

## MIT Open Access Articles

*There's Always a Bigger Fish: A Clarifying Analysis of a Machine-Learning-Assisted Side-Channel Attack*

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

**Citation:** Cook, Jack, Drean, Jules, Behrens, Jonathan and Yan, Mengjia. 2022. "There's Always a Bigger Fish: A Clarifying Analysis of a Machine-Learning-Assisted Side-Channel Attack."

**As Published:** <https://doi.org/10.1145/3470496.3527416>

**Publisher:** ACM|The 49th Annual International Symposium on Computer Architecture

**Persistent URL:** <https://hdl.handle.net/1721.1/146463>

**Version:** Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

**Terms of use:** Creative Commons Attribution 4.0 International license



# There's Always a Bigger Fish: A Clarifying Analysis of a Machine-Learning-Assisted Side-Channel Attack

Jack Cook  
MIT CSAIL  
Cambridge, MA, USA  
cookj@mit.edu

Jonathan Behrens  
MIT CSAIL  
Cambridge, MA, USA  
behrensj@mit.edu

Jules Drean  
MIT CSAIL  
Cambridge, MA, USA  
drear@mit.edu

Mengjia Yan  
MIT CSAIL  
Cambridge, MA, USA  
mengjiay@mit.edu

## ABSTRACT

Machine learning has made it possible to mount powerful attacks through side channels that have traditionally been seen as challenging to exploit. However, due to the black-box nature of machine learning models, these attacks are often difficult to interpret correctly. Models that detect correlations cannot be used to prove causality or understand an attack's various sources of information leakage.

In this paper, we show that a state-of-the-art website-fingerprinting attack powered by machine learning was only partially analyzed. In this attack, an attacker collects cache-sweeping traces, which measure the frequency at which the entire last-level cache can be accessed over time, while a victim loads a website. A neural network is then trained on these traces to predict websites accessed by the victim. The attack's usage of the cache led to a consensus that the attack exploited a cache-based side channel. However, we provide additional analysis contradicting this assumption and clarifying the mechanisms behind this powerful attack.

We first replicate the website-fingerprinting attack without making any cache accesses, demonstrating that memory accesses are not crucial to the attack's success and may even inhibit its performance. We then search for the primary source of information leakage in our new attack by analyzing the effects of various isolation mechanisms and by instrumenting the Linux kernel. We ultimately find that this attack's success can be attributed primarily to system interrupts. Finally, we use this analysis to craft highly practical and effective defense mechanisms against our attack.

## CCS CONCEPTS

• **Security and privacy** → **Side-channel analysis and counter-measures**; • **Computing methodologies** → *Supervised learning by classification*.



This work is licensed under a Creative Commons Attribution International 4.0 License.

ISCA '22, June 18–22, 2022, New York, NY, USA  
© 2022 Copyright held by the owner/author(s).  
ACM ISBN 978-1-4503-8610-4/22/06.  
<https://doi.org/10.1145/3470496.3527416>

## KEYWORDS

side channels, website fingerprinting, microarchitecture, deep learning, security

### ACM Reference Format:

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 *The 49th Annual International Symposium on Computer Architecture (ISCA '22)*, June 18–22, 2022, New York, NY, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3470496.3527416>

## 1 INTRODUCTION

Application-level security in modern systems relies heavily on underlying system software to enforce isolation between different security domains. However, achieving this isolation is challenging due to the existence of *covert channels* and *side channels*. In a covert- or side-channel attack, a transmitter program leaks information about a secret to a receiver program by leveraging contention over a shared resource. In this paper, we focus on studying side channels where the transmitter program is the victim and the receiver program is the adversary. Most side channels exploit system-level resource contention such as memory allocation [31] and file system utilization [67], or hardware resource contention such as caches [22, 46, 79, 80], branch predictors [16, 17], DRAM [58, 78], and pipeline ports [3]. Side channels are widely effective and can be used to leak cryptographic keys [55, 61, 69], website content [6, 64], and user activity [43, 54].

*Problems With Machine-Learning-Assisted Side-Channel Attacks.* Recently published side-channel attacks have made extensive use of machine learning techniques, which simplify the development of these attacks and improve their robustness [4, 7, 12, 29, 35, 40, 42, 50, 59, 62]. When exploiting a side channel, an attacker needs to extract a secret from the observation of side effects generated by a victim's execution patterns. For relatively complicated applications such as website fingerprinting [64, 73], document fingerprinting [49], and acoustic side channels [2, 18], the relationship between these observations and the secret can be difficult to identify. However, machine learning models make it possible to solve this problem with relatively high accuracy. This has led to major improvements for many side-channel attacks [4, 7, 12, 29, 30, 35, 40, 42, 50, 52, 59, 62, 64, 65].

Unfortunately, the usage of machine learning in these settings has incited a worrying trend. Machine learning models are very good at finding correlations, which enables them to be used regardless of one's understanding of the side channel being attacked. This leads to the development of powerful attacks that are poorly understood. Without proper analyses, the community is left unable to develop effective countermeasures or strategies to close the side channels entirely.

*This Paper.* In this paper, we provide such an analysis for an attack that had not been entirely understood until now. Specifically, we show that the website-fingerprinting attack proposed by Shusterman et al. [64], also known as a *sweep-counting attack*, does not primarily leverage signals generated by cache contention. In fact, we show that system interrupts are the primary source of information leakage in this attack.

In a sweep-counting attack, the attacker begins by allocating a buffer with the same size as the last-level cache (LLC). Then, over a series of short time intervals, the attacker repeatedly records *cache-sweep counts* by accessing the entire buffer multiple times and storing, into a trace, the number of times it was able to do so. When the attacker process is running in parallel with the processes rendering a victim's tab, the generated trace is characteristic of activity performed by the website loaded by the victim and can be used as that website's "fingerprint." The attack is implemented in JavaScript, allowing it to be embedded in any website. It also achieves relatively high accuracy on most web browsers and operating systems.

However, we find that in this sweep-counting attack, attributing the attack's success to the cache doesn't tell the full story. Our analysis challenges the previous assumption and shows that this attack primarily exploits interrupt-based side channels. We present two supporting arguments.

First, we demonstrate that the attack works well even after all memory accesses are removed. In our new attack, which we call a *loop-counting attack*, the attacker repeatedly increments a counter in a loop and periodically saves it to a trace. We then evaluate our loop-counting attack using website fingerprinting as an established benchmark and compare our results with the state-of-the-art cache-based attack introduced in [65]. We test multiple combinations of operating systems (Linux, Windows, and macOS) and web browsers (Chrome, Firefox, Safari, and Tor Browser). Our attack can distinguish between 100 websites with accuracy as high as 96.6% in Chrome and Safari, and as high as 95.3% in Firefox. In all but one experimental configuration, our attacker outperforms the state-of-the-art [65]. These observations show that other side channels likely play an important role in the sweep-counting attack's success.

Second, in a more controlled experiment, we directly compare the accuracy of the sweep-counting attack to our loop-counting attack and show that the extensive memory accesses made by the sweep-counting attack actually inhibit its performance. We also observe that its accuracy is much more affected by noise introduced by generating interrupts randomly than by repeatedly sweeping the last-level cache. From these observations, we hypothesize that the primary source of information leakage in both the sweep-counting attack and our loop-counting attack comes from interrupt-based side channels.

Intuitively, in both of these attacks, instruction throughput should decrease as the attacker's CPU core spends more time handling system interrupts triggered by the victim. We verify our hypothesis about a potential interrupt-based side channel with additional analysis.

First, we investigate how our loop-counting attack performs under different isolation mechanisms. Our findings make it possible to rule out various hypotheses regarding the attack's primary source of information leakage, such as frequency scaling and CPU resource contention, both of which are also able to affect instruction throughput. Our analysis shows that existing isolation mechanisms, including virtual machine isolation, do not defend well against our attack.

Second, we develop a tool that uses eBPF to instrument the Linux kernel, which helps us prove that the primary source of leakage in the loop-counting attack comes from system interrupts. We find that over 99% of our attacker's execution gaps lasting longer than 100 nanoseconds are caused by interrupts. We additionally leverage our eBPF tool to study the timing characteristics of different types of interrupts.

In our analysis, for the first time in the literature, we find that *non-movable interrupts*, such as softirqs and rescheduling interrupts, play an important role in leaking application information. This finding is important as nearly all prior work studying interrupt-based side channels [43, 68] focuses on *movable interrupts* such as graphics and network interrupts. Linux provides convenient interfaces to block information leakage due to movable interrupts by isolating them from potential attackers. However, preventing leakage due to non-movable interrupts may require major system redesigns.

Finally, we use the insights from our analysis to propose and evaluate two immediate countermeasures against our attack. One of these adds randomness to the attacker's timer, and a second introduces noise by generating interrupts at random. These countermeasures reduce our attack's accuracy in Chrome on Linux from 96.6% to 5.2% and 70.7% respectively.

