AVXProbe: Enhancing Website Fingerprinting with Side-Channel-Assisted Kernel-Level Traces

Suryeon Kim
KAIST
Daejeon, Republic of Korea
c16192@kaist.ac.kr

Seung Ho Na
KAIST
Daejeon, Republic of Korea
harry.na@kaist.ac.kr

Jaehan Kim
KAIST
Daejeon, Republic of Korea
jaehan@kaist.ac.kr

Seungwon Shin
KAIST
Daejeon, Republic of Korea
cluade@kaist.ac.kr

Hyunwoo Choi
Sungshin Women's University
Seoul, Republic of Korea
zemisolsol@sungshin.ac.kr

Abstract

Cache-based website fingerprinting attacks pose substantial privacy and security concerns. These attacks can unveil a user's browsing behavior by monitoring cache activities without depending on network traces. However, existing studies that monitor the entire cache often generate noisy data and require large datasets. This study addresses the limitations of the aforementioned methods by introducing a novel cache-based website fingerprinting technique, called *AVXProbe*, which leverages kernel-level information. In this attack, an attacker collects timing information from specific regions of the Translation Lookaside Buffer (TLB) and cache within the address space of loaded kernel modules by measuring the access time of AVX masked operations. As AVX operations are commonly used on x86, our attack is effective on a broad range of modern Intel and AMD processors. The attacker employs machine learning techniques to identify the website visited by the victim. The evaluation results show that our attack achieves up to 97.7% accuracy on 100 websites. The attack is robust, achieving nearly 90% accuracy with 3.2 seconds of data collection time and 8 training data per website, while state-of-the-art attacks demonstrate about 44-70% accuracy under the same conditions. Additionally, ablation studies reveal that the attack's performance depends on combining multiple kernel module groups rather than a single module. Our attack highlights the potential of kernel-level side-channel information to enhance website fingerprinting attacks, posing new challenges to privacy and security in web browsing.

CCS Concepts

• **Security and privacy → Pseudonymity, anonymity and untraceability**; **Side-channel analysis and countermeasures**.

Keywords

Website fingerprinting, Side channel attack, microarchitecture, Advanced Vector Extensions, AVX, security

ASIA CCS '25, Hanoi, Vietnam

ACM ISBN 979-8-4007-1410-8/25/08
https://doi.org/10.1145/3708821.3710819

ACM Reference Format:
Suryeon Kim, Seung Ho Na, Jaehan Kim, Seungwon Shin, and Hyunwoo Choi. 2025. AVXProbe: Enhancing Website Fingerprinting with Side-Channel-Assisted Kernel-Level Traces. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3708821.3710819

1 Introduction

Website fingerprinting (WF) is a technique that allows an attacker to identify specific websites or web pages accessed by the user. In classical website fingerprinting attack, the attacker relies on observing network traffic patterns between the user's browser and the server, including features such as packet size, direction, timing, and other information [3, 12, 21, 46, 49, 51, 58, 64]. By creating unique fingerprints for each website, this technique can accurately identify the website visited by the user, even when the actual content is encrypted or hidden. Given the significant privacy and security concerns involved, researchers are actively working on strategies to counter network-level website fingerprinting attacks [9, 15, 22, 28, 42, 66].

Additionally, website fingerprinting attacks can exploit microarchitectural side-channel information such as cache states [8, 19, 45, 56, 57], system interrupts [7, 68], power consumption [69], and CPU frequency [11]. These microarchitecture-based attacks can be particularly effective, even in scenarios where network-based fingerprinting is prevented, such as when the browser retrieves website contents from its response cache rather than from the network. The cache occupancy attack [8, 57] collects the cache activities of the entire Last-Level Cache (LLC) and employs machine learning or deep learning techniques to classify the collected timing information. Shusterman et al. [57] showed that cache-based side-channel information reflects both networking and rendering activity. They also demonstrated that the cache occupancy channel can achieve over 92% accuracy on 100 websites. However, despite its high accuracy in website fingerprinting attacks, the existing cache occupancy channel measures contention across the entire LLC, making it susceptible to interference from other programs. Consequently, even though machine learning and deep learning models can tolerate medium to heavy noise, they require a substantial amount of data for a successful attack.

In this study, we introduce a novel cache-based website fingerprinting attack technique, called *AVXProbe*, aimed at achieving a robust WF attack with a smaller dataset compared to previous

cache-based methods. As users navigate websites, their devices concurrently manage various tasks, including networking, rendering, encryption, and power/frequency management. These operations, which rely on hardware resources (e.g., network, power, etc.), are processed on the kernel space. Therefore, analyzing side-channel information from kernel modules involved in browser rendering can introduce new and potent techniques for website fingerprinting attack. For example, if attackers can identify the cache regions corresponding to the kernel modules involved in browser activity, they can minimize cache access when conducting cache-based WF attack. We show that by monitoring kernel activity in specific TLB and cache regions rather than the entire noisy LLC, a more robust cache-based website fingerprinting attack can be performed, achieving higher accuracy with significantly reduced dataset.

To monitor kernel activities involved in browser rendering, we can leverage recently reported side-channel attack techniques [6, 17, 30, 35]. Among these, we utilize the *AVX-TSCHA* attack [30], which exploits unprivileged AVX instructions. It is supported by a wide range of Intel and AMD-based processors, and there are currently no patches available to mitigate it. With the *AVX-TSCHA* attack, we probe the kernel space and collect kernel-level traces that exhibit browser rendering activity. Specifically, we first identify the loaded kernel modules and their addresses. Then, we probe the identified kernel addresses and measure their access times during website visits. A faster access time indicates that the corresponding kernel module is more frequently accessed during website loading. We collect kernel-level traces by repeating kernel probing for a given time. Finally, the collected data is provided to the Support Vector Machine (SVM)-based machine learning classifier for evaluation.

Using the attack methodology outlined above, called *AVXProbe*, we assess the accuracy of the attack targeting 100 websites. The evaluation results show that *AVXProbe* achieves up to 97.7% accuracy with only 20 measurements per website, surpassing the accuracy of other studies that use 100 measurements. In particular, our attack accurately fingerprints websites with nearly 90% accuracy, even with only 3.2 seconds of data collection time per instance and 8 training data per website, while state-of-the-art attacks [7, 11, 68] achieve about 44-70% accuracy under the same conditions. In addition, we conducted ablation studies to determine which kernel modules contribute more significantly to attack performance. The results reveal that website fingerprinting is not solely dependent on a single functional kernel module group, but rather displays strong synergy between specific combinations of multiple groups.
In summary, our contributions are as follows:

- We introduce a novel cache-based website fingerprinting attack, called *AVXProbe*, that leverages kernel module-level side-channel information spanning TLB, cache, and memory.
- We demonstrate the efficacy of kernel-level side channels in website fingerprinting attacks. Our attack shows high accuracy and strong robustness compared to state-of-the-art attacks.
- We perform an in-depth analysis to show the relation between kernel module combinations and attack accuracy. We report kernel module groups that have more impact on performance, suggesting more valuable information for the attack.

2 Background and Related Work

2.1 Microarchitectural Attacks on Kernel Space

TLB-based side-channel attacks [4, 6, 35, 52] can be used to monitor kernel activities. The attacker can infer user actions by observing kernel module events associated with user interactions such as mouse movements, keystrokes, and network connections. Choi et al. [6] demonstrated that attackers can monitor user behaviors by measuring the TLB states of kernel modules, specifically Bluetooth and psmouse. AVX-TSCHA [30] extended this technique to cover more scenarios, including Wi-Fi transmissions, and file downloads through a USB ethernet port. Since access time varies depending on whether the module is in use, attackers can discern user activity related to the monitored kernel module.

2.2 Website Fingerprinting

Network-based attacks. In network-level attacks, an adversary collects traces by monitoring network traffic between the client and the web server. The attacker then analyzes these traces using machine learning techniques [21, 46, 64]. These attacks pose a severe threat to user privacy, as they can bypass traditional security measures such as encryption. Furthermore, these attacks can be leveraged by censorship and surveillance systems, exposing individuals to risks by revealing sensitive information, such as religious beliefs or political views, especially in politically sensitive or repressive environments. Various defense mechanisms have been proposed to address these attacks, introducing obfuscation techniques [28], traffic shaping methods [66], and other countermeasures to disguise the true nature of web traffic and prevent accurate website identification. However, recent research [51] has shown that it is still possible to launch website fingerprinting attacks with high accuracy using deep learning techniques despite the presence of various defense techniques.

Side-channel-based attacks. Website fingerprinting attacks have been the subject of several studies that exploit local side-channel information. Researchers have explored various avenues, including GPU memory dump [41], data usage statistics on Android system [59], storage usage statistics [29], shared event loops [63], hardware performance counters [20], acoustic [13], electromagnetic [38], and CPU frequency [11].

Shusterman et al. [57] investigated a cache-based side-channel information retrieval technique called the "cache-occupancy" attack, where they allocate an LLC-sized buffer and measure access time to the entire buffer. They achieved a 92% success rate in accuracy. They emphasized that considering rendering alongside network in the cache improved their attack accuracy, even when response cache was present. Their follow-up work [56] demonstrated that their new attack technique, called sweep-counting attack, is feasible even entirely from CSS and HTML under highly restricted circumstances where JavaScript execution is completely blocked. However, Cook et al. [7] argued that the root cause of information leakage when monitoring an LLC-sized buffer is not because of memory access but because of system interrupts. They introduced a "loop-counting" attack that achieves high accuracy by repeatedly incrementing a counter in a loop without requiring memory access. Zhang et al. [68] explore a novel method for monitoring fine-grained system

interrupt behavior by abusing segment protection. They adapted the existing framework from the loop-counting attack to demonstrate website fingerprinting, with a high classification accuracy.

Dipta et al. [11] introduce *DF-SCA*, a software-based dynamic frequency side-channel attack. The CPU core frequency values, reflecting system utilization, serve as a reliable side-channel for website fingerprinting, yielding an accuracy rate of 97.6%.

