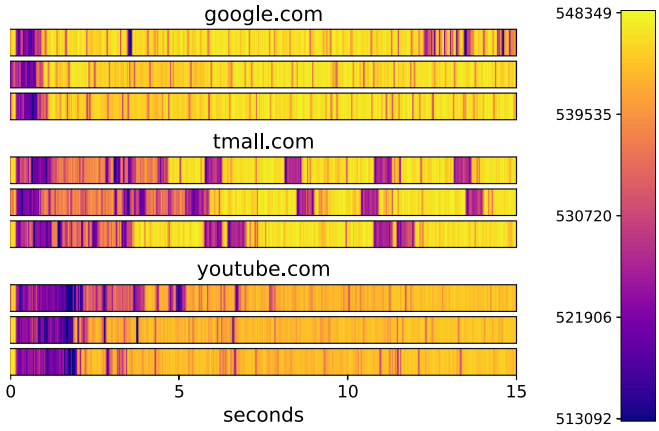


Fig. 11. Traces recorded on i7-7700 with a configuration that allows leakage from frequency changes (website classifer accuracy: 95.3±0.5%).

interrupts can be explained by the fact that many websites, such as `google.com` and `baidu.com`, do not trigger sound activity. Since the system management engine does not interact with web traffic, we observe almost no management-related interrupts. In addition, because we interact with our machines exclusively through remote access, no USB interrupts are generated, and thus no leakage occurs from that source.

D. Frequency Leakage

We now analyze the leakage through changes in frequency due to DVFS. Similar to the previous section, we apply the configuration from Section IV but skip the steps from Section IV-C that would eliminate the leakage from DVFS. We then mount a website-fingerprinting attack as described in Section III-B.

Fig. 11 shows heatmaps generated from this experiment. We find that we can clearly distinguish websites from each other. When we train a classifier on traces recorded with this configuration, we achieve an accuracy of 95.3±0.5%, demonstrating that frequency scaling is a channel that leaks information.

E. Cache Leakage

Finally, we investigate the effect of the cache. We repeat the same process from the previous sections. That is, we apply the configuration from Section IV and skip the steps that eliminate cache leakage from Section IV-D. We then mount the website-fingerprinting attack described in Section III-B.

Fig. 12. Traces recorded on i7-7700 with a configuration that allows leakage from cache evictions (website classifier accuracy: 95.2±0.6%)

We find that a classifier trained on traces recorded with this configuration can easily distinguish websites with an accuracy of 95.2%. Likewise, the heatmaps (Fig. 12) show that websites can be clearly distinguished.

Cache Attacks with no Memory Accesses: The loop-counting attack that we use basically counts the number of iterations of an empty loop that the attacker can perform in a given time interval (Listing 2). As the attack does not access memory, one could expect there to be no correlation between the state of the cache and the execution time of the loop-counting attack. Thus, the observation that the cache channel conveys enough information for carrying the attack out is somewhat counterintuitive. To explain the observation, we now investigate the interplay between cache and loop-counting attack.

Measuring Cache Misses: We repeat the experiment from the previous section, but we instrument the loop-counting attack to record the number of L1I and L1D cache misses. We do this by programming performance counters to record the number of `FRONTEND_RETIRED.L1I_MISS_PS` and `MEM_LOAD_RETIRED.L1_MISS_PS` events during the attack.

As expected, we observe that there are no L1D cache misses. Since the loop-counting attack does not perform any memory accesses, beyond what is required to record samples, there are simply no memory accesses to miss the L1D cache. This leaves L1I cache misses, and indeed we do find a non-zero number of L1I misses.

Root Cause Analysis: We hypothesize that the root cause of this leakage is due to the inclusive shared last-level cache (LLC) of the i7-7700. When a cache line is evicted from the shared inclusive LLC, it is also evicted from the private caches of each core—the L2, L1D, and L1I caches. Furthermore, the LLC does not distinguish between data and instructions.

If the victim were to access a significant amount of memory, they may inadvertently evict cache lines from the LLC that contain the instructions for the loop-counting attack. This causes delays in the loop-counting attack because the instructions are no longer available in the L1I cache and instead must be fetched from system memory.