*Contributions.* The main contributions of this paper can be summarized as follows:

- We demonstrate that in the sweep-counting attack presented in [64], cache-based side channels are not the primary source of information leakage. Instead, we provide analysis showing that the attack's primary source of leakage comes from system interrupts.
- We show that interrupt-based side channels can be used to mount a powerful website-fingerprinting attack that outperforms the state-of-the-art [65].
- We highlight the security implications of non-movable interrupts, which have not been studied in detail.
- We open-source our trace collection, model training, and eBPF toolset at <https://github.com/jackcook/bigger-fish>.

Our work highlights the importance of thoroughly analyzing side-channel attacks, especially those assisted by machine-learning techniques. We hope this work raises awareness about the limitations of machine learning as a tool and motivates the community to develop better methodologies for analyzing side channels.

## 2 BACKGROUND

### 2.1 Cache-Based Side Channels

Cache-based side channels [37, 46, 64, 80] have been extensively studied in the literature to leak various types of sensitive information, including cryptographic keys [46, 55, 80, 81], user activity [43, 44, 54], and website content [64, 65]. Many cache attacks, such as Prime+Probe [46, 55], work by creating contention on attacker-chosen cache sets between the attacker and victim and using high-resolution timers to monitor the cache.

Shusterman et al. [64, 65] proposed a cache-occupancy attack that requires no detailed knowledge of the cache's organization, such as knowledge of cache associativity and address mapping functions, and can work with low-resolution timers, notably including the timers provided by a web browser. Their attack works by measuring the average memory activity of a victim. Specifically, the attacker repeatedly measures the time it takes to access a last-level-cache-sized buffer. When the victim performs some memory accesses, it loads the corresponding data in the cache, evicting cache lines that have been previously loaded by the attacker. When the attacker executes again, it should take a longer time for the attacker to access lines that are no longer present in the cache. This timing information has been believed to be enough to infer the number of cache lines that have been evicted, which is the "cache occupancy" status of the victim. This attack is not fine-grained enough to recover cryptographic keys, but is powerful enough to predict classes of activity performed by the victim, such as web browsing activity. In this paper, we show that contrary to popular belief, this attack relies more on interrupt-based side channels than cache-based ones.

### 2.2 Interrupts

The attacks introduced in this paper primarily exploit interrupt-based side channels. Thus, we give a brief introduction to interrupt mechanisms and different interrupt types.

System interrupts are events triggered by hardware devices or software that require immediate responses. It is a hardware mechanism used to deliver information from the outside world or to otherwise signal an event that requires software attention. When an interrupt is raised, it is routed to one of the CPU cores where it triggers a special piece of code to save the current context and start executing in kernel mode. The operating system then determines the cause of the interrupt and triggers the appropriate interrupt handler. During this process, the task that was previously running on the core remains paused. Once the interrupt handler has completed, the scheduler either resumes the previous task where it left off, or context switches to another process. User code is normally oblivious to interrupts that happen while it is running, though a particularly attentive process might notice a small jump in the wall clock time.

Interrupts are notably one of the main asynchronous mechanisms that stall a program's execution. We describe several types of interrupts that are relevant to our attacks, classifying them by how they are triggered below.

*Device Interrupts (IRQs).* Device interrupts are triggered by hardware devices, such as USB and PCI devices, to signal external events, like the arrival of a network packet, or the completion of queued

work, such as writing a disk block. Each device interrupt is associated with a device source.

Operating systems have various policies for how they balance device interrupts between different cores, but often interrupts are either routed to one specific core based on the interrupt source or distributed among all cores equally. In either case, the process running on a core when an interrupt arrives may not belong to the same process as the work that triggered the interrupt. Linux provides a command line tool, `irqbalance`, through which users can specify the interrupt distribution policy to improve system performance.

*Local Interrupts.* Local interrupts originate from within the CPU itself. For instance, processors contain programmable timers that raise timer interrupts after a specified amount of time has elapsed. Timer interrupts are critical to the operation of schedulers in modern operating systems, because these schedulers rely on timer interrupts to preempt a process when its time slice expires, instead of counting on the process to yield voluntarily.

*Inter-Processor Interrupts (IPIs).* Interrupts can also be triggered by operating system software running on remote cores. For instance, when the operating system makes updates to page tables that require invalidation of TLB entries stored on other cores of the system, it uses an inter-processor interrupt to perform the required TLB shutdown.

*Softirqs and IRQ Work.* Softirqs and IRQ Work interrupts are Linux software constructs that are used to perform some of the work associated with handling device interrupts. They allow for tasks to be queued to run at a more convenient time, often when the OS is already handling a timer interrupt. Unfortunately, Linux does not provide any interface for users to configure when and where they are processed.

### 2.3 Interrupt-Based Timing Side Channels

Interrupt-based timing side channels leak information based on CPU time used for handling interrupts. These side channels involve a victim and an attacker, where the victim can be a user-space program or the operating system. The goal of the attacker is to figure out whether and when the operating system receives and processes interrupts. The attacker program executes on a CPU core and repeatedly probes a clock to record the time.

Figure 1 shows an example of an interrupt-based timing side channel attack. The attacker and victim processes are executing on two different cores. When there is no activity on the machine, the attacker can continuously execute on the CPU (e.g. before  $T_1$ ), and the observed timer samples increment steadily. However, the attacker's observation of the timer will be different if the victim issues a request to an external hardware device, such as sending a network request to a NIC card. When the requested packet arrives at the machine, the NIC card will generate a network interrupt, which can be routed to the attacker's core depending on the interrupt distribution policy used by the operating system. When the interrupt arrives at the attacker's core at  $T_1$ , the CPU pauses the attacker process and starts executing an interrupt handler. Only after the interrupt handler is completed at  $T_2$  can the operating system resume the attacker process. In this case, the attacker will observe a

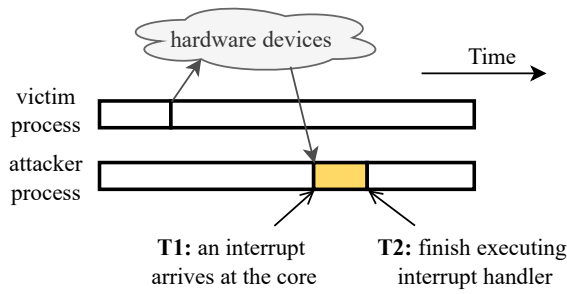


Figure 1: An example interrupt-based timing side channel.

“jump” in the observed timer samples. Based on the timer samples, the attacker can figure out when the interrupt was triggered and measure the amount of time used by the interrupt handler, which is  $T2 - T1$  in the above example.

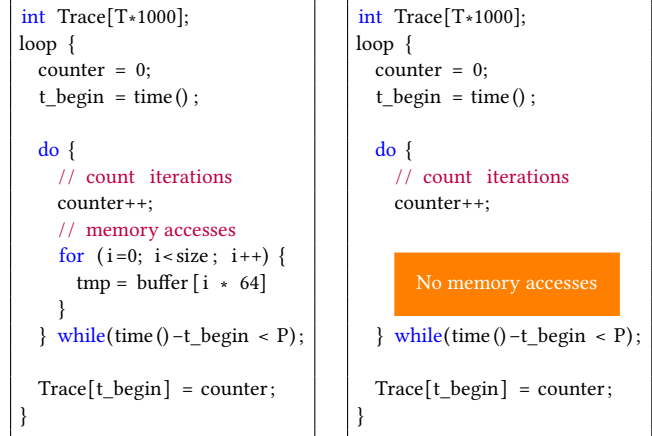
Note that interrupt-based timing side channel attacks are different from attacks that directly read system files to obtain interrupt statistics. We provide a detailed discussion on these attacks in Section 7.

## 2.4 Website-Fingerprinting Attacks and Defenses

Throughout the paper, we use website fingerprinting as an established benchmark for evaluating and comparing the effectiveness of side-channel attacks. Website fingerprinting is a type of attack where an attacker tries to distinguish which website is visited by a victim. Knowing which website a user connects to is more than sufficient to obtain detailed information about a victim, such as religious beliefs, sexual orientation, and political views. There exist many variants of website-fingerprinting attacks, which we can classify into two categories based on the resources that can be accessed by the attacker: *on-path attacks* and *co-located attacks*.

An on-path attacker executes on a different machine from the victim. The attacker observes all the network packets sent and received by the victim’s machine and infers the website based on the timing and size of the observed network packets [10, 20, 25, 27, 28, 32, 33, 41, 47, 56, 57, 60, 75, 76]. Such an attacker can only observe the network-level activity of the victim, and thus can be mitigated by obfuscating network traffic patterns [8, 9, 11, 53, 74, 77].