Zhang et al. [67] exploit unprivileged idle-loop optimization instructions (umonitor and umwait) of recent Intel microarchitectures. These instructions provide architectural feedback of a specified memory region that can be utilized as an interrupt proving technique. They demonstrated that interrupt-timing attack can detect which website a user opens with 78% accuracy. However, their method differed from other studies that collected side-channel data via browser, as they relied on network packet characteristics from downloading resources with `curl`, making direct performance comparisons with other works challenging.

Software-based Running Average Power Limit (RAPL) interface can also be used for website fingerprinting [69]. They achieved a high accuracy of 99%. Given that they only targeted 37 websites, it is highly likely that the accuracy would decrease if targeting the 100 websites as in other works. Thus, comparing their study with other works may not be fair. Furthermore, the attack can be easily mitigated by restricting access to it. Recently, most Linux distributions have restricted RAPL access to the "root" user only.

Gulmezoglu et al. [19] identify dominant features impacting accuracy by correlating side-channel information with website request types using browser developer tools. While they focus on explaining characteristics by correlating side-channel information with website resources, our research stands out for solely utilizing collected side-channel information through ablation studies.

Unlike previous literature, our work is distinguished by its usage of the timing information directly from the kernel modules' address space and not the LLC cache. We focus on kernel modules directly associated with browser rendering, making our approach a more fine-tuned analysis. In Section 6.3.3, we demonstrate the potency of our attack on a highly reduced dataset, showing that our results significantly surpass the accuracy of previous state-of-the-art attacks [7, 11, 68]. Our methodology resembles cache template attacks [18, 54], which analyze side-channel information by constructing templates based on the cache access patterns of specific operations. However, our attack differs from general cache template attacks in two key ways: 1) it considers not only the cache but also the TLB, and 2) it applies a template-based approach at the granularity of kernel modules rather than the entire cache structure.

Machine learning. Website fingerprinting attacks are primarily automated through the utilization of advanced Machine Learning (ML) and Deep Learning (DL) algorithms. These algorithms are designed to learn unique patterns associated with each website and to extract distinguishing information from side-channel traces without relying on intuition and expert domain knowledge. Given the distinctive features of each attack primitive, it is crucial to identify appropriate data representation for specific side-channel information. This emphasizes the importance of selecting an adequate ML/DL algorithm in determining attack performance. To date, various algorithms have been employed to facilitate feature engineering, including Random Forest [21], Support Vector Machines (SVM) [46], k-Nearest Neighbors (kNN) [64], Stacked Denoising Autoencoder (SDAE), Convolutional Neural Networks (CNN), and Long Short-Term Memory (LSTM) [7, 51, 56, 57, 69].

2.3 Advanced Vector Extensions

AVX is a Single Instruction Multiple Data (SIMD) instruction set supported by Intel and AMD processors. With AVX, arithmetic and data transfer operations can be processed simultaneously. One of the optimizations in AVX involves masked load/store operations, namely `VMASKMOV` in AVX and `VPMASKMOV` in AVX2. These operations allow for the conditional movement of packed data elements to/from memory, based on mask bits. During the execution of the instructions, faults can occur when accessing an illegal address. However, if the mask bit is set to "*zero*", it does not issue any exceptions [24]. Previous works [6, 30] demonstrated that the *AVX-TSCHA* attack can bypass Kernel Address Space Layout Randomization (KASLR), a technique that randomizes the base address of the kernel image and kernel modules at boot or driver load time. In this paper, we repurpose the *AVX-TSCHA* technique to collect kernel-level traces for website fingerprinting. Unlike prior studies, we gather data not only from a single kernel module but from the entire loaded kernel module. Additionally, we incorporate timing data from TLB, cache, and memory.

3 Research Goal

Previous works have typically analyzed packet characteristics at the network level or utilized information obtained from specific single side channels, as described in Section 2.2. When a browser loads a website, it performs various functions, including networking, file system access, memory allocation, in a complex manner. Collecting side-channel information from each kernel module's address space would enable a more comprehensive analysis. Based on this, we formulate the following first research question:

RQ1) *Can leveraging additional microarchitectural information enhance cache-based WF performance?*

To address the first question, we demonstrate the feasibility of website fingerprinting through a novel attack method, *AVXProbe*, capable of collecting side-channel information at the kernel module granularity in Section 5. We also show high accuracy across various experimental settings in Section 6.2.

Machine learning and deep learning for automatic feature extraction often require extensive labeled datasets. However, reducing training data could offer advantages for attackers. Given that our dataset is derived from the fine-tuned address space of kernel modules rather than the entire noisy cache, we hypothesize our attack is feasible with a significantly smaller dataset than prior works. Therefore, we derive the following second research question:

RQ2) *How potent is the extracted data for WF compared with that of existing attacks?*

To address the second question, we experiment with dataset reduction along two dimensions: 1) side-channel data collection time per single measurement, accounting for website loading time, and 2)

Algorithm 1: Collecting traces from module's address space

```
int Trace[NUM];
mask ← {0, 0, 0, 0, 0, 0, 0, 0}
/* Collect timing information on each module's address
   space */
Procedure ProbeAddr()
    start ← START_ADDR_MODULE;
    end ← END_ADDR_MODULE;
    // collect equal traces for each module
    N ← 500 / MODULES SIZE
    for i = 0; i ≤ N; i++ do
        while start ≤ end do
            Trace[start*i] = AccessTime(start);
            start ← start + 0x1000; // 1 KB boundary
/* Measure access time for the address */
Procedure AccessTime(addr)
    mfence;
    time1 ← rdtscp();
    vmaskmovp (addr), mask, dest;
    time2 ← rdtscp();
    return time2 - time1;
```

the number of measurements taken per website. Throughout the experiment, our primary objective is to determine the extent to which the dataset could be minimized while ensuring the accuracy exceeded the 90% threshold, as we showcase in Section 6.3. We also present the comparison results between our attack and the state-of-the-art attacks in Section 6.3.3.

Finally, we hypothesize that kernel modules directly involved in website loading yield more meaningful information compared to unrelated ones. Thus, we pose the following third research question:

RQ3) *How do different combinations of grouped kernel modules contribute to the attack performance?*

To address the third question, we categorize kernel modules into groups according to their respective functionalities. Subsequently, through ablation studies, we examine which group has a more significant impact on accuracy in Section 6.4.

4 Threat Model

Hardware assumptions. We assume that the processor supports AVX or AVX2 instructions, particularly the masked load/store instructions. Considering that AVX was introduced with the Intel Sandy Bridge (2011) and AMD Bulldozer (2011) processors [14, 27], it is reasonable to expect that most modern mobile, desktop, server and cloud systems come equipped with default support for AVX.

Attacker abilities. We exploit unprivileged access that attackers can execute arbitrary instructions on the local machine. This assumption is commonly observed in side-channel based website fingerprinting attacks [11, 26, 32, 68, 69]. The unprivileged attacker executes malicious code on the victim's device to access local system resources. To effectively monitor access time patterns across various levels of the cache structure while a victim process loads a website, both the attacker and the victim processor are running on the same logical core. We use `taskset` to pin the probing process and the browser process to the same logical core, similar to the approach used in [68]. The attacker can determine the kernel version, the CPU model, and kernel functions' constant offsets. Both the user-level and kernel-level ASLR randomly place the process (sections and libraries) and the kernel (kernel text and modules).

The attacker maintains the browser window size in its default setting to replicate typical victim behavior. While network-level website fingerprinting attacks often disable or clear the browser's response cache between page loads [21, 43, 47, 65], we do not clear the cache before accessing each website, as our attack leverages an ensemble of browser behaviors in kernel space rather than relying solely on network traffic patterns.

Table 1: Top 15 modules actively used during website loading.

Group	Kernel Module Name
Render	`drm`, `drm_kms_helper`, `drm_ttm_helper`, `ttm`
Network	`ip_tables`, `x_tables`, `realtek`, `r8169`
Crypto	`crc32_pclmul`, `cryptd`, `ghash_clmulni_intel`
Power/Temp	`rapl`, `coretemp`
Sound	`soundcore`, `snd_seq_device`