TABLE III
CORRELATION BETWEEN ATTACK ITERATIONS AND VARIOUS SYSTEM AND PERFORMANCE EVENTS ON EACH OF OUR THREE MACHINES

		Cache Occupancy			Loop Counting			Mwait
	Machine	**i9-12900KF**	**i5-8259U**	**i7-7700**	**i9-12900KF**	**i5-8259U**	**i7-7700**	**i9-12900KF**
Baseline[1]	**Interrupt Count**	0.04±0.01	0.01±0.03	−0.03±0.01	−0.04±0.02	0.05±0.04	0.02±0.02	0.04±0.02
	Interrupt Length	−0.24±0.07	−0.24±0.09	−0.09±0.02	−0.25±0.04	−0.34±0.09	−0.13±0.03	0.06±0.03
	Frequency	0.70±0.13	0.37±0.33	0.33±0.07	**0.77±0.06**	**0.57±0.08**	**0.84±0.11**	**0.42±0.14**
	L1I Cache	−0.15±0.06	−0.13±0.04	0.01±0.03	0.04±0.05	−0.13±0.05	0.03±0.03	−0.15±0.15
	L3 Cache	**-0.99±0.01**	**-0.76±0.11**	**-0.84±0.13**	−0.11±0.07	−0.46±0.09	−0.26±0.06	−0.07±0.05
Single Core[2]	**Interrupt Count**	−0.23±0.10	−0.55±0.15	−0.13±0.06	−0.23±0.08	−0.51±0.10	−0.15±0.06	**0.42±0.12**
	Interrupt Length	−0.37±0.10	−0.46±0.17	−0.15±0.05	−0.36±0.07	−0.49±0.11	−0.22±0.07	0.34±0.14
	Frequency	0.70±0.13	0.39±0.34	0.32±0.07	**0.78±0.06**	0.56±0.08	**0.83±0.07**	0.22±0.13
	L1I Cache	−0.20±0.10	−0.45±0.15	−0.15±0.04	−0.20±0.08	−0.43±0.09	−0.17±0.07	0.28±0.13
	L3 Cache	**-0.99±0.01**	**-0.74±0.10**	**-0.82±0.16**	−0.25±0.08	**-0.61±0.07**	−0.30±0.06	0.08±0.10

[1] Baseline refers to the default interrupt configuration.
[2] Single Core refers to the configuration where all interrupts are delivered to the attacker core.

Non-Inclusive Caches: To test this hypothesis, we repeat the same experiment on our i9-12900KF machine. This machine has a non-inclusive LLC, meaning that when a cache line is evicted from the LLC, it is not evicted from the private caches of each core. When we do this, we indeed find that the number of L1I cache misses is zero and that there is little to no cache leakage measured by the loop-counting attack.

VI. QUANTIFYING EACH LEAKAGE SOURCE'S CONTRIBUTION

The observation from the qualitative analysis, that multiple leakage sources carry enough information to enable efficient website-fingerprinting attacks, calls into question the conclusions of past works regarding the root cause of the leakage they observe [18], [42]. Recall that each attack is a combination of leakage sources. In this section, we develop a metric for quantitatively analyzing the relative contribution of each channel to the overall attack. We apply the system and processor performance events outlined in Section III-D to monitor each source. We then present the results of our experiments and conclude by discussing the implications of our findings.

A. Methodology

We follow the methodology outlined in Section III-D. Specifically, we modify the inner loop of each attack to record several system and processor performance events associated with each leakage source. We then compute the Pearson correlation coefficient between the attack data and each leakage source. This coefficient indicates the contribution of each leakage source to the total leakage observed by the attacker. We report the average and standard deviation of the coefficient across all 10,000 traces.

Specifically, for each five-millisecond sample, we record the number of loop iterations, the number of interrupts delivered to the attacker core, the total length of interrupt handling experienced by the attacker, the attacker core frequency, and the number of cache misses on the attacker core.

B. Contributions of Sources

We now report the correlation coefficient between each leakage source and the number of iterations of the attack loop for each of the attacks. We designate the third core as the attacker core and perform our experiments on this core. Since the operating system assigns specific cores to handle interrupts from specific devices, we further investigate the impact of interrupts on these attacks under an extreme case where all movable interrupts are moved to the attacker core. The results of the three attacks are shown in Table III.

The Pearson correlation coefficient can take both positive and negative values. A positive value indicates a positive relationship between the leakage source and the attack data. The larger the value observed in the leakage source, the larger the corresponding attack data. For example, with frequency scaling, a higher frequency allows the attack to collect more data within a limited time. A negative value indicates an inverse relationship. For instance, in the cache-occupancy attack, a higher number of L3 cache misses means the attack can access less cache, thereby reducing the amount of attack data. To enable a fair comparison, we report and analyze only the absolute values of the correlation coefficients in the following discussion.

Cache Occupancy: We find that the correlation between L3 Cache and the cache-occupancy attack is the highest of the measured events. This is irrespective of whether the machine features an inclusive (i5-8259 U and i7-7700) or non-inclusive (i9-12900KF) cache or whether interrupts are moved to the attacker core or not. This suggests that the cache is the primary contributor to the leakage measured by the cache-occupancy attack. We note that Frequency and Interrupt Length are also highly correlated with the cache-occupancy attack. Correlation with Interrupt Count increases when interrupts are moved to the attacker core.