A co-located attacker executes on the same machine as the victim and shares multiple micro-architectural resources with the victim, including caches, DRAM, and GPUs. In the case of a low-privileged attacker, the co-location can be achieved by observing the victim accessing a malicious website running attacker-controlled JavaScript code. Prior work has demonstrated the usage of micro-architectural timing side channels to distinguish different websites with high accuracy [19, 23, 52, 54, 64, 65]. For example, Shusterman et al. [65] proposed a website-fingerprinting attack that can distinguish 100 websites with accuracy up to 92%. The attack works by collecting traces reflecting cache occupancy while the browser loads and renders websites, and then uses deep neural networks to classify the traces correctly. We discuss other co-located website-fingerprinting attacks in Section 7.



(a) Sweep-counting attack

(b) Loop-counting attack

Figure 2: Pseudo-code of the sweep-counting attack (a) and our loop-counting attack (b).

*Threat Model.* In this paper, we use interrupt-based timing side channels to perform website-fingerprinting attacks as a recurring example for our analysis. In this context, we follow the co-located attack model. Unless explicitly stated otherwise, our attack’s code is implemented in JavaScript and is restricted to the low-resolution timer provided by the web browser being used.

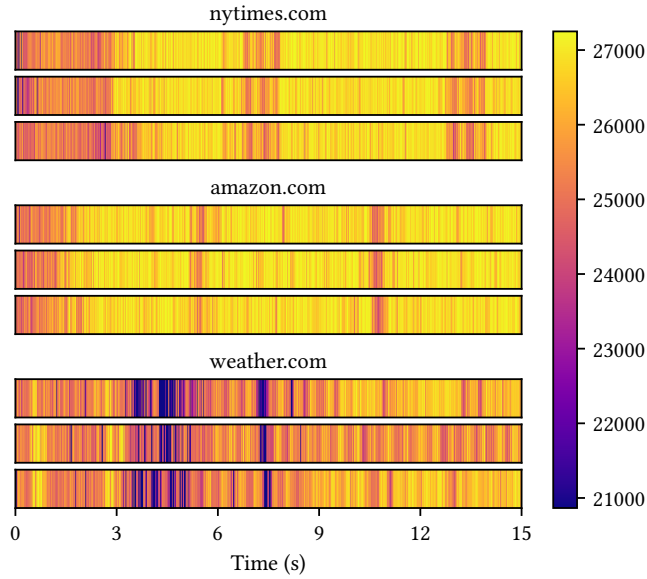
## 3 THE LOOP-COUNTING ATTACK

In this section, we build an attacker program that is almost identical to the sweep-counting attack [64], but does not perform any memory accesses. We call our new attack a *loop-counting attack*. We show that the profiles of the traces collected by the two attackers are highly correlated, suggesting that only a small amount of information is lost when foregoing cache accesses.

### 3.1 Attack Description

We show pseudo-code for a sweep-counting attacker in Figure 2a and our loop-counting attacker in Figure 2b. In both algorithms, the attacker takes a parameter of period length  $P$  as input. It then constructs a trace, where each element in the trace measures how many iterations of the inner-most loop were executed every  $P$  milliseconds. In the sweep-counting attack’s code, the loop body contains an increment operation, memory accesses to a large buffer, and a call to the `time()` function. Note that the buffer’s size matches the size of the last-level cache so that one completion of the inner loop sweeps the entire last-level cache. The counter value can thus be used to infer how many of the accessed cache lines reside in the cache. Conversely, in the loop-counting attack’s code, we make no memory accesses inside the inner-most loop, but instead only have an increment instruction and a call to the `time()` function.

Since the sweep-counting attack’s code (in Figure 2a) measures the throughput of memory accesses, it was previously believed that the resulting trace was a good proxy for memory throughput and cache occupancy over time. However, in addition to cache hits and misses, the throughput of memory instructions can be



**Figure 3: Example loop-counting traces collected over 15 seconds. Darker shades indicate smaller counter values and lower instruction throughput.**

affected by many other factors, including interrupts, which have been overlooked by previous work.

Intuitively, within a given period of time, if the program is preempted by an interrupt handler, the attacker spends less time executing the loop and thus fewer iterations can be completed, leading to smaller trace values. Therefore, traces can capture a large amount of timing information from system interrupts. It is still possible that our trace includes signals from other sources. We examine this possibility in detail in Section 5.

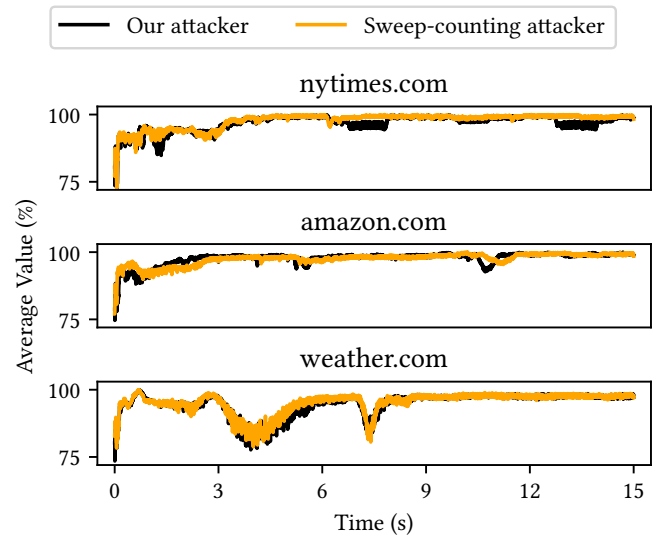
### 3.2 Trace Examples

In Figure 3, we present loop-counting traces generated using the algorithm from Figure 2b on three victim websites (nytimes.com, amazon.com, and weather.com).

We implement our attacker in JavaScript as a service worker. Service workers run in the background on a separate thread from the sites hosting them. Web browsers do not degrade the performance of service workers when their host tabs leave the foreground, allowing our attack to work well even when the attacker's tab is not visible.

Each trace is collected while a victim is browsing potentially sensitive websites in a different tab. The JavaScript attacker repeatedly calls `performance.now()` to measure the time, which is returned at a low precision and often with some added noise, depending on the browser being used. We generally pick a value for the period length  $P$  that is larger than the timer resolution provided by the browser in order to accurately measure throughput. For consistency, if not explicitly stated otherwise, we set  $P$  to 5 milliseconds.

We collect each trace by running the loop-counting attack code (Figure 2b) for 15 seconds while the browser loads and renders a website in another tab. The shades in the trace correspond to



**Figure 4: Normalized trace values averaged over 100 runs while loading three websites. Traces collected with both attackers are strongly correlated with each other.**

counter values ranging approximately from 21,000 to 27,000. Darker shades represent smaller counter values, indicating that the attacker is preempted by interrupts and paused for more time.

We observe that traces for the same website are similar to each other, while traces for different websites are quite different. For example, according to the traces in Figure 3, we can infer that amazon.com performs much of its activity in the first 2 seconds, with spikes in activity around 5 and 10 seconds. Visual cues such as these indicate that loop-counting traces can be used as fingerprints to distinguish different websites.

### 3.3 Comparing Attacker Traces

In Figure 4, we compare the behavior of traces collected by the loop-counting and the sweep-counting attackers. Each plot depicts averaged traces over 100 runs from its respective website, which were then normalized by dividing each value by the maximum iteration count observed by that attacker. This maximum count was about 27,000 loop iterations for our attacker, and 32 for the sweep-counting attacker. The averaged traces from each attacker are strongly correlated with each other, with a correlation coefficient of  $r = 0.87$  for nytimes.com,  $0.79$  for amazon.com, and  $0.94$  for weather.com, suggesting that traces collected by the sweep-counting attacker are shaped by the same system events as those collected by the loop-counting attacker.

## 4 EVALUATION

In this section, we demonstrate that the sweep-counting attack by Shusterman et al. [64] does not primarily rely on cache-based side channels. Specifically, we evaluate our loop-counting attack against a state-of-the-art cache-based website-fingerprinting attack [65] and find that our attack outperforms it in all but one experimental configuration. We also find that the accuracy of the



sweep-counting attack is only mildly affected by noise introduced by sweeping the last-level cache. By contrast, we find that noise introduced by randomly-generated interrupts significantly inhibits its performance, leading us to believe that the primary source of information leakage in the sweep-counting attack comes from system interrupts.

#### 4.1 Evaluation Setup

To compare the two attacker programs, we use website fingerprinting as a benchmark. Specifically, we compare our work to the state-of-the-art “cache-occupancy” attack also presented by Shusterman et al. [65]. To make a fair comparison, we closely follow their methodology, including experimental configurations, data collection methods, machine learning model, and hyperparameters.

Like Shusterman et al., we leverage machine learning to create an attack consisting of two phases: an *offline training phase* and an *online attack phase*. In the offline training phase, the attacker collects a dataset composed of labeled traces, annotated with their corresponding websites, and uses the dataset to train a machine learning classifier. This step is performed on a machine under the attacker’s control. In the online attack phase, the attacker collects a trace while the victim visits an unknown website. The attacker then uses the trained classifier to predict which website was visited by the victim.