5 Data Collection and Processing

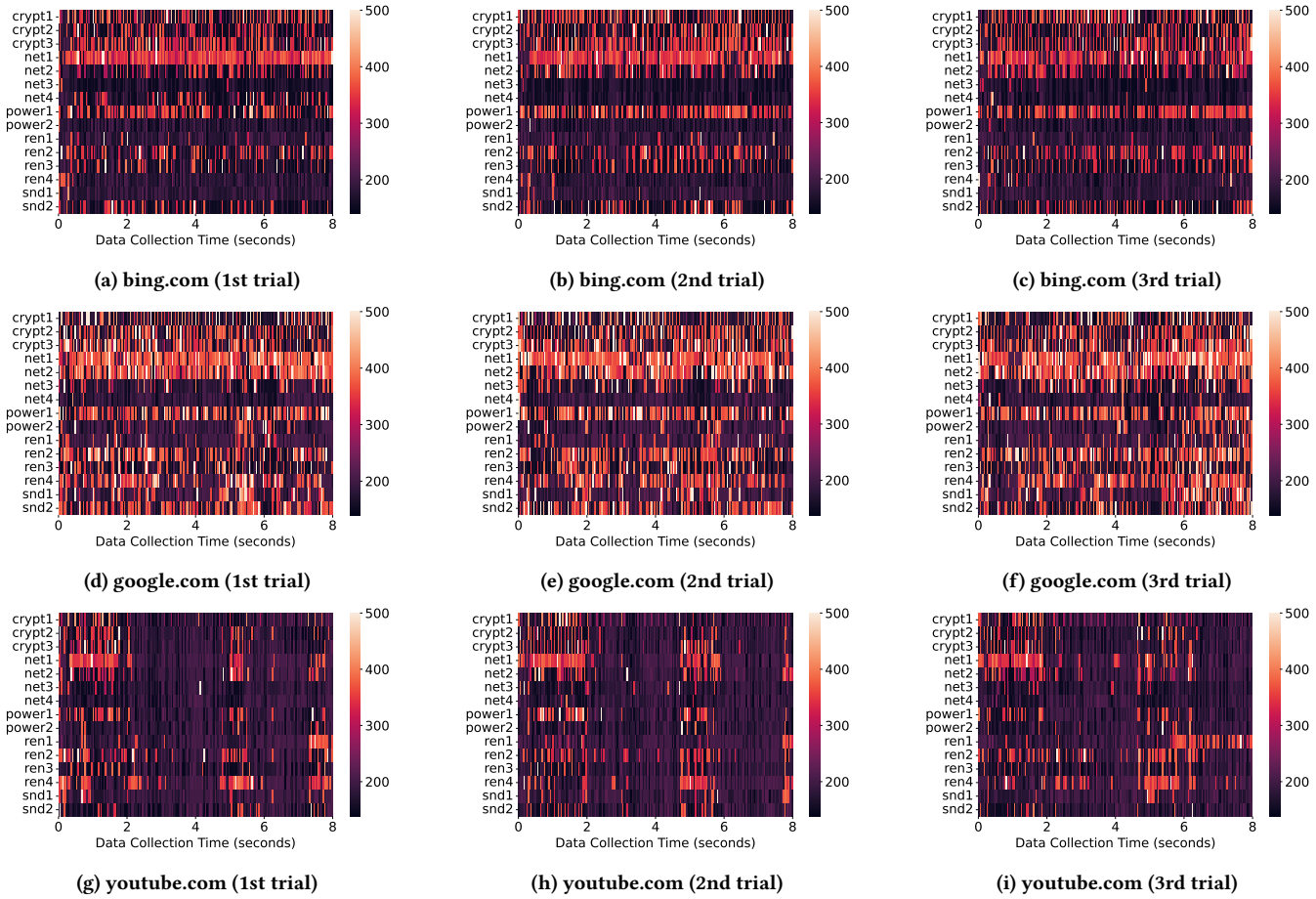
We introduce our methodology for probing the kernel module's address space for website fingerprinting, followed by demonstrating the feasibility of website fingerprinting attacks. We collect kernel-level timing side-channel information using the AVX attack technique [30]. Among the various vulnerable properties of the AVX masked operations, we leverage the following three properties for the website fingerprinting attack:

- ℙ1: The AVX masked operations can suppress exceptions caused by inaccessible memory access.
- ℙ2: The masked operations can distinguish between mapped and unmapped pages.
- ℙ3: The masked operations can identify the TLB state.

Based on ℙ1, we can probe inaccessible kernel space without triggering an exception. Utilizing ℙ2, we can identify the addresses of currently loaded kernel modules. By leveraging ℙ3, we can observe whether more frequent TLB hits occur in a specific kernel module's address space, indicating a higher utilization frequency.

Previous study [4, 6, 30, 35, 52] limited their scope to detecting specific user behaviors by heuristically selecting the kernel module and demonstrating the increase in TLB hits, presuming a direct correlation with user actions, but lacked systematic experimentation. In contrast, we incorporated cache-level data to provide richer features to apply our attack to the website fingerprinting task. Consequently, we collect timing information reflecting TLB, cache, and memory accesses from entire loaded kernel modules, allowing for a more comprehensive analysis compared to previous research.

First, we aim to investigate whether discernible patterns are monitored in each kernel module's address space when accessing different websites. To this end, we collect timing information across the address space of entire loaded kernel modules while visiting



(a) bing.com (1st trial) (b) bing.com (2nd trial) (c) bing.com (3rd trial)

(d) google.com (1st trial) (e) google.com (2nd trial) (f) google.com (3rd trial)

(g) youtube.com (1st trial) (h) youtube.com (2nd trial) (i) youtube.com (3rd trial)

Figure 1: Heatmap of side-channel information collected from Bing, Google, and YouTube (each site visited three times). The x-axis represents traces over 8 seconds, and the y-axis lists the top 15 kernel modules from Table 1, with lengthy names abbreviated as "Group Name + Sequence Number". The color scale (150-500 cycles) indicates access time.

Google and YouTube websites. As a result, we observe that the more frequently used kernel modules vary depending on the visited website. Furthermore, timing information appears in a hierarchical manner across different cache levels, not limited to TLB hits. We also identified distinct distributions divided by three access times (≈170, ≈208, and ≈288 cycles). The fastest threshold value corresponds to TLB hits, the second value to cache hits, and the last value to memory accesses. These distributions showed unique patterns for each website, suggesting that kernel module-level timing information can serve as meaningful features for website fingerprinting.

We further explain how we collect timing side channel information from the memory regions of kernel module. While simulating the victim's website visits, we measure execution times by accessing each kernel module's address space at 1 KiB intervals. Our objective is to detect traces in the TLB and cache left by the victim's website visits, which allows monitoring of access patterns through distinct timing information. Since loading times for Alexa Top 100 websites can take up to approximately 8 seconds [1], we collect side-channel information within this time frame in a single measurement.

Kernel modules vary in size, ranging from the largest module, `Nvidia`, at 52 MiB, to the smallest module, `crypto_simd`, at only 4 KiB. Simply collecting data from the entire address space of each module would introduce significant bias due to the size discrepancy. Therefore, we ensure an equal amount of data collection per module, regardless of size. For instance, when collecting 500 traces for each module, we revisit the addresses of smaller modules until this count is reached (e.g., revisiting the addresses up to 125 times to accumulate 500 traces for a 4 KiB module). For large modules, we select addresses within the first 500 intervals of 1 KiB. The detailed procedure can be found in Algorithm 1.

To counteract the effects of hardware prefetching, which could be triggered by a sequential data probing method, we randomize the access pattern. This approach ensures that timing differences remain observable. However, if the kernel module size is small, even with random accesses, all entries might remain in the TLB, resulting in TLB hit every time. To prevent this, we periodically evicted TLB entries while collecting data [16, 31, 62].

In summary, our dataset collection process consists of three steps:

Table 2: Results of AVXProbe attack for different experimental settings.

Browser	Processor	Time	Traces per module	Num of modules	Traces per instance	Measurements per website	Accuracy
Chrome 112	Intel Alderlake (i5-12400F)	8 sec	500	127	63,500	20	97.7%
Firefox 112	AMD Zen+ (RYZEN5 2600)	8 sec	500	75	37,500	50	96.6%
Tor browser 12	Intel Rocketlake (i7-11700K)	24 sec	1,500	107	160,500	20	74.5%

Step 1: Module detection- Probe the kernel module's address space and derive mapped pages of loaded modules.

Step 2: Data collection- Probe the addresses of each loaded kernel module at 1 KiB intervals (during 8 seconds, we collected 500 traces per module).

Step 3: TLB flushing- Periodically flush the TLB entries while data collection for each module.

We collect side-channel information using the methods described in Steps 1 to 3 while visiting Bing, Google, and YouTube three times each. Our objective is to determine whether there is sufficient similarity within the same site and significant differences between different sites at the module level.

Experimental setup. We utilize an AMD Zen+ processor with 16 GiB of memory, running Ubuntu 20.04.4 LTS with kernel version 5.15.0 and a Chrome 112 browser without any extensions installed. To automate website visits, we rely on the Selenium library [2].

Result. On Zen+ machine, we identified 75 loaded kernel modules and collected 37,500 traces (500 per module in a single measurement). By comparing traces from website loading and idle state, we identified the top 15 modules with increased TLB and cache hits, indicating more frequent access during website loading. This process pinpointed kernel modules related to *Rendering*, *Networking*, *Cryptography*, *Power/Temperature*, and *Sound* functionalities as the most distinctive features, as shown in Table 1. The traces collected from Bing, Google, and YouTube, each loaded three times, are depicted as a heatmap in Figure 1. Darker colors indicate faster access times, indicating that the kernel module's address was accessed during website loading.

When comparing data from Bing, Google, and YouTube, there are noticeable differences between each site. Specifically, we observed differences at the individual module level, even for the same functional group. For example, in the *Network* group modules of Google, the `net4` module (`r8169` in Table 1) was accessed more frequently, while the `net1` module (`ip_tables`) showed less activity. Meanwhile, `net3` module (`realtek`) was accessed more frequently in Bing compared to Google. These results suggest that the side-channel information collected from each kernel module can be a meaningful feature for website fingerprinting.

6 Evaluation

This section demonstrates the efficacy of our novel kernel-level traces for website fingerprinting, evaluated on a highly reduced dataset to showcase robustness. We compare our attack with state-of-the-art side-channel based methods [7, 11, 68] and conduct ablation studies to systematically analyze the impact of individual kernel modules on attack performance.

6.1 Evaluation Setup

We collect data from machines equipped with three processors: Intel Alder Lake (i5-12400F), Intel Rocket Lake (i7-11700K), and AMD Zen+ (RYZEN5 2600). We target browsers Chrome 112, Firefox 112, and Tor browser 12, all without any extensions. All machines run Ubuntu version 20.04 LTS desktop with Linux kernel 5.15.0. For the target websites, we use the 100 most visited websites of Alexa Top list to follow a method similar to previous research [7, 11, 68]. The list of target websites is available in Appendix A Table 7.

Previous side-channel research [7, 11, 56, 57, 69] typically follows similar evaluation settings regarding website loading time (10 seconds to 50 seconds) and the number of measurements (100 samples for each website). The number of traces varies, ranging from 1,000 to 30,000, depending on the side channel's sampling rate. To account for these factors potentially affecting attack accuracy, we design various experimental settings, as shown in Table 2. The settings differ regarding browsers, processors, data collection time (i.e., traces per measurement), and the number of measurements. The number of loaded kernel modules varies across machines due to hardware differences, ranging from 75 to 127 in the three processors we tested. Notably, no artificial loading or unloading of modules occurred. Traces per instance are calculated by multiplying predefined traces by the number of loaded kernel modules.