Loop Counting: Frequency exhibits the highest correlation across all our desktop machines in the baseline configuration, significantly exceeding that of other sources. When interrupts are redirected to the attacker's core, we observe an enhanced correlation between Interrupt Count and the loop-counting attack. On the i9-12900KF and i7-7700, Frequency continues to dominate in terms of correlation, again substantially surpassing other leakage sources, indicating that frequency scaling serves as the primary leakage source in these machines.

However, when interrupts are redirected to the attacker's core on the NUC (i5-8259 U), the L3 cache exhibits the highest correlation with the loop counting attack (0.61 $\pm$ 0.07), followed by frequency (0.56 $\pm$ 0.08) and interrupt count (-0.51 $\pm$ 0.08). A plausible explanation is that the large number of interrupts forces the CPU to repeatedly load multiple interrupt handlers, incurring increased L1I and L3 cache misses. This effect is likely amplified on the NUC due to its relatively small L3 cache (6 MB) compared with the i9-12900KF (30 MB) and the i7-7700 (8 MB). To further examine this, we measured the correlation on the NUC between Interrupt Count and cache activity under the same configuration. The results show a correlation of 0.91 $\pm$ 0.03 with the L1I cache and 0.75 $\pm$ 0.04 with the L3 cache, supporting our interpretation. Moreover, because the L3 cache is also affected by frequency, the strong correlation between the L3 cache and the loop counting attack reflects the combined influence of interrupts and frequency. Given that frequency itself shows a comparable correlation, we conclude that both interrupts and frequency contribute to the observed leakage on the NUC.

MWAIT: We report the results for our i9-12900KF machine only as it is the only machine to support the `umwait` instruction necessary for the attack. We find that Frequency has the highest correlation with the mwait attack under the baseline configuration and that Interrupt Count shows the highest correlation when interrupts are moved to the attacker core.

Conclusion: As shown in Table III, the cache-occupancy attack most strongly correlates with events related to last-level cache activities, which aligns with the proposal of Shusterman et al. [18]. For the loop-counting attack, we identify frequency scaling as the primary leakage source in most cases, and a combination of system interrupts and frequency scaling in other cases, thereby extending the proposal of Cook et al. [42]. For the mwait attack, we identify system interrupts and frequency scaling as the primary leakage sources, thereby extending the proposal of Zhang et al. [53].

Since we have demonstrated that each leakage source can leak enough information for a successful attack in Section V, it is challenging to prevent cross-core leakage from a system-level redesign perspective, as redesigning all of these microarchitectural channels would significantly impact performance. Instead of solely focusing on redesigning specific side channels, we advocate for mitigation strategies aimed at reducing the coarse-grained measurement capabilities of attacks. This includes methods like randomized timers or noise injection. Notably, injecting noise into the cache source for the cache-occupancy attack, and into the DVFS and interrupt sources for the loop-counting attack, along with the primary contributor identified in Section VI, could be a comparably more effective approach.

Comparison with Cook et al. In previous work, Cook et al. [42] suggest that the primary contributor to the cache-occupancy and loop-counting attacks is leakage from interrupts. They remove one leakage source at a time and observe accuracy decreases, with the removal of the IRQ interrupts leakage source resulting in the highest decrease, from 95.2% to 88.3%. We believe the discrepancy between our results and theirs arises from different methods of measuring contributions, particularly the use of machine learning accuracy as a metric for channel contribution. We note that accuracy can increase even when information is removed. Fig. 3 shows a baseline configuration with an accuracy of 94.3$\pm$0.8%, while Figs. 12,9,10, and 11 show configurations with only one leakage source, achieving no less than 95.2% accuracy. Instead, we use the Pearson correlation coefficient as an estimate of the contribution.

TABLE IV
ACCURACY BASED ON DATA FROM EACH CHANNEL

Data Source	Baseline[1]		Single Core[2]	
	i5-8259U	i7-7700	i5-8259U	i7-7700
Loop Counting	**92.5$\pm$1.0%**	**95.7$\pm$1.0%**	**92.7$\pm$1.2%**	**94.6$\pm$0.8%**
Cache Occupancy	**93.8$\pm$1.0%**	**95.6$\pm$1.0%**	**93.6$\pm$0.8%**	**95.4$\pm$0.7%**
Interrupt Count	83.3$\pm$0.9%	85.3$\pm$1.4%	89.1$\pm$1.6%	87.4$\pm$1.9%
Interrupt Length	77.8$\pm$2.0%	77.9$\pm$2.6%	85.6$\pm$2.0%	87.0$\pm$1.6%
Frequency	**93.4$\pm$0.7%**	**95.5$\pm$0.6%**	**93.3$\pm$1.0%**	**94.7$\pm$0.9%**
L1I Cache	63.5$\pm$2.5%	77.3$\pm$1.9%	88.2$\pm$1.3%	83.3$\pm$2.0%
L3 Cache	87.7$\pm$1.5%	**93.3$\pm$1.0%**	**90.4$\pm$1.2%**	**92.6$\pm$0.9%**

[1] Baseline refers to the default interrupt configuration.
[2] Single Core refers to the configuration where all interrupts are delivered to the attacker core.