*Data Collection.* We perform our evaluation under two setups: a *closed-world setup* and an *open-world setup*.

In the closed-world setup, the attacker knows the complete set of the websites that the victim will visit. In our experiments, the attacker aims to distinguish the victim’s accesses of 100 different websites. We select the top 100 websites according to Alexa<sup>1</sup>, excluding websites that 1) are pornographic in nature, 2) serve essentially the same content (e.g. if google.com is included, google.co.uk will not be included), and 3) experience frequent issues with our test infrastructure (see Appendix A).

We collect 100 traces from each of the 100 websites, forming a dataset of 10,000 traces. The browser’s cache is not cleared before accessing each website, mirroring typical user behavior.

In the open-world setup, the attacker does not have full knowledge of the websites that will be visited by the victim. Instead, the victim will access a set of “sensitive” and “non-sensitive” websites. The attacker only knows the set of sensitive websites, and aims to precisely identify the victim’s accesses to these websites. When the victim visits a non-sensitive website, the attacker should simply report “non-sensitive”.

To perform the evaluation for the open-world setup, we treat the 100 websites used in the closed-world setup as sensitive websites and label these traces with their URLs. We then collect 5,000 additional traces, each corresponding to a single unique website. These websites are also chosen from the Alexa top sites list, and the 5,000 traces are labeled as non-sensitive.

*Machine Learning Model.* We use a Long Short-Term Memory (LSTM) network to train our machine learning classifier, as LSTM

networks are highly capable of identifying patterns in temporal data. We use the same model and hyperparameters<sup>2</sup> as [65].

To train the classifier, we use standard 10-fold cross validation to avoid overfitting. We split the dataset into 10 folds, each containing 10% of the traces. We then select one fold and use it as a held-out test set, and further split the remaining traces into two sets: a training set with 81% of the traces and a validation set with the remaining 9% of the traces. We train the classifier using the training set, stop training when the validation accuracy starts decreasing, and report the prediction accuracy by applying the trained classifier on the held-out test set. This procedure is repeated for each fold, and the average accuracy across the 10 folds is reported as the final accuracy.

#### 4.2 Evaluation Results

In Table 1, for each combination of web browser and operating system, we report our attack’s accuracy in our closed- and open-world setups. We obtain traces from five machines. We use three desktop computers, each with an Intel Core-i5 processor running Ubuntu 20.04, a workstation computer with an Intel Xeon processor running Windows 10 Enterprise, and a MacBook with an Intel Core-i5 processor running macOS Big Sur 11.5. We evaluate our attack on four commercial web browsers: Chrome 92, Firefox 91, Safari 14, and Tor Browser 10. Trace collection is automated with the Selenium browser automation framework [1].

Tor Browser is a modified version of Firefox with additional security features intended to block local side channel attacks. Due to these security features, it takes noticeably longer to load a page on Tor Browser. We use 15-second traces when attacking Chrome, Firefox, and Safari, and 50-second traces when attacking Tor Browser.

*Closed-world Results.* In the closed-world setup, our attack has a base success rate of 1%, since our classifier makes a prediction out of 100 possible websites. These websites were selected according to the procedure detailed in Section 4.1. We directly compare the accuracy of our attack to the accuracy of the cache-occupancy attack from [65]. Note that the contents of the top 100 Alexa websites have changed since [65] was published, and we perform our work on newer operating systems and web browsers. We bold the accuracy for experiments where our attack achieves higher accuracy.

When attacking Chrome, which reduces the timer’s resolution to 0.1 milliseconds and adds random “jitter,” our attack is stronger for every experimental setup. For example, when attacking Chrome running in Windows in a closed-world setup, our attack achieves an accuracy of 92.5%, while the cache-occupancy attack only achieves a success rate of 80.0%. Our attack’s accuracy does not degrade when attacking Firefox and Safari, which use less precise timer resolutions of 1ms.

Tor Browser uses an extremely low timer resolution of 100ms. Our attack is still effective on Tor Browser, though with significantly reduced accuracy. In the closed-world setup, our attack’s accuracy is 49.8%, and the top-5 accuracy, the rate at which the correct website is one of the model’s top five predictions, is 86.4%. Our

<sup>1</sup><https://www.alexa.com/topsites>, visited on July 9, 2021.

<sup>2</sup>LSTM (32 units, sigmoid activation) with 2 pairs of convolutional layers (256 filters, stride = 3, ReLU activation) and max pooling layers (pool size = 4), a dropout layer (rate = 0.7), and a fully connected classification layer (output size = 100, softmax activation). We use the Adam optimizer with learning rate = 0.001.

**Table 1: Classification accuracy obtained with JavaScript loop-counting attacker.**

Browser	Timer Resolution	Operating System	Closed World		Open World			
			Loop-Counting Attack	Cache Attack [65] <sup>†</sup>	Loop-Counting Attack			Cache Attack [65] <sup>†</sup>
					Sensitive	Non-Sensitive	Combined Accuracy	Combined Accuracy
Chrome 92	0.1ms w/ jitter	Linux	<b>96.6</b> ±0.8	91.4±1.2	95.8±0.8	99.4±0.3	<b>97.2</b> ±0.3	86.4±0.3
		Windows	<b>92.5</b> ±1.0	80.0±1.6	91.4±0.8	99.2±0.6	<b>94.5</b> ±0.5	86.1±0.8
		macOS	<b>94.4</b> ±1.0	–	92.4±1.0	97.6±1.0	<b>94.3</b> ±1.0	–
Firefox 91	1ms w/ jitter	Linux	<b>95.3</b> ±0.7	80.0±0.6	95.2±1.0	99.9±0.1	<b>96.4</b> ±0.8	87.4±1.2
		Windows	<b>91.9</b> ±1.2	87.7±0.8	90.9±0.9	99.6±0.3	<b>93.7</b> ±0.6	87.7±0.3
		macOS	<b>94.4</b> ±0.8	–	93.5±1.1	98.6±0.7	<b>95.0</b> ±0.8	–
Safari 14	1ms	macOS	<b>96.6</b> ±0.5	72.6±1.3	95.1±0.9	99.0±0.7	<b>96.7</b> ±0.6	80.5±1.0
Tor Browser 10	100ms	Linux	<b>49.8</b> ±4.2	46.7±4.1	46.2±2.9	89.8±2.2	62.9±2.4	62.9±3.3
Tor Browser 10 (top 5)	100ms	Linux	<b>86.4</b> ±2.6	71.9±2.1	86.2±2.1	97.5±1.0	<b>90.7</b> ±1.2	82.7±1.8

<sup>†</sup> We use the same data collection method and LSTM-based model as [64] and [65]. However, the following differences exist between our evaluation setup and theirs: 1) [65] performed their experiments in 2018 and thus used older operating systems and web browsers, 2) Firefox has changed its timer resolution from 2ms in 2018 to the current 1ms, and 3) we use different Intel CPUs.

attack consistently performs at least as well as the cache-occupancy attack [65].

To check whether the resulting differences between the two attacks are meaningful, we use a standard 2-sample t-test to compute the statistical significance of our classifier compared to the classifier from [65]. Our results are always significant over [65] with  $p < 0.0001$ , except for the Tor Browser top-1 result for the closed-world setup, which is significant with  $p < 0.05$ .

*Open-world Results.* In our open-world setup, as explained in Section 4.1, we add 5,000 “non-sensitive” traces to our existing collection of 10,000 “sensitive” closed-world traces to make a complete dataset of 15,000 traces. We then train a new model with 101 classes: one for each sensitive website, and an additional “non-sensitive” class with all 5,000 open-world traces. This experimental design is identical to the design used in [65].

We report the base accuracy and standard deviation of this model for each experiment in the “loop-counting attack” column of Table 1, along with the respective accuracy from [65]. We bold the accuracy for experiments where our attack achieves higher accuracy, and note that our attack is stronger in all experiments except for Tor Browser, in which we achieve the same accuracy. We refer to these accuracy metrics as the model’s “combined accuracy.”

The base success rate in these experiments is 33%, which is the success rate of a blind guess of “non-sensitive.” This makes open-world accuracy figures difficult to compare to closed-world accuracy figures, since the closed-world models have a base success rate of 1% by comparison. To reduce confusion, we also report the model’s accuracy on sensitive and non-sensitive sites individually.

*Robustness to Background Noise.* We additionally test our attack’s performance in the presence of system-wide background noise on Chrome. In the absence of noise, our attack achieves 96.6% accuracy, as shown in Table 2. When running Slack and Spotify (playing music) alongside our attacker, we observe a drop of a just

few points in accuracy, down to 93.4%. This indicates that other applications do not generate enough noise to have a significant impact on our attack. This also strengthens our attack’s practicality and shows that it could be used in a real-world setting.