We perform a website fingerprinting attack as the task of trace classification; given a group of module traces, we determine its website label out of 100 websites. As listed under 'Measurements per Website' of Table 2, the dataset for each environment setting is very limited (less than or equal to 50 per website label). For each dataset, 80% of data per website are randomly sampled for training, with the remaining 20% reserved for testing. Given the restricted nature of our attack setting, the small amount of training data is insufficient for training deep learning models and is likely to cause overfitting with limited generalization [60]. Hence, we opt for SVM for classification due to its effectiveness in handling smaller data and resistance to overfitting [61].

6.2 Evaluation Results

On an Intel Alder Lake processor with Chrome browser, we achieved the highest accuracy of 97.7%. On an AMD Zen+ processor with Firefox browser, accuracy was slightly lower at 96.6%, despite collecting more number of measurements per website. This decrease can be attributed to two factors. First, on the Zen+ processor, we identified only 75 kernel modules, which is approximately 60% lower than the 127 modules identified on the Alder Lake processor. Consequently, the total number of traces per instance was approximately twice as high on the Alder Lake processor. Second, the

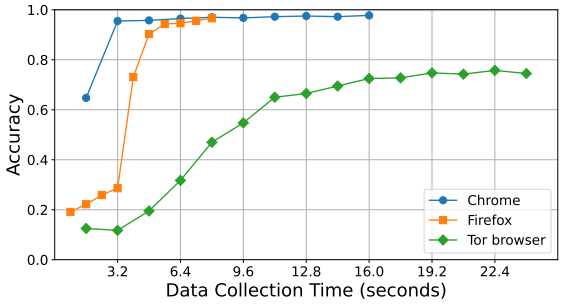


Figure 2: Correlation between accuracy and data collection time per measurement (training data = 16).

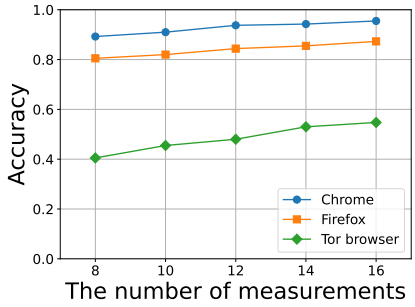


Figure 3: Correlation between accuracy and the number of training data per website.

access time granularity on the Zen+ processor was 34, significantly coarse-grained than the value of 3 on the Alder Lake processor. Since the introduction of Zen architecture, AMD-based CPUs have had a reduced resolution of timestamp counters [37]. This difference in timing information granularity likely resulted in the Zen+ processor providing less informative data, potentially negatively affecting the overall accuracy.

In the Tor browser on an Intel Rocket Lake processor, the success rate reached 74.5%, which was lower than that of other browsers. This discrepancy may be attributed to the routing behavior of the Tor network. To provide a more comprehensive evaluation for the Tor browser setting, we also consider the Top-5 accuracy[1], which is a commonly used metric in side-channel-based WF attacks [7, 11, 56, 57, 68]. By applying the Top-5 metric to the Tor browser environment, we attained a success rate of 95.25%.

Our next evaluation goal is twofold: first, to assess the efficacy of side-channel information by reducing data collection time and the quantity of measurements per website (Section 6.3); and second, to examine the relative impact of side-channel information from different kernel modules on attack performance (Section 6.4). Specifically, our aim is to identify which kernel modules' side-channel information has the most significant effect on attack performance among all kernel modules.

6.3 Reduction of Dataset Size

Aside from measuring website fingerprinting accuracy, we evaluate how valuable the extracted timing information is by seeing how robust the attacks remain when subject to limiting conditions such as data collection time per single measurement, and the number of measurement per website. Furthermore, we compare the robustness of our attack with state-of-the-art attacks and confirm the efficacy of our fine-tuned kernel-level timing information.

6.3.1 Data Collection Time. In previous studies, researchers assume website loading times ranging from 10 to 30 seconds for standard browsers and 30 to 50 seconds for Tor browsers. Based on these assumptions, side-channel information is collected during the specified durations. Initially, we referenced website resource loading times from the Google PageSpeed Insights (PSI) service [1] for the Alexa Top 100 websites to determine proper data collection time. The longest loading time was 7.6 seconds, the shortest time was 0.8 seconds, and the average time was 2.48 seconds. Based on the results, we collect data for 8 seconds in Firefox browsers, and 16 seconds for Chrome to assess accuracy over time further. To account for longer loading times, we set the data collection time to 24 seconds for Tor browser.

We measured the changes in accuracy as we reduced the data collection time from the default setting down to 3.2 seconds, as shown in Figure 2. For Firefox, an accuracy rate of 90.3% was achieved after 4.8 seconds. Chrome exhibited a remarkable accuracy rate of 95.5% within just 3.2 seconds. For the Tor browser, an accuracy rate of 54.75% was achieved after 9.6 seconds, with a substantial increase to 69.5% after 14.4 seconds. In settings where the duration is shorter than 3.2 seconds, the accuracy decreases significantly across all browser configurations. This underscores the necessity of at least 3.2 seconds to effectively collect timing information, as target websites take an average of 2.48 seconds to load resources.

6.3.2 Quantity of Measurements per Website. Previous studies typically collect 100 measurements per website [7, 56, 57, 68]. We hypothesize that leveraging robust kernel-level side-channel information would allow us to achieve sufficient accuracy with fewer measurements than prior work. Therefore, we start with 50 measurements per website for Firefox and 20 measurements per website for Chrome and Tor browsers, which is fewer than the 100 measurements commonly used. Meanwhile, we utilize the optimal data collection time determined in Section 6.3.1. For Firefox, we truncate data at 4.8 seconds, for Chrome at 3.2 seconds, and for Tor at 9.6 seconds. We split the data into an 80/20 ratio for training and testing. For instance, we conduct 20 measurements per website for Chrome and Tor, with 16 for training and 4 for testing. For Firefox, we collected 50 measurements per website, with 40 allocated for training and 10 for testing. The testing data are then fixed, while the number of training data was gradually reduced from 16 to 14, 12, 10, and 8 to evaluate the impact on accuracy, obtaining the results shown in Figure 3. We include only 16 training data for each browser setting to maintain graph scale consistency. On an Intel Alder Lake processor using Chrome, we achieved 95.5% accuracy with 16 training data and maintained 91% accuracy with just 8 training data, demonstrating that high accuracy can be achieved even with less than 10 training data.

[1]Top-k accuracy is a metric used to assess whether the correct class is among the top-k predictions. In a Top-5 scenario, the attacker can guess up to 5 possible websites, and if the correct website is among these guesses, it is considered a success.

Table 3: Classification accuracy obtained with our attack and the baselines while reducing the data collection time, with the number of training data fixed at 16.

Work	16s	12.8s	9.6s	6.4s	3.2s
Loop-Counting [7]	79.55%	78.85%	78.50%	78.15%	74.50%
DF-SCA [11]	N/A	N/A	89.99%	87.99%	81.00%
SegScope [68]	68.30%	67.80%	67.50%	66.75%	66.50%
AVXProbe	97.75%	97.50%	96.75%	96.50%	95.50%

Table 4: Classification accuracy obtained with our attack and the baselines while reducing the number of training data per website, with the data collection time fixed at 3.2 seconds.

Work	16	14	12	10	8
Loop-Counting [7]	74.50%	71.76%	71.73%	65.50%	58.00%
DF-SCA [11]	81.00%	80.66%	77.99%	75.99%	69.99%
SegScope [68]	66.50%	64.33%	60.13%	48.33%	44.80%
AVXProbe	95.50%	94.25%	93.75%	91.00%	89.25%

6.3.3 Comparison to the state-of-the-art Attacks. Our attack achieves over 95% accuracy with just 20 measurements, performing similarly to previous works that required 100 measurements. Therefore, we aim to determine whether other works could still maintain high accuracy with smaller datasets, similar to our research. We compare our attack with the state-of-the-art work in the field of side-channel-based website fingerprinting attacks [7, 11, 68]. We selected these baselines from the most recent studies that have publicly available source code. The Loop-Counting attack [7] uses a malicious JavaScript-based threat model, while the others [11, 68] employ a threat model similar to ours, collecting side-channel information with unprivileged native code execution.

Experimental setup. All the attacks target Alexa Top 100 websites. Each attack employs different side-channel attack primitives and utilizes distinct machine learning classifiers such as LSTM, CNN, k-nearest-neighbors. The attack method and the ML/DL technique in each study are used without modification. For fair comparison, we standardized the default setting of the works to align with our experimental setup, including the number of measurements (e.g., 20 per website) and the data collection time (16 seconds). For classification, we divided the dataset into 10 subsets for cross-validation, with 80% of the data used for training and 20% for testing. We conducted experiments using chrome browser on Intel Alder Lake processor for the Loop-counting and SegScope attack. For the DF-SCA attack, we utilize the publicly available datasets from their original work conducted on Intel Tiger Lake.

Result. We matched the number of measurements to 20 per website, then gradually reduced the data collection time to 3.2 seconds following the method described in Section 6.3.1, while measuring accuracy. We present the results in Table 3[2]. When we reduced the data collection time to 3.2 seconds, *AVXProbe* maintained a high accuracy of 95.5%, while the accuracy of other works declined between 81% to 66%. Using the method of Section 6.3.2, we further reduced the datasets by limiting training data. The results are presented in Table 4. When we used eight training samples per website, *AVXProbe* maintained an accuracy of nearly 90%. In comparison, other works experienced a significant drop in performance, falling below 70%. These results demonstrate the resilience of *AVXProbe* compared to the state-of-the-art attacks under constrained conditions. This can be attributed to the fact that we collected timing information only from each kernel module, which is less noisy than the entire LLC cache.