VII. DISCUSSION

A. Attacks Via Individual Sources

In this section, we perform novel co-located website-fingerprinting attacks using data from each individual channel, collected in Sections III-D and VI-B. Instead of relying on loop-counting, cache-occupancy, or mwait attacks, we independently monitor each leakage source from corresponding system and performance events at 5 ms intervals and feed the collected data into the same classifier described in Section III.

As shown in Table IV, monitoring different leakage channels individually, rather than relying on timing or coarse-grained system performance measurements, also achieves comparable attack accuracy to loop-counting and cache-occupancy attacks, particularly for the frequency and L3 Cache leakage channels. Frequency shows similar accuracy to loop-counting and cache-occupancy across different settings on both the i5-8259 U and i7-7700. These results further support the contribution of the frequency and L3 Cache leakage channels, as discussed in Section VI-B.

B. Countermeasures

As shown in Section IV, eliminating each leakage source requires major system redesigns, which is not feasible. Furthermore, in Section V, we conclude that focusing on a single channel, or even a few channels, is insufficient when designing defenses. Every channel may independently leak enough information to enable a successful attack, and thus all channels must be considered. Therefore, we recommend mitigation strategies that target the coarse-grained measurement capabilities of attacks, such as randomized timers or the injection of noise, rather than focusing solely on redesigning one or several specific side channels. Combined with the primary contributor identified in

Section VI, injecting noise from the cache source for the cache-occupancy attack, and from DVFS and interrupt sources for the loop-counting attack, could be comparably more effective.

Randomized Timers: Modern browsers have employed several strategies to limit an attacker's observation capabilities by reducing the precision of timers. Two typical techniques are quantized timers and jittered timers. The former, used in Tor Browser, fixes the timer resolution to 100 ms so that the timer output is quantized to multiples of 100 ms. The latter, used in Chrome, adds a small random perturbation to the returned value, typically within 0.2 ms. However, as shown by Cook et al. [42], increasing the timer resolution or adding small random noise is insufficient to resist attacks, since the adversary can still achieve an accuracy of 86.0% to 96.6% under these defenses. To address this, Cook et al. [42] propose a randomized timer that increases monotonically with random increments at random intervals. This design causes the reported time to deviate randomly from the real time by up to about 100 ms, achieving 1.0% top-1 accuracy and 5.1% top-5 accuracy for loop-counting attacks. While the randomized timer is effective in mitigating website fingerprinting attacks, it may negatively affect applications that require high-resolution timers, such as online games.

Besides, even if randomized timers are deployed in the browser, prior work has shown that attackers can bypass such defenses. For example, Kim et al. [96] demonstrate how to construct a high-resolution timer without relying on browser-provided clocks, while Katzman et al. [97] amplify subtle timing differences to make them observable even with coarse-grained timers. A feasible alternative for a website-fingerprinting attacker is to repurpose the loop-counting attack as a timing source while using the cache-occupancy attack as the actual side channel.

Adding Noise: Shusterman et al. [18] introduce cache noise using Mastik [98] to generate random cache accesses. Specifically, it repeatedly evict the entire last-level cache, which causes the cache-occupancy attack to achieve less than 1% accuracy when using the classifier. Cook et al. [42] add a Chrome extension that generates thousands of interrupts by scheduling activity bursts and network events. Their spurious interrupt countermeasure reduces the accuracy of the loop-counting attack from 95.7% to 62.0%. Additionally, to defend against cross-app information leaks on iOS, Wang et al. [99] use probabilistic search to approximate an optimal subset of the channels that expose the most information, gradually injecting noise into each channel one by one until the result reaches a pre-defined threshold. After deploying their mitigation, the accuracy of website-fingerprinting decreases from 93.7% to 1.0%. The above three methods are theoretically effective for mitigating co-located website-fingerprinting attacks, though the more effective the mitigation, the greater the performance loss for the browser. We use the cache-occupancy attack as an example. As shown by Shusterman et al. [18], after injecting cache-access noise into the cache-occupancy attack, the accuracy decreases to less than 1%. However, this defense also incurs a non-negligible performance cost: on average, the system experiences a 5% slowdown on the SPEC 2006 benchmark, with the worst case reaching 14% on `bwave`.