*Interpretation.* The results we report in Table 1 show that our loop-counting attacker, which makes no memory accesses, consistently outperforms the state-of-the-art website-fingerprinting attack. This finding demonstrates that side channels that do not leverage the cache can be exploited to create a powerful attack. This leads us to believe that the sweep-counting attack may already be using other sources of information leakage. We analyze this possibility in depth in Section 5.

**Takeaway 1:** Side channels other than the cache provide enough signal to craft a powerful website-fingerprinting attack.

### 4.3 Comparison to the Sweep-Counting Attack

We now perform a more controlled experiment to exclude variations due to differences in website content and machine configuration. Specifically, we directly compare the sweep-counting and the loop-counting attackers with the exact same configuration, running all experiments on the same machine attacking Chrome 100, with an Intel Core-i5 processor running Ubuntu 20.04.

We additionally compare the effects of two noise-injection countermeasures on each attacker. The first countermeasure, proposed by [65], introduces *cache-sweep* noise by repeatedly sweeping the cache, causing evictions from each cache line. The second countermeasure, introduced by us, introduces noise by generating spurious interrupts. We present the countermeasure effects in this section and defer the discussion of the technical details of the new countermeasure to Section 6.2.



**Table 2: Classification accuracy obtained with our loop-counting attack and the sweep-counting attack [64] in the presence of different sources of noise.**

Attack	No Noise	Cache-Sweep Noise	Interrupt Noise
Loop-Counting	95.7%	92.6%	62.0%
Sweep-Counting [64]	78.4%	76.2%	55.3%

We present our results in Table 2. Our attack achieves significantly higher accuracy out of 100 websites in all three scenarios. Notably, the sweep-counting attack is far more sensitive to interrupt noise than cache noise. The cache-sweep noise reduces the sweep-counting attack’s accuracy by 2.2%, while the interrupt noise reduces the attack’s accuracy by 23.1%, a relatively large drop in performance. The decreases in performance due to the two sources of noise is comparable to the responses exhibited by our interrupt-based attacker. Therefore, we hypothesize that these attacks may both exploit interrupt- and cache-based side channels, but that system interrupts constitute the primary source of information leakage.

**Takeaway 2:** Cache contention appears not to be the primary source of information leakage in the sweep-counting attack presented by Shusterman et al. [64].

## 5 THE PRIMARY SOURCE OF LEAKAGE

It is important to analyze how signals that enable attacks originate and propagate within a system. Without properly analyzing the various sources of leakage that power a side channel, the community is left unable to develop effective countermeasures or strategies to close the side channel entirely. In this section, we obtain a thorough understanding of the loop-counting attack presented in Section 3 and its underlying interrupt-based timing side channel.

### 5.1 Effects of Isolation Mechanisms

We start our analysis by looking at how different isolation mechanisms affect our loop-counting attack. We use an attacker program that implements the algorithm from Figure 2b in Python. The Python code calls the `time.time()` function to read the system timer. In contrast with our JavaScript attacker, when using a Python attacker, we avoid all potential interference from the web browser, such as global event queues and low-precision timers.

Table 3 shows classification accuracy when evaluating the Python attack code on Chrome 92 in a closed-world setup. Note that we create each configuration by incrementally adding a new isolation mechanism in addition to the mechanisms from the previous configuration. For example, the last configuration inherits all isolation mechanisms from previous configurations, including disabling frequency scaling, pinning to separate cores, removing IRQ interrupts, and additionally running the attacker process and the victim process in two separate virtual machines.

**Table 3: Classification accuracy obtained with Python loop-counting attacker under various isolation mechanisms.**

Isolation Mechanism	Top-1 Accuracy	Top-5 Accuracy
Default	95.2%	99.1%
+ Disable frequency scaling	94.2%	98.6%
+ Pin to separate cores	94.0%	98.3%
+ Remove IRQ interrupts	88.2%	97.3%
+ Run in separate VMs	91.6%	97.3%

*Disable Frequency Scaling.* Recall that the loop-counting attack code measures instruction throughput, which can be impacted by processor frequency scaling. Therefore, we first verify whether the signals used by our attack are affected by frequency scaling. Specifically, our test machine has a clock speed of 1.6-3GHz. We disable frequency scaling by using the Linux command `cpufreq-set` to fix the CPU frequency at 2.5GHz. We observe a small decrease of 1% in the top-1 accuracy, which indicates that our attack is still highly effective even without any signal due to frequency scaling.

*Separate Cores.* We also investigate whether our attack is caused by contention of CPU resources, which can happen when the operating system schedules the attacker process and the victim process onto the same core. On our 4-core machine without hyper-threading, we use the Linux command `taskset` to pin the attacker process to logical core 1 and the victim browser to logical core 2 to avoid scheduling contention. We observe a negligible decrease of 0.2% in the attack’s accuracy, indicating that our attack does not rely on CPU contention.

*Remove IRQ Interrupts.* As discussed in Section 2, there exist many types of interrupts. For several IRQ types, such as graphics interrupts, network interrupts, and SATA interrupts, there is no restriction on where these interrupts should be processed. We use the Linux command `irqbalance` to bind all movable IRQs to logical core 0 in order to prevent them from interfering with the attacker process. Timer interrupts, softirqs, rescheduling interrupts, and TLB shootdowns are left on the attacker’s core, since these interrupts execute on all cores and the operating system does not provide an interface to move them.

We observe moderate decreases of 5.8% and 1% in the top-1 and top-5 accuracy respectively. There are two takeaways from these results. First, the timing characteristics of device interrupts play an important role in leaking website identity, given that different websites trigger characteristic network and graphics activities. Second, removing all movable interrupts is not sufficient to mitigate the attack: the remaining non-movable interrupts might play an important role in leaking the victim’s activity. We demonstrate the essential role of these interrupts in Section 5.2.

*Run in Separate VMs.* Finally, we evaluate how our interrupt-based side channel performs under a stronger isolation mechanism using virtual machines. We run the attacker and victim processes in two separate VMs, with all isolation mechanisms from the previous configurations enabled. We observe a slight increase of 3.4% in the top-1 accuracy. Such an observation contradicts our expectation, as

one would expect the stronger isolation offered by virtual machines would reduce the effectiveness of our attack. A plausible explanation for this phenomenon is that when routing an interrupt to a core running a VM, the signal is amplified as the interrupt needs to be processed by both the host OS and the guest OS. For example, VM entries and exits are generally much more expensive than process-level context switches. This observation has worrying implications for cloud computing environments, which rely on similar isolation mechanisms to separate different clients.

**Takeaway 3:** Current isolation mechanisms are insufficient to completely mitigate the loop-counting attack, which monitors instruction throughput for website fingerprinting.

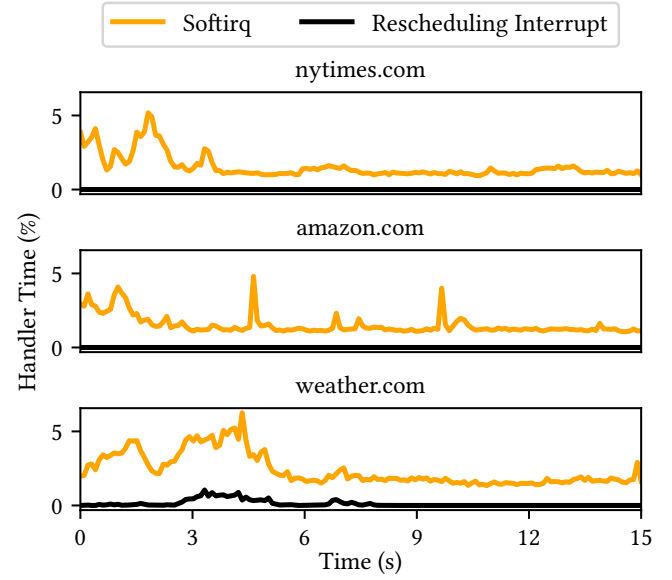
## 5.2 Identifying the Underlying Side Channel

We now use the instrumentation functionalities provided by Linux to analyze the loop-counting attack in more detail and uncover the underlying side channels being exploited. In particular, Linux allows setting kprobes and tracepoints at specific points in the kernel binary. When execution reaches those points, Linux will call into small user-provided programs which can inspect the current system state and record data. To ensure safety, these programs are specified using the restricted API of eBPF byte code [34].

*Analysis Tool.* We use eBPF to log the timestamp and root cause of various types of interrupts arriving at a specific core and then compare them to the gaps observed by a user-space attacker pinned to the same CPU core. In this case, our attacker is written in Rust and watches for jumps in the local time by repeatedly reading from Linux's `CLOCK_MONOTONIC` time source. Reading from the monotonic clock is slower than directly accessing the CPU timestamp counter, but still incurs very little overhead because it is implemented via vDSO (virtual dynamic shared object). Since eBPF also has access to the monotonic clock, time measurements are synchronized between the two programs and we can attribute specific interrupts recorded in the kernel with specific gaps observed by the user-space attacker.