[2]Since we use a public dataset for the DF-SCA attack, collected only for ten seconds per measurement, we marked accuracy for time exceeding ten seconds as N/A.

Table 5: Ablation study results: the left column shows outcomes after excluding one of ten groups, while the right shows results with only one group retained. Baseline accuracy is 97.7%.

Group	Acc.	Group	Acc.
w/o Sound	93.5%	*w/* Sound	85.7%
w/o Crypto	94.0%	*w/* Bus	85.2%
w/o Render	94.2%	*w/* Misc	75.7%
w/o Misc	94.5%	*w/* Power/Temp	75.5%
w/o HID	94.5%	*w/* Filesystem	75.0%
w/o Data	95.0%	*w/* Render	73.5%
w/o Power/Temp	95.0%	*w/* HID	69.2%
w/o Bus	95.0%	*w/* Crypto	65.5%
w/o Filesystem	95.2%	*w/* Data	57.7%
w/o Network	95.5%	*w/* Network	53.0%

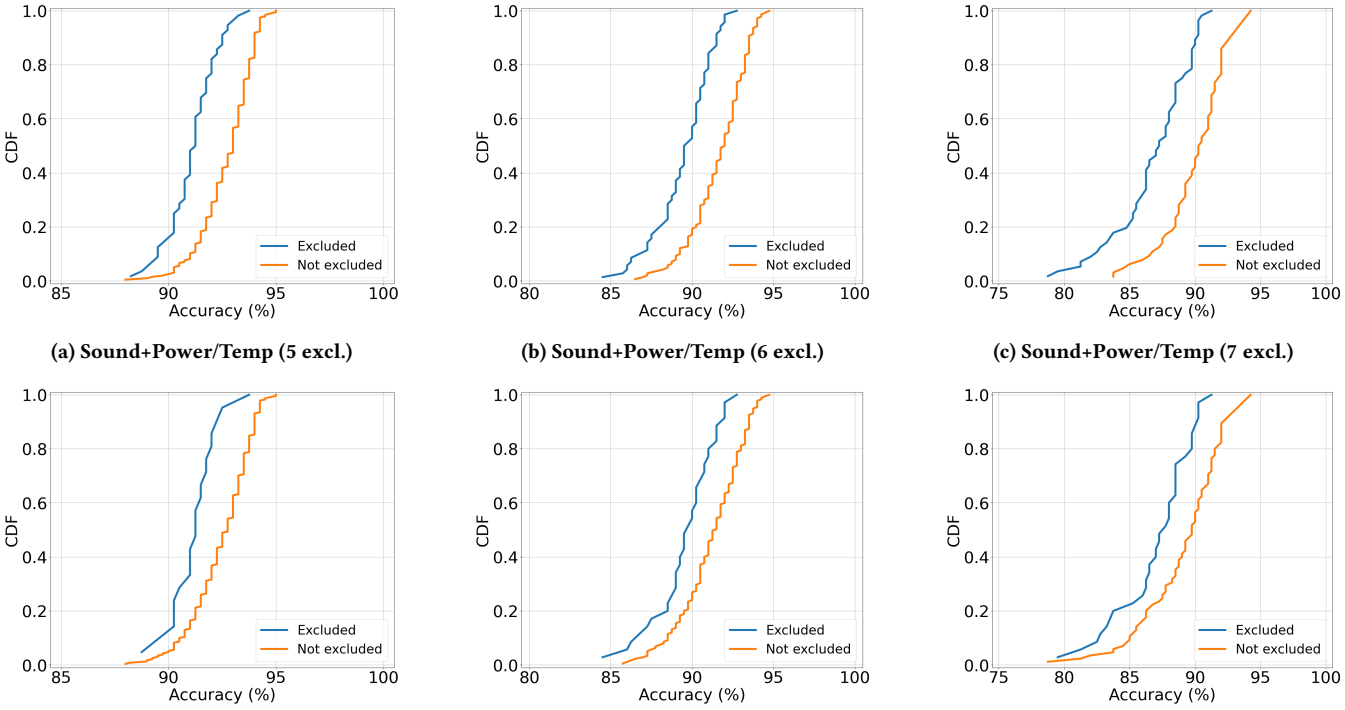
6.4 Impact of Kernel Module Groups

Unlike prior research, which focuses on single side-channel information, we collect data at the granularity of kernel modules. As discussed in Section 5, various kernel modules are accessed during website loading. Modules handling rendering, encryption, and power functions, which are closely related to website loading, are accessed more frequently than others. Intuitively, we predict that the data from these modules are likely to be more informative. To identify which kernel module's trace provides more meaningful information, we categorize 127 loaded kernel modules running on the Intel Alder Lake processor into ten groups based on their functionality, using kernel path information as shown in Appendix B Table 8. Then, we conduct ablation studies to analyze the impact of group of kernel modules on accuracy, starting from a baseline of 97.7% accuracy with all ten groups included. If there is a significant accuracy drop when a specific kernel module group is excluded, this group can be considered an important factor that contains more meaningful information compared to other groups.

We systematically excluded every combination of one to nine groups. The results, obtained by excluding one from ten groups, are presented in the left column of Table 5. In contrast, the results from retaining only one and excluding the remaining nine are shown in the right column. Comparing these two sets of results reveal similar trends but not complete alignment. We speculate that certain activities may involve the interaction of multiple functionalities, and combining data collected from various module groups may

Table 6: Top 5 group combinations (two to four groups) with the highest drop in accuracy. The units for ratio and accuracy are in percentage (%). Ratio signifies the proportion of identified combination groups within the filtered dataset results and Acc. is the respective average accuracy. Delta indicates the decrease in accuracy from the baseline (97.7%).

2 Groups	Ratio	Acc. (Δ)	3 Groups	Ratio	Acc. (Δ)	4 Groups	Ratio	Acc. (Δ)
Sound+Power/Temp	16.0	84.1 (13.6↓)	Sound+Power/Temp+Crypto	8.0	83.9 (13.8↓)	Power/Temp+Crypto+Sound+Misc	6.0	83.4 (14.3↓)
Render+Sound	8.0	90.2 (7.5↓)	Power/Temp+Sound+Misc	6.7	84.1 (13.6↓)	Render+Power/Temp+Sound+Misc	4.0	83.8 (13.9↓)
Crypto+Render	6.7	88.8 (8.9↓)	Render+Crypto+Sound	3.3	90.1 (7.6↓)	Power/Temp+Sound+Misc+Data	2.0	84.3 (13.4↓)
Crypto+Sound	5.3	90.7 (7.0↓)	Sound+HID+Filesystem	3.3	89.9 (7.1↓)	Render+Crypto+Sound+Misc	2.0	84.9 (12.8↓)
Render+HID	4.7	90.0 (7.7↓)	Crypto+Sound+Bus	2.7	90.5 (7.2↓)	Sound+HID+Filesystem+Bus	2.0	90.0 (7.7↓)



(a) Sound+Power/Temp (5 excl.) **(b) Sound+Power/Temp (6 excl.)** **(c) Sound+Power/Temp (7 excl.)**

(d) Sound+Power/Temp+Crypto (5 excl.) **(e) Sound+Power/Temp+Crypto (6 excl.)** **(f) Sound+Power/Temp+Crypto (7 excl.)**

Figure 4: CDF graph showing frequency containing best combination of 2 (Sound+Power/Temp), and 3 (Sound+Power/Temp+Crypto) groups among entire dataset. The best combination of groups prominently occupy the top ranks within each group number of the ablation test results, where 5, 6, and 7 groups have been excluded.

improve accuracy. Thus, our next goal is to identify combinations of groups that generate synergistic effects on accuracy.

For groups containing 2, 3, and 4 members, we aim to discover the combinations with the most significant impact on accuracy. To achieve this, we conduct ablation studies for a total of 912 scenarios, consisting of cases where we excluded 3 to 7 groups from a total of 10 ($\binom{10}{n}$ combinations for exclusion of n groups). For each of the ablation studies where 3, 4, 5, 6, or 7 groups were excluded, we selected the 30 combinations that led to the largest decrease in accuracy. This process resulted in a total of 150 group combinations being selected. Within the selected dataset, we identify the top five combinations, each containing modules of sizes 2, 3, and 4, that appeared most frequently, as shown in Table 6. Unlike in Table 5, the *Power/Temp* group consistently appeared in the best combination in all group sizes. Among the groups listed in Table 1, the *Network* group was the only one not to appear in a top-ranking position. This is likely due to its smaller size, as it only contains five modules. This makes it challenging for the *Network* group to be compared on equal footing with other groups, which have up to 30 modules. As a result, the best combination of two groups was *Sound+Power/Temp*, and for three groups, it was *Sound+Power/Temp+Crypto*.

To evaluate the significance of the best combination groups of size two and three, we analyze their impact on accuracy across the entire dataset. Specifically, we consider all possible combinations of excluding 5, 6, and 7 groups out of 10, resulting in 252, 210, and 120 combinations, respectively. We then compare the accuracy distributions when the best combination groups of size two and three are excluded or not. As shown in Figure 4, we present the Cumulative

Distribution Function (CDF) of accuracy, by highlighting whether these groups are counted among the excluded 5, 6, and 7 groups. Removing the best combination groups results in notable accuracy degradation, reaffirming their crucial role in fingerprinting performance across the entire dataset.

In summary, our analysis revealed that website fingerprinting relies on the combined effects of multiple functional kernel module groups rather than a single kernel module. We leave further experiments for directly discerning the causal impact of each kernel module for future work.