VIII. Conclusion

In this work, we introduce a framework to analyze the root causes of three co-located website-fingerprinting attacks. We use qualitative analysis to identify and validate the main leakage sources for each attack. Through quantitative analysis, we determine the leakage source that contributes the most to the success of the attack. Since each leakage source can leak enough information for a successful attack, defending at the system level is infeasible, as it would require major system redesigns. Therefore, we recommend defending against the coarse-grained measurement capabilities of attacks by using methods such as randomized timers or the injection of noise.

Our results further highlight the need for a transparent interface between the microarchitecture of modern processors and the software systems that run on them. In the meantime, we believe that our methodology can help to further the understanding of coarse-grained microarchitectural attacks and help inform the design of future mitigations.

Acknowledgment

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The authors thank the reviewers, as well as Jason Kim and Jalen Chuang from the Hardware Security Lab at the Georgia Institute of Technology, for their assistance with the experiments

References

[1] A. C. Aldaya, C. P. García, L. M. A. Tapia, and B. B. Brumley, "Cache-timing attacks on RSA key generation," in *Proc. IACR Trans. Cryptographic Hardware Embedded Syst.*, 2019, pp. 213–242.
[2] B. B. Brumley and R. M. Hakala, "Cache-timing template attacks," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2009, pp. 667–684.
[3] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX amplifies the power of cache attacks," in *Proc. Int. Conf. Cryptographic Hardware Embedded Syst.*, 2017, pp. 69–90.
[4] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Proc. Cryptographer's Track RSA Conf.*, 2006, pp. 1–20.
[5] E. Ronen, R. Gillham, D. Genkin, A. Shamir, D. Wong, and Y. Yarom, "The 9 lives of Bleichenbacher's CAT: New Cache ATtacks on TLS implementations," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 435–452.
[6] Y. Yarom and K. Falkner, "Flush+Reload: A high resolution, low noise, L3 cache side-channel attack," in *Proc. USENIX Conf. Secur. Symp.*, 2014, pp. 719–732.
[7] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Jump over ASLR: Attacking branch predictors to bypass ASLR," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–13.
[8] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida, "ASLR on the line: Practical cache attacks on the MMU," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, Art. no. 26.
[9] R. Hund, C. Willems, and T. Holz, "Practical timing side channel attacks against Kernel space ASLR," in *Proc. IEEE Symp. Secur. Privacy*, 2013, pp. 191–205.
[10] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *Proc. 11th USENIX Conf. Offensive Technol.*, 2017, Art. no. 11.
[11] J. Van Bulck et al., "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proc. USENIX Conf. Secur. Symp.*, 2018, pp. 991–1008.
[12] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T.-H. Lai, "SgxPectre: Stealing intel secrets from SGX enclaves via speculative execution," in *Proc. IEEE Symp. Secur. Privacy*, 2020, pp. 28–37.