One limitation we face is that Linux restricts which kernel functions can be traced<sup>3</sup>, with recent versions (5.11 and later) being slightly less restrictive. This means that we are unable to monitor all entry points into the operating system, but nonetheless when running with Intel Turbo Boost disabled<sup>4</sup>, our tool detects that the overwhelming majority of the gaps are caused by an interrupt type that we are able to monitor.

*Analysis Results.* Figure 5 reports the overall time spent in interrupt handlers during each 100-millisecond interval of averaged traces over 100 runs from three popular websites, sample traces for which were presented in Section 3. In this experiment, we use `irqbalance` to prevent the attacker's core from receiving IRQs, so that almost all observable execution gaps come from non-movable



**Figure 5: Percentage of time spent processing interrupts averaged over 100 runs while loading three different websites.**

interrupts. Notice that the amount of time spent handling interrupts closely matches the appearance of the traces in Figure 3. For instance, most of the interrupt-handler activity when loading `nytimes.com` happens in the first 4 seconds while for `amazon.com`, we observe spikes at 5 and 10 seconds.

Different websites can even trigger different types of non-movable interrupts. For example, `weather.com` routinely triggers rescheduling interrupts, which we found often occur alongside TLB shootdowns. Identifying the JavaScript-based mechanisms causing different sites to trigger different types of interrupts is left as future work.

Notably, we find that our eBPF tool confirms that over 99% of execution gaps longer than 100 nanoseconds are caused by interrupts. We consider this result to serve as a rigorous proof that our loop-counting attacker primarily exploits signals from system interrupts.

**Takeaway 4:** Our loop-counting attack primarily exploits signals from system interrupts.

*Analysis Implications.* This result highlights the robustness of our attack. Interrupt-based side channels are universal, as interrupts exist on every platform and can be detected by many different attackers. For instance, our traces and the trace of interrupt-handler activity are generated using different attack code (polling `CLOCK_MONOTONIC` versus measuring instruction throughput with Chrome's reduced resolution timer), and written in different programming languages (Rust versus JavaScript).

We additionally identify the security problems associated with non-movable interrupts, such as softirqs and rescheduling interrupts, which have never before been studied in side-channel attacks. For example, softirqs are a mechanism used by the Linux kernel

<sup>3</sup>See <https://lore.kernel.org/lkml/20131101112530.14657.87835.stgit@kbuild-fedora.novalocal/>

<sup>4</sup>When running with Turbo Boost enabled, we observe a significant number of execution gaps that don't seem to correspond with time spent in the OS. Finding the cause of these gaps and whether they represent another side channel is left as future work.

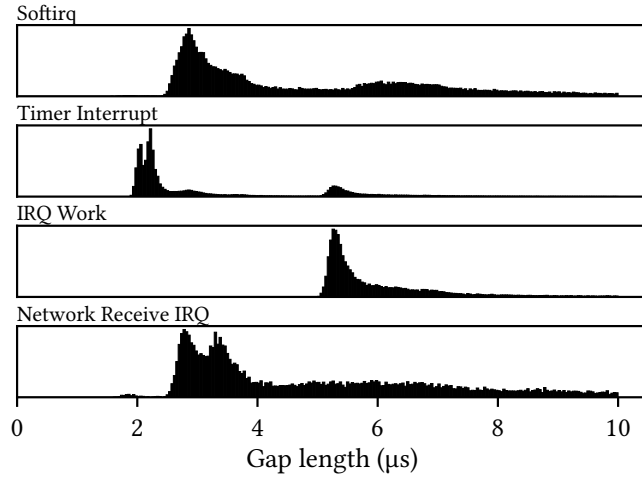


Figure 6: Distributions of interrupt handling times.

to handle complex interrupt-related tasks that are not critical and can be deferred. This helps to keep the kernel responsive and to correctly handle other time-sensitive interrupts. This mechanism is especially helpful for long-running tasks, such as the decryption of a network packet or the launch of an operation on the GPU. A lightweight interrupt handler will simply queue a softirq that will perform the operation at a more appropriate time. The operating system can decide to allocate this softirq to a different core, potentially moving operations onto a core shared with an attacker. Unfortunately, the Linux kernel does not offer any interface to control the dispatching of softirqs.

Truly understanding the causal relationship between non-movable interrupts and other system events would require instrumenting the kernel at a more in-depth level than allowed by eBPF. KUTrace [66] is a good example of such a tool. Since it is infeasible to isolate non-movable interrupts from applications running on the system, our attack highlights the fact that existing isolation mechanisms are ineffective to mitigate interrupt-based side channels. Major system redesigns are necessary to close this side channel.

**Takeaway 5:** Non-movable interrupts, such as softirqs and rescheduling interrupts, leak information from victim processes. Blocking information leakage from these interrupts may require major system redesigns.

### 5.3 Interrupt Characteristics

We further analyze several types of interrupts in Figure 6. The figure shows various interrupt types that occur frequently during our experiments and the distributions of user-space execution gaps that they cause, averaged over 50 page loads spanning 10 websites. Here, we run on a different core to avoid observing network-receive interrupts and IRQ work processing. All of the gaps associated with these interrupts are longer than  $1.5\mu\text{s}$ , due to the high overhead of context switches caused by mitigations for Meltdown and other relevant CPU vulnerabilities.

Multiple interrupts can be associated with a single gap in user-space execution. This is particularly common for softirqs and IRQ work because neither can happen on their own, and thus are typically run while processing a timer interrupt. This is visible in Figure 6. Because the x-axis reflects the total gap length observed by the attacker rather than just the amount of time spent processing that particular interrupt, the spike at  $5.5\mu\text{s}$  for IRQ work matches the spike at that value for timer interrupts (though the scale is different, because the vast majority of timer interrupts do not involve processing IRQ work).

**Takeaway 6:** Different types of interrupts have characteristic handling time distributions.

## 6 COUNTERMEASURES

As we saw in Section 5.2, defending against our attack by closing the interrupt-based side channel will require major system redesigns. As a partial solution, we propose and evaluate two countermeasures against our loop-counting attack.

The first countermeasure explores a randomized timer similar to a “fuzzy timer” that has been analyzed theoretically in prior work [21, 38]. We also provide an analysis of timers that are currently used in real-world browsers, along with the security benefits of each one individually.

The second countermeasure directly injects noise into the interrupt-based side channel. It is implemented in a Chrome extension and generates thousands of interrupts while sites load.

### 6.1 Randomized Timer

Modern browsers have employed several strategies to limit attackers’ observation capabilities by reducing the precision of the timer. Unfortunately, our results from Section 4.2 show that these mechanisms are ineffective. We find key limitations in existing timer fuzzing techniques and propose a new mechanism.

*Existing Timers.* Existing timer fuzzing techniques involve two strategies: reducing the timer’s resolution, and adding *jitter* to the timer’s returned value. The browser has access to a precise timer, which we denote as  $T_{real}$ , and outputs a “fake” timer value that generally deviates from the actual time, which we denote as  $T_{secure}$ . To reduce the timer’s resolution, the browser outputs a quantized timer value using the formula below,

$$T_{secure} = \left\lfloor \frac{T_{real}}{\Delta} \right\rfloor \cdot \Delta$$

where  $\Delta$  is the timer resolution, which is currently 0.1ms in Chrome, 1ms in Firefox and Safari, and 100ms in Tor Browser.

The second technique is to add small perturbations to the timer. For example, Chrome adds jitter using the following formula.<sup>5</sup>

$$T_{secure} = \left\lfloor \frac{T_{real}}{\Delta} \right\rfloor \cdot \Delta + \epsilon, \quad \epsilon \in \{0, \Delta\}$$

Since the timer must increase monotonically,  $\epsilon$  is not picked randomly but computed using a hash function. Given that  $\epsilon$  is either 0

<sup>5</sup><https://chromium-review.googlesource.com/c/chromium/src/+853505>

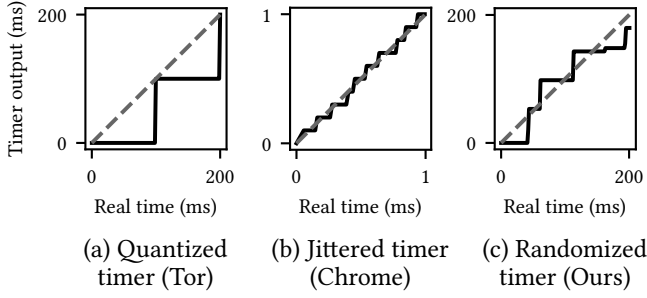


Figure 7: Example outputs of different timers.