6.5 Characteristics of Side-channel Dataset

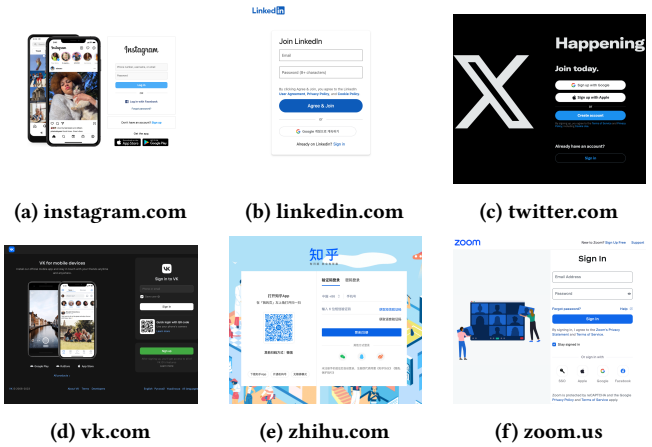
Since adopting of ML/DL techniques has ensured high accuracy in cache-based website fingerprinting attacks lately, there has been comparatively less focus on research utilizing domain knowledge or expert intuition for feature selection. In the domain of application fingerprinting, Li et al. [34] leveraged expert domain knowledge to extract 123 features due to the lack of sufficient training data for deep learning techniques. These features enabled high accuracy even without relying on ML/DL techniques. Similarly, our attack technique achieved high accuracy through automated feature extraction using machine learning. However, we aim to analyze which characteristics of our side-channel dataset served as key features. We explored the characteristics of our timing-based side-channel information in Section 5. We identified that the timing information represents a dataset characterized by hierarchical distinctions based on access time to TLB, cache, and memory. We speculate that despite the conversion of timing information, initially ranging from 150 to 500, into much simpler values based on each hardware access time, sufficient information for classification would still be retained. To demonstrate the feasibility of our assumption, we converted timing information into values of 1, 2, 3, or 4 based on the range in which the timing information fell (~170, 171~207, 208~287, 288~). If the timing information for a specific module is [nvidia, 170], it is transformed into [nvidia, 1]. Similarly, if the timing falls within the range of 171 to 207, it becomes [nvidia, 2], and so forth. Employing the SVM model, we achieved an impressive accuracy of nearly 80% with just a single feature on 100 websites. This result highlights the validity of our intuition regarding the efficacy of utilizing timing information from diverse hierarchical levels.

7 Discussion

7.1 Analysis of Misclassification Factors

To explore avenues for further enhancing the classification success rate, we investigated instances of false positives and negatives, aiming to identify common factors hindering website fingerprinting. This analysis revealed three following cases, some of which could be considered as countermeasures.

Similar login UI. In over half of the cases, occurrences were observed across six websites displaying a login screens: Instagram, LinkedIn, Twitter, VK, Zhihu, and Zoom. These websites immediately redirect users to the login page upon accessing the website. As depicted in Figure 5, these login interfaces share a similar design, featuring common elements such as account information input fields, a login button, a company logo, and social login options. As



(a) instagram.com (b) linkedin.com (c) twitter.com

(d) vk.com (e) zhihu.com (f) zoom.us

Figure 5: Login screens from six websites with similar UIs.

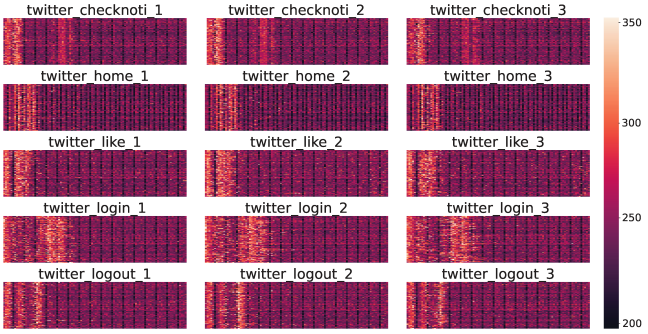


Figure 6: Heatmap of five types of user activities on Twitter.

a result, the resources required for logging in these websites exhibit similarities. Recognizing and classifying these similarities under the same category would likely lead to a better success rate.

JavaScript developed by the same company. A notable misclassification rate was observed on Instagram, with four out of ten samples identified as WhatsApp. This can be attributed to the common JavaScript code that handles the login, authentication, and security functions. Instagram and WhatsApp utilize JavaScript developed by Facebook (Meta), leading to a similar browser rendering. The utilization of third-party JavaScript and libraries, commonly employed for user tracking and advertising, could potentially diminish the accuracy, as observed in the cases of Instagram.

Displaying dynamic content. Websites like Dropbox, iqiyi.com, and qq.com automatically play media upon access. They also frequently change their dynamic content during the data collection period, leading to high rates of false positives and negatives. These characteristics could be leveraged as effective countermeasures against website fingerprinting attacks.

7.2 Fine-grained User Activity Inference

A common limitation of website fingerprinting is that attackers can only identify the main homepage without knowing the internal

pages visited by the user. Recent research [34] demonstrates that finer-grained user actions in Android applications can be identified using app fingerprinting techniques assisted by reverse engineering.

A similar approach could be adopted in the domain of website fingerprinting. For instance, once attackers identifies the website visited by a victim, they could track specific actions on the site, such as logging in or navigating to sub-pages. We conduct a small-scale experiment involving two popular websites: Twitter (now known as X) and Github. We simulate five representative user actions for each website alongside three common actions that apply to all websites: 1) visiting the main homepage, 2) logging in, and 3) logging out. For Twitter, we include the following activities additionally: "check notifications" and "click the like button". As for GitHub, we add the activities of "navigate to a project" and "create a new file". We automate the activities using Selenium.

In the Intel Kaby Lake processor (i7-7700), we collect 500 traces for 90 modules, resulting in 45,000 traces for a single measurement. We collect only 10 measurements for each of the ten activities, the smallest in our experiments. Figure 6 shows the heatmap of five user activities on the Twitter website. The x-axis represents traces over 8 seconds, while the y-axis denotes 90 kernel modules without labels. The height of the y-axis is minimized, making the differences between kernel modules less pronounced and highlighting overall trends. Different burst patterns are noticeable, particularly in the login and logout activities. Unlike visiting the main website, login and logout involve multiple behaviors, such as entering account information, pressing the login/logout button, and being redirected to another page after authentication. Using SVM, we achieved a 100% accuracy in identifying user behaviors, surpassing the 10% accuracy of random guess. A more realistic portrayal of user behavior could be achieved by targeting a wider range of websites and user actions using techniques like site mapping. We leave this exploration for future work.

7.3 Security Implications

Side-channel attacks have been proposed for various use cases such as keystroke monitoring [36, 39], cache attack on OpenSSL AES T-table implementation [25], implementing covert channel [6, 48], breaking KASLR [17], and leaking kernel memory with Spectre gadgets [35, 67]. Recent applications of side-channel attacks have expanded into the realm of model stealing [10, 23, 40, 44, 50, 68]. While they are primarily utilized to extract model architectures, some attacks focus on other elements such as hyperparameters or learned model parameters. The *AVX-TSCHA* attack primitive we utilized has been exploited in previous studies to break KASLR [6, 30]. We repurposed and tailed this technique for launching a website fingerprinting attack, called *AVXProbe*. With fine-grained, information-rich traces at the kernel module level, we built a robust side-channel dataset. We showed that even with reduced data collection time and fewer measurements per website, the accuracy did not significantly decrease compared to the state-of-the-art attacks. *AVXProbe* can be adapted for other side channel attacks, such as keystroke monitoring, building covert channels, and inferring model architecture. Future work will focus on exploring additional applications.

7.4 Attacks on Other Operating Systems

AVXProbe is OS-independent and works on machines equipped with Intel or AMD processors released after 2011. Although we demonstrated our attack on Linux, it is highly likely feasible on other popular operating systems.

Windows. In Windows 10 (ver. 21H1), the kernel and drivers are located between `0xfffff80000000000`-`0xfffff8800000000`, with a 2 Mib boundary if KASLR is enabled. Thus, the base address of the kernel image and kernel modules has 262,144 possible offsets (i.e., 18 bits of entropy). By probing the kernel address space with AVX/AVX2 instruction, we could derandomize the addresses of kernel modules even on Kernel Virtual Address Shadow (KVAS, windows implementation of KPTI)-enabled Windows. Then, it will be straightforward to collect side-channel information for website fingerprinting using the same method we applied on Linux.

macOS. The kernel is mapped between `0xffffff8000000000` and `0xffffff8020000000`, with 2 Mib alignment, resulting in 256 possible offsets. We could identify the kernel's address on macOS version 10.13.1 without Double Map feature (the macOS implementation of KPTI). However, on macOS 10.13.2 with Double Map enabled, even if we locate the Double Map region, we were unable to find the kernel base address since it is implemented separately from the kernel image. Consequently, our attack may face challenges on macOS with the Double Map feature enabled.

7.5 Limitations

Beyond closed-world scenario. The concepts of "closed-world" and "open-world" have been extensively used in previous website fingerprinting researches [7, 55–57, 69]. In the closed-world scenario, the assumption is that the victim visits one of the 100 sensitive websites of interest to the attacker. In the open-world scenario, the attacker additionally collects data for 50 non-sensitive websites. The sensitive sites are labeled into 100 classes, while the non-sensitive websites are labeled as a single label of "non-sensitive", resulting in a total of 101 classes. However, this concept is impractical, as covering the myriad of websites a victim may visit is challenging.

In another scenario explored in [5], termed "monitored vs. unmonitored", the attacker seeks to determine whether test data are part of the monitored set. They utilized three classifiers: synthetic traffic, synthetic traffic with updates (hybrid), and genuine traffic. Despite targeting just five websites, the synthetic-trained model exhibited an accuracy of only 0.03%, highlighting the challenges of website fingerprinting attacks in real-world scenarios. Note that this study focused on Tor network traffic, not side-channel-based data. However, addressing these challenges from a side-channel perspective would be worthwhile in future research.

Automated website testing detection. During data collection, we encountered several websites that employed various methods to detect and block automated access. These methods included CAPTCHA challenges and dynamic URL generation during login, where users are prompted to enter passwords on a newly generated URL. Overcoming these defenses requires tailored manual efforts for each specific method. Thus, we exclude these websites from the current scope of our analysis.