[13] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *Proc. USENIX Conf. Secur. Symp.*, 2015, pp. 897–912.
[14] M. Lipp, D. Gruss, M. Schwarz, D. Bidner, C. Maurice, and S. Mangard, "Practical keystroke timing attacks in sandboxed JavaScript," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2017, pp. 191–209.
[15] D. Genkin, L. Pachmanov, E. Tromer, and Y. Yarom, "Drive-by key-extraction cache attacks from portable code," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2018, pp. 83–102.
[16] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in JavaScript and their implications," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1406–1418.
[17] A. Shusterman, A. Agarwal, S. O'Connell, D. Genkin, Y. Oren, and Y. Yarom, "Prime+Probe 1, JavaScript 0: Overcoming browser-based side-channel defenses," in *Proc. USENIX Conf. Secur. Symp.*, 2021, pp. 2863–2880.
[18] A. Shusterman et al., "Robust website fingerprinting through the cache occupancy channel," in *Proc. USENIX Conf. Secur. Symp.*, 2019, pp. 639–656.
[19] P. Cronin, X. Gao, H. Wang, and C. Cotton, "An exploration of ARM system-level cache and GPU side channels," in *Proc. 37th Annu. Comput. Secur. Appl. Conf.*, 2021, pp. 784–795.
[20] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: GPU side channel attacks are practical," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 2139–2153.
[21] H. Taneja, J. Kim, J. J. Xu, S. van Schaik, D. Genkin, and Y. Yarom, "Hot pixels: Frequency, power, and temperature attacks on GPUs and ARM SoCs," in *Proc. USENIX Conf. Secur. Symp.*, 2023.
[22] P. Kocher et al., "Spectre attacks: Exploiting speculative execution," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 1–19.
[23] A. Agarwal et al., "Spook.js: Attacking Chrome strict site isolation via speculative execution," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 699–715.
[24] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache Telepathy: Leveraging shared resource attacks to learn DNN architectures," in *Proc. USENIX Conf. Secur. Symp.*, 2020, pp. 2003–2020.
[25] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 605–622.
[26] C. Percival, "Cache missing for fun and profit," *BSDCan*, pp. 1–13, 2005. [Online]. Available: https://www.daemonology.net/papers/htt.pdf
[27] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas, "Attack directories, not caches: Side channel attacks in a non-inclusive world," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 888–904.
[28] O. Acıiçmez, S. Gueron, and J.-P. Seifert, "New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures," in *Proc. IMA Int. Conf. Cryptogr. Coding*, 2007, pp. 185–203.
[29] O. Acıiçmez, Ç. K. Koç, and J.-P. Seifert, "Predicting secret keys via branch prediction," in *Proc. Cryptographers' Track RSA Conf.*, 2006, pp. 225–242.
[30] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, and D. Ponomarev, "BranchScope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, vol. 53, pp. 693–707, 2018.
[31] T. Zhang, K. Koltermann, and D. Evtyushkin, "Exploring branch predictors for constructing transient execution Trojans," in *Proc. 35th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 667–682.
[32] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *Proc. USENIX Conf. Secur. Symp.*, 2018, pp. 955–972.
[33] J. Koschel, C. Giuffrida, H. Bos, and K. Razavi, "TagBleed: Breaking KASLR on the isolated Kernel address space using tagged TLBs," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2020, pp. 309–321.
[34] S. Van Schaik, C. Giuffrida, H. Bos, and K. Razavi, "Malicious Management Unit: Why stopping cache attacks in software is harder than you think," in *Proc. USENIX Conf. Secur. Symp.*, 2018, pp. 937–954.
[35] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring(s): Side channel attacks on the CPU on-chip ring interconnect are practical," in *Proc. USENIX Conf. Secur. Symp.*, 2021, pp. 645–662.
[36] J. Wan, Y. Bi, Z. Zhou, and Z. Li, "MeshUp: Stateless cache side-channel attack on CPU mesh," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 1506–1524.
[37] O. Acıiçmez and J.-P. Seifert, "Cheap hardware parallelism implies cheap security," in *Proc. Workshop Fault Diagnosis Tolerance Cryptogr.*, 2007, pp. 80–91.
[38] A. Bhattacharyya et al., "SMoTherSpectre: Exploiting speculative execution through port contention," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 785–800.
[39] A. C. Aldaya, B. B. Brumley, S. ul Hassan, C. P. García, and N. Tuveri, "Port contention for fun and profit," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 870–887.
[40] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. A. Faruque, "Leaky DNN: Stealing deep-learning model secret with GPU context-switching side-channel," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2020, pp. 125–137.
[41] R. Owens and W. Wang, "Non-interactive OS fingerprinting through memory de-duplication technique in virtual machines," in *Proc. IEEE Int. Perform. Comput. Commun. Conf.*, 2011, pp. 1–8.
[42] 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 *Proc. 49th Annu. Int. Symp. Comput. Architecture*, 2022, pp. 204–217.
[43] N. Matyunin et al., "MagneticSpy: Exploiting magnetometer in mobile devices for website and application fingerprinting," in *Proc. 18th ACM Workshop Privacy Electron. Soc.*, 2019, pp. 135–149.
[44] B. Gülmezoglu, "XAI-based microarchitectural side-channel analysis for website fingerprinting attacks and defenses," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 6, pp. 4039–4051, Nov./Dec. 2021.
[45] B. Gülmezoglu, T. Eisenbarth, and B. Sunar, "Cache-based application detection in the cloud using machine learning," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 288–300.
[46] Z. Wang and R. B. Lee, "Covert and side channels due to processor architecture," in *Proc. 22nd Annu. Comput. Secur. Appl. Conf.*, 2006, pp. 473–482.
[47] C. Reis, A. Moshchuk, and N. Oskov, "Site isolation: Process separation for web sites within the browser," in *Proc. USENIX Conf. Secur. Symp.*, 2019, pp. 1661–1678.
[48] M. B. Tracker, "SVG filter timing attack," 2013. [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=711043
[49] Chromium Project, "Timing attack on denormalized floating point arithmetic in SVG filters circumvents same-origin policy," 2016. [Online]. Available: https://bugs.chromium.org/p/chromium/issues/detail?id=615851
[50] M. Patrignani and M. Guarnieri, "Exorcising spectres with secure compilers," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 445–461.
[51] D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, and S. Mangard, "KASLR is dead: Long live KASLR," in *Proc. Int. Symp. Eng. Secure Softw. Syst.*, 2017, pp. 161–176.
[52] Q. Ge, Y. Yarom, T. Chothia, and G. Heiser, "Time protection: The missing OS abstraction," in *Proc. 14th EuroSys Conf.*, 2019, pp. 1:1–1:17.
[53] R. Zhang, T. Kim, D. Weber, and M. Schwarz, "(M)WAIT for it: Bridging the gap between microarchitectural and architectural side channels," in *Proc. USENIX Conf. Secur. Symp.*, 2023, pp. 7267–7284.
[54] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, pp. 613–615, 1973.
[55] G. Irazoqui Apecechea, T. Eisenbarth, and B. Sunar, "S$A: A shared cache attack that works across cores and defies VM sandboxing - and its application to AES," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 591–604.
[56] Y. Wang, R. Paccagnella, E. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86," in *Proc. USENIX Conf. Secur. Symp.*, 2022, pp. 679–697.
[57] Y. Wang et al., "DVFS frequently leaks secrets: Hertzbleed attacks beyond SIKE, cryptography, and CPU-only data," in *Proc. IEEE Symp. Secur. Privacy*, 2023, pp. 2306–2320.
[58] C. Liu, A. Chakraborty, N. Chawla, and N. Roggel, "Frequency throttling side-channel attack," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2022, pp. 1977–1991.
[59] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "DRAMA: Exploiting DRAM addressing for cross-CPU attacks," in *Proc. USENIX Conf. Secur. Symp.*, 2016, pp. 565–581.
[60] V. van der Veen and B. Gras, "DramaQueen: Revisiting side channels in DRAM," in *Proc. Workshop DRAM Secur.*, 2023.
[61] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: A timing attack on OpenSSL constant-time RSA," *J. Cryptographic Eng.*, vol. 7, pp. 99–112, 2017.
[62] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 368–379.