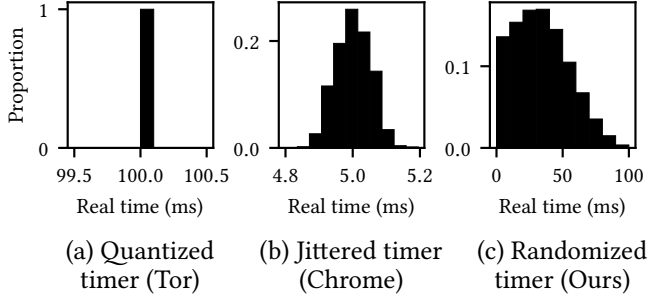


Figure 8: Distributions of durations of one 5-millisecond attacker loop with different timers.

or  $\Delta$ , the difference between the value  $T_{secure}$  and the actual time  $T_{real}$  is guaranteed to be less than  $2 \cdot \Delta$  ms.

**Our Randomized Timer.** We propose a randomized timer that empirically provides stronger security than existing timer-based defense mechanisms. Our timer increases monotonically with *random increments* at *random intervals*. It works as follows. Every  $\Delta$  ms, we generate 2 random integers  $\alpha$  and  $\beta$ . If the difference between the current returned timer value and the real time is within the range of  $\alpha \cdot \Delta$ , the timer value is not changed. If the difference exceeds  $\alpha \cdot \Delta$ , the returned value is updated to  $T_{secure} + \beta \cdot \Delta$ . The computation process is summarized below.

$$T'_{secure} = \begin{cases} T_{secure} & \text{if } T_{real} - T_{secure} \leq \alpha \cdot \Delta \\ T_{secure} + \beta \cdot \Delta & \text{if } \alpha \cdot \Delta < T_{real} - T_{secure} \leq \text{threshold} \\ T_{real} + \beta \cdot \Delta & \text{otherwise} \end{cases}$$

**Comparing Different Timers.** Figure 7 compares the behavior of three secure timers: a quantized timer with a resolution of 100ms (used by Tor Browser) in Figure 7a, a jittered timer with a resolution of 0.1ms (used by Chrome) in Figure 7b, and our randomized timer in Figure 7c. The dashed lines in each plot represent the values that would be returned by a timer with perfect precision. Our randomized timer uses the following parameters:  $\alpha$  and  $\beta$  following a uniform distribution  $\mathcal{U}_{[5,55]}$  and  $\text{threshold} = 100\text{ms}$ .

Figure 8 compares the attacker's performance when restricted to the usage of the same three secure timers. Our loop-counting attacker functions by repeatedly measuring throughput during

Table 4: Classification accuracy obtained with Python loop-counting attacker with different timers.

Timer	$\Delta$ (ms)	$P$ (ms)	Top-1 Accuracy	Top-5 Accuracy
Jittered	0.1	5	96.6%	99.4%
Quantized	100	5	86.0%	96.9%
Randomized	1	5	1.0%	5.1%
		100	1.9%	6.9%
		500	5.2%	13.7%

periods of 5ms. When it loses its ability to do this, its performance degrades significantly. In Figure 8a, we see that when the timer's precision is reduced to 100ms with no randomness, as is done by Tor Browser, the attacker loses its ability to measure 5ms intervals, but can still precisely measure the throughput of 100ms intervals. In Figure 8b, we see that adding small amounts of jitter, as is done by Chrome, slightly inhibits the attacker's ability to do this, as the period lengths vary from 4.8ms to 5.2ms roughly following a Gaussian distribution. However, in Figure 8c, we see that our timer makes it impossible for the attacker to accurately estimate how much time has passed in a single attacker loop, as it could range from 0 to 100ms of real time.

**Evaluation Results.** We implement our randomized timer in Chrome and show the classification accuracy for a closed-world setup in Table 4. When setting the attacker's period length  $P$  (from Figure 2b) to 5ms and using the default jittered timer in Chrome, the attack achieves 96.6% accuracy. A quantized timer with a resolution of 100ms reduces the attack's accuracy to 86%, which is still significantly higher than the 1% base accuracy. When using our randomized timer, the attack's accuracy drops to 1% and the top-5 accuracy drops to 5.1%, almost the same as a blind guess.

Our randomized timer forces the attacker to use a larger attack period length. However, even when the period length is set to 100ms or 500ms, the randomized timer mitigation remains highly effective, as the attack's accuracy remains low, at 1.9% and 5.2% respectively.

In summary, our randomized timer defends well against our attack. It offers a higher timer resolution than the timer used by Tor Browser, making it more practical to use, and it is relatively easy to implement. Our timer should be used in settings where increased security is desired. For applications that require a high-resolution timer in order to function, such as online games, browsers can adopt a custom permission model which would allow users to grant permission to a high-precision timer on a case-by-case basis [43].

## 6.2 Adding Noise with Spurious Interrupts

**Countermeasure Description.** This countermeasure functions by scheduling thousands of activity bursts and network pings at random intervals, which generates thousands of interrupts. We implement our countermeasure as a Chrome extension and train our model on traces collected while the extension is active. This countermeasure comes with a reasonable cost: adding system noise slows down Chrome. The average load time on the 100 websites used in our closed-world experiments increases slightly when our Chrome

extension is enabled, from 3.12 seconds to 3.61 seconds, a 15.7% increase.

*Evaluation.* We compare the effectiveness of our interrupt-based countermeasure with the cache-sweep countermeasure introduced by [65]. The cache-sweep countermeasure repeatedly evicts the entire last-level cache by allocating an LLC-sized buffer and accessing every cache line in a loop. We present our results in Table 2. Our spurious interrupt countermeasure decreases our attack’s accuracy from 95.7% to 62.0%, while the cache-sweep mitigation only decreases its accuracy to 92.6%. As discussed in Section 4.3, this result supports our finding that the primary source of information leakage in both attacks comes from system interrupts.

## 7 RELATED WORK

### 7.1 Interrupt-Related Attacks

Prior work has looked at attacks that leverage interrupts to breach system privacy in a different way than presented in this paper. These attacks generally require direct access to system files containing interrupt statistics, only consider one type of interrupt, or intentionally introduce interrupts to pause the victim program and boost the effectiveness of other types of side channels.

In Linux, all reported interrupts are counted by the kernel and logged in the system file `/proc/interrupts`, which can be accessed by any process. Several attacks exploit such statistical information to monitor application activities [68], monitor keystroke timings and user behaviors on touch screens [15], and detect website content [48]. Fortunately, these attacks are easy to mitigate as one could simply disable non-privileged access to the interrupt pseudo-file.

Other work uses timing side channels to monitor keystrokes [43, 63, 70]. Note that none of these attacks have considered non-movable interrupts and only consider a simplistic scenario that, as a result, can easily be defeated by handling the keyboard interrupts on a different core than the attacker (see Section 5.1).

Another line of work exploits interrupts to target systems equipped with Intel SGX [13] by periodically pausing the execution of victim programs. For example, SGX-Step [71] and CopyCat [51] introduce high-frequency timer interrupts to pause enclave programs and obtain fine-grained side channel observations. They can achieve the maximal temporal resolution of measuring the victim state while executing every single instruction. SGXlinger [26] exploits the fact that context switches in enclave programs require draining multiple micro-architectural states, including the store buffer. Thus, an attacker can trigger context switches using interrupts and infer the memory intensiveness of an enclave program based on context switch latency. Nemesis [72] exploits the micro-architectural behavior that delays interrupt processing until instruction retirement. Interrupt latency, which varies based on the instructions executed when the interrupt arrives, is used by the attacker to infer the execution state of an enclave program.

### 7.2 Co-located Website-Fingerprinting Attacks

A co-located website-fingerprinting attack executes attacker-controlled code on the same machine as the victim web browser and leaks website content by exploiting system vulnerabilities or shared micro-architectural resources.

Lee et al. [39] discovered a GPU vulnerability in 2014 where both NVIDIA and AMD GPUs did not initialize newly allocated GPU memory. They showed that an unprivileged attacker process could directly read browser data when a GPU was used to accelerate webpage rendering. PerfWeb [24] leaks website identity by exploiting hardware performance counters. The above two attacks can be easily mitigated by fixing the GPU vulnerability and limiting access to performance counters.

Side-channel attacks can be used for website fingerprinting. Memento [31] is a side-channel attack that tracks changes in the browser’s memory footprint. Loophole [73] exploits contention on shared event loops, which are FIFO queues in browsers that store and dispatch website events. Oren et al. [54] demonstrated a Prime+Probe cache side-channel attack implemented in JavaScript and used their attack to monitor website content and user activities. Shusterman et al. [65] proposed a cache-occupancy side-channel attack that can leak website identity with a high accuracy using low-resolution timers. Their follow-up work [64] demonstrated that the cache occupancy-attack can bypass multiple countermeasures deployed in modern browsers and can even be mounted from CSS when JavaScript is disabled. These attacks all exploit shared micro-architectural structures, which are commonly mitigated through resource partitioning [5, 14, 36, 45]. However, interrupt-based timing side-channel attacks, as shown in this paper, are much more difficult to mitigate. Our attack succeeds in large part due to the presence of resources that cannot be partitioned.

## 8 CONCLUSION

This paper highlighted the importance of thoroughly analyzing side-channel attacks by clarifying the primary source of information leakage behind a machine-learning-assisted attack.