Native code. Due to the absence of support for AVX masked operations in JavaScript and WebAssembly (WASM), our attack cannot be executed within browsers. However, WASM provides partial support for AVX extensions, and ongoing efforts to adapt AVX instructions may enable future attack possibilities. Additionally, if an alternative side-channel vulnerability within JavaScript or WASM capable of monitoring kernel activity is discovered, conducting attacks on browsers could become feasible.

8 Countermeasures

This section presents countermeasures against the *AVXProbe* attack. We consider hardware-level mitigation by disabling the AVX extension [30] and software-level measures such as adding noise to mask timing differences in browser rendering. We encourage stakeholders to consider and implement applicable methods.

8.1 Hardware

Remove/restrict access to AVX instruction. There is currently no known patch available to mitigate our attack. One of the most straightforward countermeasures is to disable or restrict vulnerable features effectively making them privileged methods inaccessible to attackers. Given that *AVXProbe* utilizes the vulnerable features of the AVX/AVX2 extensions, we can consider replacing the masked load/store instructions with no operations (NOPs) when the mask bits are all set to zero. We assessed the prevalence of masked load/store instructions to evaluate the impact of this mitigation. On Ubuntu 20.04 LTS with the default installation, we found only 6 out of 4,104 executables containing masked load/store instructions. Therefore, we concluded that limiting or replacing masked operations does not significantly affect the system.

Modifying CPU architecture. The vulnerable feature of the AVX instructions that we exploit lies in cache translations in TLBs when attempting an unprivileged AVX memory load on inaccessible kernel addresses, allowing us to infer the TLB state using timing information. If unprivileged instructions are prevented from changing the TLB state when accessing kernel addresses, this vulnerability would be mitigated. We discovered that the latest AMD Zen2/Zen3 processors are not vulnerable to our attack because they do not cache inaccessible addresses in the TLB. This suggests that other processors could benefit from a similar strategy. However, applying microarchitectural level patches to already deployed systems is challenging, as evidenced by previous cases like the MDS attack [53], where successful patching took over 18 months from disclosure. Moreover, patches are often circumvented by other variants.
Even if AVX masked operations are disabled or patched, attackers may still utilize other techniques, such as prefetching [35], to monitor kernel address space with user privilege. We leave the detailed performance evaluation for future work.

8.2 Software

Adversarial patch. To defend against ML/DL based website fingerprinting attacks, adversarial patch techniques [33, 55] have been proposed. These systems modify user network traffic by injecting dummy packets into random locations, perturbing the patterns of real-time network traffic. Cook et al. [7] introduced a countermeasure by adding spurious interrupts using a Chrome extension. They implemented noise by incrementing counter values and sending network pings to a bunch of websites at random intervals. We suggest enhancing the effectiveness of the countermeasure by increasing the noise related to the functions outlined in Table 1. However, we did not consider rendering functions due to security policies that prohibit modifying rendering or adding overlays to websites not owned by the attacker. Instead, we suggest using the existing implementation as a basic template [7] and adding more noise features, such as encrypting a random string and playing audio at random intervals in the background. This method will effectively obfuscate the pattern of browser rendering in various aspects. From the perspective of web server administrators, adding dynamic content that changes in a short time can be an effective defense strategy.

KPTI separation. Our attack remains effective even on KPTI-enabled Linux systems. Despite KPTI's protection, we can still locate minimal kernel areas, such as the trampoline code, and calculate the kernel base address using OS-specific offsets. However, when these regions are separated from the rest of the kernel image, as seen with macOS's Double Map feature, identifying the kernel base address becomes impossible. To mitigate our attack on Linux and Windows, a similar approach of segregating KPTI regions, as implemented in macOS, could be considered. Additionally, Function Granular KASLR (FGKASLR) could enhance security by rearranging individual kernel functions, making it difficult for an attacker to pinpoint specific functions even if the kernel's base address is exposed. Furthermore, adopting a more robust fine-grained KASLR, such as address space re-randomization, would provide additional protection against our attack.

9 Conclusion

We introduce a novel attack method, called *AVXProbe*, for analyzing side-channel information in the kernel space for website fingerprinting. We collect timing information from loaded kernel modules instead of the noisy entire LLC cache. We demonstrate that our approach can achieve high accuracy in various experimental settings, highlighting the efficacy of our method. We also show the potency of our attack compared to state-of-the-art attacks on highly reduced datasets. Finally, ablation studies on grouped kernel modules revealed synergistic effects among certain combinations. Our study could motivate further research for finding potential novel side-channels considering kernel module functions related to website loading. As the first kernel-level side-channel attack for website fingerprinting, we highlight the potency of our attack and suggest disabling vulnerable features to mitigate our attack.

Acknowledgments

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. RS-2022-II220740, the Development of Darkweb Hidden Service Identification and Real IP Trace Technology, 70%; No. 2021-0-00118, Development of decentralized consensus composition technology for large-scale nodes, 30%).

References

[1] 2023. PageSpeed Insights. https://pagespeed.web.dev/.
[2] 2023. The Selenium Project. https://www.selenium.dev/.

[3] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2019. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proceedings on Privacy Enhancing Technologies* 4 (2019), 292–310.

[4] Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, et al. 2019. Fallout: Leaking Data on Meltdown-resistant CPUs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 769–784.

[5] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. 2022. Online website fingerprinting: Evaluating website fingerprinting attacks on Tor in the real world. In *31st USENIX Security Symposium (USENIX Security 22)*. 753–770.

[6] Hyunwoo Choi, Suryeon Kim, and Seungwon Shin. 2023. AVX Timing Side-Channel Attacks against Address Space Layout Randomization. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[7] Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. 2022. There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 204–217.

[8] Patrick Cronin, Xing Gao, Haining Wang, and Chase Cotton. 2021. An Exploration of ARM System-Level Cache and GPU Side Channels. In *Annual Computer Security Applications Conference*. 784–795.

[9] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. 2020. TrafficSliver: Fighting Website Fingerprinting Attacks with Traffic Splitting. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1971–1985.

[10] Shuwen Deng, Bowen Huang, and Jakub Szefer. 2022. Leaky frontends: Security vulnerabilities in processor frontends. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 53–66.

[11] Debopriya Roy Dipta and Berk Gulmezoglu. 2022. DF-SCA: Dynamic Frequency Side Channel Attacks are Practical. In *Proceedings of the 38th Annual Computer Security Applications Conference*. 841–853.

[12] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *2012 IEEE symposium on security and privacy*. IEEE, 332–346.

[13] Daniel Genkin, Mihir Pattani, Roei Schuster, and Eran Tromer. 2019. Synesthesia: Detecting screen content via remote acoustic side channels. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 853–869.

[14] Pawel Gepner, Victor Gamayunov, and David L Fraser. 2011. Early performance evaluation of AVX for HPC. *Procedia Computer Science* 4 (2011), 452–460.

[15] Jiajun Gong and Tao Wang. 2020. Zero-delay Lightweight Defenses against Website Fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*. 717–734.

[16] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2018. Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 955–972.

[17] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. 2016. Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 368–379.

[18] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. 2015. Cache template attacks: Automating attacks on inclusive {Last-Level} caches. In *24th USENIX Security Symposium (USENIX Security 15)*. 897–912.

[19] Berk Gulmezoglu. 2021. XAI-based microarchitectural side-channel analysis for website fingerprinting attacks and defenses. *IEEE transactions on dependable and secure computing* 19, 6 (2021), 4039–4051.

[20] Berk Gulmezoglu, Andreas Zankl, Thomas Eisenbarth, and Berk Sunar. 2017. PerfWeb: How to violate web privacy with hardware performance events. In *Computer Security–ESORICS 2017, Proceedings, Part II 22*. Springer, 80–97.

[21] Jamie Hayes, George Danezis, et al. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *USENIX security symposium*. 1187–1203.

[22] James K Holland and Nicholas Hopper. 2022. RegulaTor: A Straightforward Website Fingerprinting Defense. *Proc. Priv. Enhancing Technol.* 2022, 2 (2022), 344–362.

[23] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. 2020. Deepsniffer: A dnn model extraction framework based on learning architectural hints. , 385–399 pages.

[24] Intel. 2021. Intel® 64 and IA-32 Architectures Optimization Reference Manual. *Intel Corporation* (2021).

[25] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. 2014. Wait a minute! A fast, Cross-VM attack on AES. In *Research in Attacks, Intrusions and Defenses: 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings 17*. Springer, 299–319.

[26] Suman Jana and Vitaly Shmatikov. 2012. Memento: Learning secrets from process footprints. In *2012 IEEE Symposium on Security and Privacy*. IEEE, 143–157.

[27] Hwancheol Jeong, Sunghoon Kim, Weonjong Lee, and Seok-Ho Myung. 2012. Performance of SSE and AVX instruction sets. *arXiv preprint arXiv:1211.0820* (2012).

[28] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an efficient website fingerprinting defense. In *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I 21*. Springer, 27–46.

[29] Hyungsub Kim, Sangho Lee, and Jong Kim. 2016. Inferring browser activity and status through remote monitoring of storage usage. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 410–421.

[30] Suryeon Kim, Seungwon Shin, and Hyunwoo Choi. 2023. AVX-TSCHA: Leaking Information Through AVX Extensions in Commercial Processors. *Computers & Security* (2023), 103437.

[31] Jakob Koschel, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. TagBleed: Breaking KASLR on the Isolated Kernel Address Space using Tagged TLBs. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 309–321.

[32] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. 2014. Stealing webpages rendered on your browser by exploiting GPU vulnerabilities. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 19–33.

[33] Ding Li, Yuefei Zhu, Minghao Chen, and Jue Wang. 2022. Minipatch: Undermining DNN-Based Website Fingerprinting With Adversarial Patches. *IEEE Transactions on Information Forensics and Security* 17 (2022), 2437–2451.