[63] S. Seonghun, D. Debopriya Roy, and G. Berk, "DefWeb: Defending user privacy against cache-based website fingerprinting attacks with intelligent noise injection," in *Proc. 39th Annu. Comput. Secur. Appl. Conf.*, 2023, pp. 379–393.
[64] M. Li, K. Bu, C. Miao, and K. Ren, "TreasureCache: Hiding cache evictions against side-channel attacks," *IEEE Trans. Dependable Secure Comput.*, vol. 21, no. 5, pp. 4574–4588, Sep./Oct. 2024.
[65] Q. Wang, X. Zhang, H. Wang, Y. Gu, and M. Tang, "BackCache: Mitigating contention-based cache timing attacks by hiding cache line evictions," *arXiv:2304.10268*, 2023.
[66] A. Hintz, "Fingerprinting websites using traffic analysis," in *Proc. 2nd Int. Conf. Privacy Enhancing Technol.*, 2002, pp. 171–178.
[67] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-Bayes classifier," in *Proc. 2009 ACM workshop Cloud Comput. Secur.*, 2009, pp. 31–42.
[68] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in Onion Routing based anonymization networks," in *Proc. 10th Annu. ACM Workshop Privacy Electron. Soc.*, 2011, pp. 103–114.
[69] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2012, pp. 605–616.
[70] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, "Website detection using remote traffic analysis," in *Proc. Int. Symp. Privacy Enhancing Technol. Symp.*, 2012, pp. 58–78.
[71] T. Wang and I. Goldberg, "Improved website fingerprinting on ToR," in *Proc. 12th ACM Workshop Workshop Privacy Electron. Soc.*, 2013, pp. 201–212.
[72] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 263–274.
[73] J. Hayes and G. Danezis, "*k*-fingerprinting: A robust scalable website fingerprinting technique," in *Proc. USENIX Conf. Secur. Symp.*, 2016, pp. 1187–1203.
[74] A. Panchenko et al., "Website fingerprinting at Internet scale," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, Art. no. 23477.
[75] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018.
[76] R. Jansen, M. Juarez, R. Galvez, T. Elahi, and C. Diaz, "Inside Job: Applying traffic analysis to measure Tor from within," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018.
[77] S. Li, H. Guo, and N. Hopper, "Measuring information leakage in website fingerprinting attacks and defenses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1977–1992.
[78] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 227–238.
[79] X. Cai, R. Nithyanand, and R. Johnson, "CS-BuFLO: A congestion sensitive website fingerprinting defense," in *Proc. 13th Workshop Privacy Electron. Soc.*, 2014, pp. 121–130.
[80] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *Proc. 13th Workshop Privacy Electron. Soc.*, 2014, pp. 131–134.
[81] T. Wang and I. Goldberg, "Walkie-Talkie: An efficient defense against passive website fingerprinting attacks," in *Proc. USENIX Conf. Secur. Symp.*, 2017, pp. 1375–1390.
[82] G. Cherubin, J. Hayes, and M. Juárez, "Website fingerprinting defenses at the application layer," in *Proc. Privacy Enhancing Technol.*, 2017, pp. 186–203.
[83] F. Rauscher, A. Kogler, J. Juffinger, and D. Gruss, "IdleLeak: Exploiting idle state side effects for information leakage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024.
[84] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Power analysis attacks of modular exponentiation in smartcards," in *Proc. 1st Int. Workshop Cryptographic Hardware Embedded Syst.*, 1999, pp. 144–157.
[85] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Berlin, Germany: Springer, 2008.
[86] Z. Zhang, S. Liang, F. Yao, and X. Gao, "Red Alert for Power Leakage: Exploiting Intel RAPL-induced side channels," in *Proc. ACM Asia Conf. Comput. Commun. Secur*, 2021, pp. 162–175.
[87] M. Lipp et al., "Platypus: Software-based power side-channel attacks on x86," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 355–371.
[88] M. Lipp, D. Gruss, and M. Schwarz, "AMD prefetch attacks through power and time," in *Proc. USENIX Conf. Secur. Symp.*, 2022, pp. 643–660.
[89] AMD Corporation, "Amd CVE-2020-12912," 2020. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2020-12912
[90] Intel Corporation, "Intel CVE-2020-8694," 2020. [Online]. Available: https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00389.html
[91] D. R. Dipta and B. Gulmezoglu, "DF-SCA: Dynamic frequency side channel attacks are practical," in *Proc. 38th Annu. Comput. Secur. Appl. Conf.*, 2022, pp. 841–853.
[92] Intel Corporation, "Intel 64 and IA-32 architectures software developer's manual," 2023. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
[93] X. Zhang et al., "SegScope: Probing fine-grained interrupts via architectural footprints," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architecture*, 2024, pp. 424–438.
[94] M. Dai, R. Paccagnella, M. Gomez-Garcia, J. McCalpin, and M. Yan, "Don't mesh around: Side-channel attacks and mitigations on mesh interconnects," in *Proc. USENIX Secur.*, 2022, pp. 2857–2874.
[95] Y. Wang, A. Ferraiuolo, and G. E. Suh, "Timing channel protection for a shared memory controller," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architecture*, 2014, pp. 225–236.
[96] J. Kim, S. Van Schaik, D. Genkin, and Y. Yarom, "ileakage: Browser-based timerless speculative execution attacks on apple devices," in *Proc. 2023 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 2038–2052.
[97] D. Katzman, W. Kosasih, C. Chuengsatiansup, E. Ronen, and Y. Yarom, "The gates of time: Improving cache attacks with transient execution," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 1955–1972.
[98] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," 2016. [Online]. Available: https://github.com/0xADE1A1DE/Mastik
[99] Z. Wang, J. Guan, X. Wang, W. Wang, L. Xing, and F. Alharbi, "The danger of minimum exposures: Understanding cross-app information leaks on iOS through multi-side-channel learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 281–295.