[34] Jianfeng Li, Hao Zhou, Shuohan Wu, Xiapu Luo, Ting Wang, Xian Zhan, and Xiaobo Ma. 2022. {FOAP}:{Fine-Grained}{Open-World} Android App Fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*. 1579–1596.

[35] Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. {AMD} prefetch attacks through power and time. In *31st USENIX Security Symposium (USENIX Security 22)*. 643–660.

[36] Moritz Lipp, Daniel Gruss, Michael Schwarz, David Bidner, Clémentine Maurice, and Stefan Mangard. 2017. Practical keystroke timing attacks in sandboxed javascript. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*. Springer, 191–209.

[37] Moritz Lipp, Vedad Hadžić, Michael Schwarz, Arthur Perais, Clémentine Maurice, and Daniel Gruss. 2020. Take a way: Exploring the security implications of AMD's cache way predictors. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. 813–825.

[38] Nikolay Matyunin, Yujue Wang, Tolga Arul, Kristian Kullmann, Jakub Szefer, and Stefan Katzenbeisser. 2019. Magneticspy: Exploiting magnetometer in mobile devices for website and application fingerprinting. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*. 135–149.

[39] John V Monaco. 2018. Sok: Keylogging side channels. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 211–228.

[40] Seung Ho Na, Hyeong Gwon Hong, Junmo Kim, and Seungwon Shin. 2022. Closing the loophole: rethinking reconstruction attacks in federated learning from a privacy standpoint. In *Proceedings of the 38th Annual Computer Security Applications Conference*. 332–345.

[41] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. 2018. Rendered insecure: Gpu side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*. 2139–2153.

[42] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2021. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*. 2705–2722.

[43] Rishab Nithyanand, Xiang Cai, and Rob Johnson. 2014. Glove: A bespoke website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. 131–134.

[44] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. 2023. I know what you trained last summer: A survey on stealing machine learning models and defences. *Comput. Surveys* 55, 14s (2023), 1–41.

[45] Yossef Oren, Vasileios P Kemerlis, Simha Sethumadhavan, and Angelos D Keromytis. 2015. The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1406–1418.

[46] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale. In *NDSS*.

[47] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. 103–114.

[48] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. 2022. Dvfsspy: Using dynamic voltage and frequency scaling as a covert channel for multiple procedures. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 654–659.

[49] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The Utility of Packet Timing in Website Fingerprinting Attacks. *Proc. Priv. Enhancing Technol.* 2020, 3 (2020), 5–24.

[50] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. 2022. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE symposium on security and privacy (SP)*. IEEE, 1157–1174.

[51] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2017. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376* (2017).

[52] Michael Schwarz, Claudio Canella, Lukas Giner, and Daniel Gruss. 2019. Store-to-Leak Forwarding: Leaking Data on Meltdown-resistant CPUs (Updated and Extended Version). *arXiv preprint arXiv:1905.05725* (2019).

[53] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *CCS*.

[54] Martin Schwarzl, Erik Kraft, and Daniel Gruss. 2023. Layered Binary Templating. In *International Conference on Applied Cryptography and Network Security*. Springer, 33–58.

[55] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. 2021. Patch-based defenses against web fingerprinting attacks. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*. 97–109.

[56] Anatoly Shusterman, Ayush Agarwal, Sioli O'Connell, Daniel Genkin, Yossi Oren, and Yuval Yarom. 2021. {Prime+ Probe} 1,{JavaScript} 0: Overcoming Browser-based {Side-Channel} Defenses. In *30th USENIX Security Symposium (USENIX Security 21)*. 2863–2880.

[57] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. 2019. Robust Website Fingerprinting Through the Cache Occupancy Channel. In *28th USENIX Security Symposium (USENIX Security 19)*.

[58] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1928–1943.

[59] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. 2016. Exploiting data-usage statistics for website fingerprinting attacks on Android. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. 49–60.

[60] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[61] Ingo Steinwart and Andreas Christmann. 2008. *Support vector machines*. Springer Science & Business Media.

[62] Andrei Tatar, Daniël Trujillo, Cristiano Giuffrida, and Herbert Bos. 2022. {TLB; DR}: Enhancing {TLB-based} Attacks with {TLB} Desynchronized Reverse Engineering. In *31st USENIX Security Symposium (USENIX Security 22)*. 989–1007.

[63] Pepe Vila and Boris Köpf. 2017. Loophole: Timing Attacks on Shared Event Loops in Chrome.. In *USENIX Security Symposium*. 849–864.

[64] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 143–157.

[65] Tao Wang and Ian Goldberg. 2013. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. 201–212.

[66] Tao Wang and Ian Goldberg. 2017. {Walkie-Talkie}: An efficient defense against passive website fingerprinting attacks. In *26th USENIX Security Symposium (USENIX Security 17)*. 1375–1390.

[67] Ruiyi Zhang, Taehyun Kim, Daniel Weber, and Michael Schwarz. 2023. ({M) WAIT} for It: Bridging the Gap between Microarchitectural and Architectural Side Channels. In *32nd USENIX Security Symposium (USENIX Security 23)*. 7267–7284.

[68] Xin Zhang, Zhi Zhang, Qingni Shen, Wenhao Wang, Yansong Gao, Zhuoxi Yang, and Jiliang Zhang. 2024. SegScope: Probing fine-grained interrupts via architectural footprints. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 424–438.

[69] Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xing Gao. 2021. Red alert for power leakage: Exploiting intel rapl-induced side channels. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 162–175.

A Targeted Website List

We collected side-channel information from the Alexa Top 100 websites following the methodology outlined in Section 6. Table 7 lists the target websites.

B Grouped Kernel Module List

As shown in Table 8, we categorize 127 loaded kernel modules on the Intel Alder Lake processor into ten functional groups by referring to the path information of kernel modules. Our classification resulted in different numbers of kernel modules in each group. The "Bus" group has the highest count with 23 kernel modules, whereas the "Network" group has the fewest with only 5 modules.

If all kernel module groups consisted of the same number of kernel modules, it would have allowed for a more equitable analysis of each function's impact on accuracy. However, adjusting the number of modules in other groups to match the group with the smallest number of modules carries the risk of introducing unwanted bias. Therefore, we opted to keep the modules unchanged.

115.com	3dmgame.com	adobe.com
alibaba.com	apple.com	archive.org
avito.ru	aws.amazon.com	baidu.com
bbc.com	bing.com	bilibili.com
booking.com	canva.com	chase.com
cnn.com	csdn.net	deepl.com
digikala.com	discord.com	douban.com
doubleclick.net	douyu.com	dropbox.com
duckduckgo.com	ebay.com	espn.com
etsy.com	fandom.com	fc2.com
fiverr.com	force.com	freepik.com
github.com	godaddy.com	google.com
grammarly.com	huya.com	ilovepdf.com
imdb.com	imgur.com	instagram.com
instructure.com	intuit.com	iqiyi.com
jd.com	linktr.ee	linkedin.com
live.com	mail.ru	mega.io
medium.com	microsoft.com	msn.com
naver.com	netflix.com	nicovideo.jp
notion.so	nytimes.com	office.com
okta.com	onlyfans.com	openai.com
pinterest.com	qq.com	quora.com
rakuten.co.jp	researchgate.net	savefrom.net
shopify.com	sogou.com	sohu.com
spotify.com	stackoverflow.com	taboola.com
taobao.com	telegram.org	tiktok.com
tmall.com	tradingview.com	trello.com
twitch.tv	tumblr.com	twitter.com
udemy.com	upwork.com	vimeo.com
vk.com	w3schools.com	weather.com
weibo.com	wetransfer.com	whatsapp.com
wikipedia.org	wordpress.com	yahoo.com
youdao.com	youtube.com	zhihu.com
zoom.us		

Table 7: Websites list.

Group (Num)	Kernel Module Name
Bus (23)	soundwire_bus, i2c_algo_bit, parport_pc, parport, i2c_i801, i2c_smbus, xhci_pci_renesas, ppdev, intel_lpss_pci, ahci, xhci_pci, libahci
Crypto (8)	xor, blake2b_generic, crct10dif_pclmul, ghash_clmulni_intel, aesni_intel, crypto_simd, cryptd, crc32_pclmul
Filesystem (13)	xfs, ufs, qnx4, ntfs, msdos, minix, jfs, hfsplus, hfs, btrfs, nls_iso8859_1, binfmt_misc, autofs4
Render (13)	amdgpu, gpu_sched, drm_ttm_helper, ttm, drm_kms_helper, cec, rc_core, fb_sys_fops, syscopyarea, sysfillrect, sysimgblt, drm, video
Hid (9)	ledtrig_audio, input_leds, joydev, intel_hid, mac_hid, sparse_keymap, hid_generic, usbhid, hid
Misc (16)	zstd_compress, cpuid, kvm_intel, kvm, gigabyte_wmi, wmi_bmof, mei_me, pmt_telemetry, mei, pmt_class, msr, lp, efi_pstore, intel_lpss, wmi
Network (5)	sch_fq_codel, ip_tables, x_tables, r8169, realtek
Power/temp (13)	snd_soc_acpi_intel_match, snd_soc_acpi, intel_rapl_msr, intel_rapl_common, intel_tcc_cooling, x86_pkg_temp_thermal, intel_powerclamp, coretemp, snd_intel_sdw_acpi, rapl, intel_cstate, acpi_tad, acpi_pad
Sound (22)	snd_sof_intel_hda_common, snd_sof_intel_hda, snd_sof_xtensa_dsp, snd_sof, snd_soc_hdac_hda, snd_hda_ext_core, snd_soc_core, snd_hda_codec_realtek, snd_compress, snd_pcm_dmaengine, snd_intel_dspcfg, snd_hda_core, snd_hwdep, snd_pcm, snd_seq_midi, snd_seq_midi_event, snd_rawmidi, snd_seq, snd_seq_device, snd_timer, snd, soundcore
Data (5)	raid6_pq, libcrc32c, iommu_v2, ee1004, idma64

Table 8: Group information of Kernel Modules.