Yusi Feng received the PhD degree in computer systems organization from the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China, in 2025. She is a postdoctoral researcher with the Southern University of Science and Technology. Her research interests include system security and side-channel attacks.

Sioli O'Connell is currently working toward the PhD degree with the University of Adelaide. His thesis on microarchitectural side-channel attacks in browsers has been accepted and he expects to receive the degree in September 2025. His research interests include side-channel attacks, web browsers, and computer architecture.

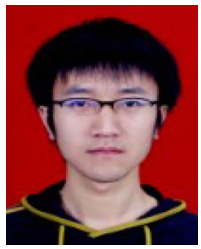
Xin Zhang received the BSc degree in information security from Hunan University, in 2022. He is currently working toward the PhD degree with Peking University, Beijing, China. His research interests include system security, computer architecture and side channel attack.

Chitchanok Chuengsatiansup is a professor of Cybersecurity with the Hasso-Plattner Institute and the University of Potsdam. Her research aims with enhancing security and efficiency of mechanisms to protect sensitive information. She is interested in analyzing and optimizing interrelated factors to achieve high security protection while maintaining high performance.

Daniel Genkin is an Alan and Anne Taetle Early Career associate professor with the School of Cybersecurity and Privacy, Georgia Tech. His research interests are in system security and cryptography. He is interested in both theory and practice with particular interests in side-channel attacks, hardware security, cryptanalysis, secure multiparty computation (MPC), verifiable computation and SNARKS.

Yuval Yarom is a professor of Computer Security with Ruhr University Bochum. His research focuses on the interface between the software and the hardware. In particular, He is interested in the discrepancy between the way that programmers think about software execution and the concrete execution in modern processors.

Yinqian Zhang (Senior Member, IEEE) is a professor with the Department of Computer Science and Engineer ing, Southern University of Science and Technology. His research interest include system security, including side channels, trusted and confidential computing, and cloud security.



Zhi Zhang (Member, IEEE) received the PhD degree from the University of New South Wales. He is a lecturer with 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 and WWW.