Previous work by Shusterman et al. [64, 65] introduced a powerful website-fingerprinting attack that leverages timing side channels to achieve high accuracy. While effective, we found that this attack was not entirely understood. In this paper, we showed this attack does not only exploit cache contention, but in fact primarily exploits interrupt-based side channels. We constructed a loop-counting attack that makes no memory accesses and achieves even stronger performance. We additionally provided in-depth analysis to fully understand how our attack works under different isolation mechanisms and which interrupts play an important role in leaking information. Our analysis showed that it is virtually impossible to isolate all interrupts from a specific CPU core, since some interrupts are needed for the system to function properly. Finally, we proposed and evaluated two countermeasures that defend well against our new attack.

## ACKNOWLEDGMENTS

We thank our shepherd Yanjing Li and our anonymous reviewers for helpful feedback; Peter Deutsch for resolving issues with our machines so we could run our experiments; Sacha Servan-Schreiber for inspiring the title of our paper; and our friends, family, and labmates for their support. This research is partially supported by NSF Grant CNS-2046359, AFOSR Grant FA9550-20-1-0402, and the MIT-IBM Watson AI Lab.

## REFERENCES

- [1] Gunes Acar, Marc Juarez, et al. 2020. tor-browser-selenium - Tor Browser Automation With Selenium. <https://github.com/webfp/tor-browser-selenium>.
- [2] Mohammad Abdullah Al Faruque, Sujit Rokka Chhetri, Arquimedes Canedo, and Jiang Wan. 2016. Acoustic Side-Channel Attacks on Additive Manufacturing Systems. In *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCCPS)*. <https://doi.org/10.1109/ICCCPS.2016.7479068>
- [3] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida Garcia, and Nicola Taveri. 2019. Port Contention for Fun and Profit. In *2019 IEEE Symposium on Security and Privacy (SP)*. <https://doi.org/10.1109/SP.2019.00066>
- [4] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, Caroline Sporleder, et al. 2010. Acoustic Side-Channel Attacks on Printers. In *USENIX Security Symposium*.
- [5] Thomas Bourgeat, Ilia Lebedev, Andrew Wright, Sizhuo Zhang, and Srinivas Devadas. 2019. M16: Secure Enclaves in a Speculative Out-Of-Order Processor. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*.
- [6] Benjamin A Braun, Suman Jana, and Dan Boneh. 2015. Robust and Efficient Elimination of Cache and Timing Side Channels. (2015). Preprint, arXiv:1506.00189 [cs.CR].
- [7] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. 2017. Convolutional Neural Networks With Data Augmentation Against Jitter-Based Countermeasures. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer.
- [8] Xiang Cai, Rishab Nithyanand, and Rob Johnson. 2014. CS-BuFLO: A Congestion Sensitive Website Fingerprinting Defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*.
- [9] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A Systematic Approach to Developing and Evaluating Website Fingerprinting Defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*.
- [10] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching From a Distance: Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*.
- [11] Giovanni Cherubin, Jamie Hayes, and Marc Juárez. 2017. Website Fingerprinting Defenses at the Application Layer. *Proceedings on Privacy Enhancing Technologies* (2017).
- [12] Shane S Clark, Hossen Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. 2013. Current Events: Identifying Webpages by Tapping the Electrical Outlet. In *European Symposium on Research in Computer Security*. Springer.
- [13] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* (2016).
- [14] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium (USENIX Security 16)*.
- [15] Wenrui Diao, Xiangyu Liu, Zhou Li, and Kehuan Zhang. 2016. No Pardon for the Interruption: New Inference Attacks on Android Through Interrupt Timing Analysis. In *2016 IEEE Symposium on Security and Privacy (SP)*. <https://doi.org/10.1109/SP.2016.32> ISSN: 2375-1207.
- [16] Dmitry Evtvushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2016. Jump Over ASLR: Attacking Branch Predictors to Bypass ASLR. In *MICRO*.
- [17] Dmitry Evtvushkin, Ryan Riley, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2018. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. In *ASPLOS*.
- [18] Sina Faezi, Sujit Rokka Chhetri, Arnav Vaibhav Malawade, John Charles Chaput, William Grover, Philip Brisk, and Mohammad Abdullah Al Faruque. 2019. Oligo-Snoop: A Non-Invasive Side Channel Attack Against DNA Synthesis Machines. In *Network and Distributed Systems Security (NDSS) Symposium 2019*.
- [19] Daniel Genkin, Lev Pachmanov, Eran Tromer, and Yuval Yarom. 2018. Drive-by Key-Extraction Cache Attacks From Portable Code. In *International Conference on Applied Cryptography and Network Security*. Springer.
- [20] Xun Gong, Nikita Borisov, Negar Kiyavash, and Nabil Schear. 2012. Website Detection Using Remote Traffic Analysis. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer.
- [21] J.W. Gray. 1993. On Analyzing the Bus-Contention Channel Under Fuzzy Time. In *[1993] Proceedings Computer Security Foundations Workshop VI*. <https://doi.org/10.1109/CSFW.1993.246643>
- [22] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In *DIMVA*.
- [23] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. 2015. Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches. In *USENIX Security*.
- [24] Berk Gulmezoglu, Andreas Zankl, Thomas Eisenbarth, and Berk Sunar. 2017. PerfWeb: How to Violate Web Privacy With Hardware Performance Events. In *European Symposium on Research in Computer Security*. Springer.
- [25] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *25th USENIX Security Symposium (USENIX Security 16)*.
- [26] Wenjian He, Wei Zhang, Sanjeev Das, and Yang Liu. 2018. SGXlinger: A New Side-Channel Attack Vector Based on Interrupt Latency Against Enclave Execution. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*. <https://doi.org/10.1109/ICCD.2018.00025> ISSN: 2576-6996.
- [27] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies With the Multinomial Naïve-Bayes Classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*.
- [28] Andrew Hintz. 2002. Fingerprinting Websites Using Traffic Analysis. In *International workshop on privacy enhancing technologies*. Springer.
- [29] Gabriel Hospodar, Benedikt Gierlich, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. 2011. Machine Learning in Side-Channel Analysis: A First Study. *Journal of Cryptographic Engineering* (2011).
- [30] 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. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*.
- [31] Suman Jana and Vitaly Shmatikov. 2012. Memento: Learning Secrets From Process Footprints. In *2012 IEEE Symposium on Security and Privacy*. IEEE.
- [32] Rob Jansen, Marc Juarez, Rafa Galvez, Tariq Elahi, and Claudia Diaz. 2018. Inside Job: Applying Traffic Analysis to Measure Tor From Within. In *NDSS*.
- [33] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A Critical Evaluation of Website Fingerprinting Attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*.
- [34] The Linux kernel development community. 2013. BPF (Berkeley Packet Filter) Documentation. <https://www.kernel.org/doc/html/latest/bpf/index.html>. Accessed on 08.13.2021.
- [35] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. 2019. Make Some Noise. Unleashing the Power of Convolutional Neural Networks for Profiled Side-Channel Analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019).
- [36] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. 2018. DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE.
- [37] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *S&P*.
- [38] David Kohlbrenner and Hovav Shacham. 2016. Trusted Browsers for Uncertain Times. In *25th USENIX Security Symposium (USENIX Security 16)*.
- [39] 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.
- [40] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. 2015. A Machine Learning Approach Against a Masked AES. *Journal of Cryptographic Engineering* (2015).
- [41] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring Information Leakage in Website Fingerprinting Attacks and Defenses. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*.
- [42] Pavel Lifshits, Roni Forte, Yedid Hoshen, Matt Halpern, Manuel Philipose, Mohit Tiwari, and Mark Silberstein. 2018. Power to Peep-All: Inference Attacks by Malicious Batteries on Mobile Devices. *Proceedings on Privacy Enhancing Technologies* (2018).
- [43] Moritz Lipp, Daniel Gruss, Michael Schwarz, David Bidner, Clémentine Maurice, and Stefan Mangard. 2017. Practical Keystroke Timing Attacks in Sandboxed JavaScript. In *ESORICS*.
- [44] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. 2016. ARMageddon: Cache Attacks on Mobile Devices. In *USENIX Security*.
- [45] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B. Lee. 2016. CATalyst: Defeating Last-Level Cache Side Channel Attacks in Cloud Computing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. <https://doi.org/10.1109/HPCA.2016.7446082>
- [46] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-Level Cache Side-Channel Attacks are Practical. In *S&P*.
- [47] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. 2010. Website Fingerprinting and Identification Using Ordered Feature Sequences. In *European Symposium on Research in Computer Security*. Springer.
- [48] Haoyu Ma, Jianwen Tian, Debin Gao, and Chunfu Jia. 2020. Walls Have Ears: Eavesdropping User Behaviors via Graphics-Interrupt-Based Side Channel. In *Information Security*. Springer International Publishing, Cham. [https://doi.org/10.1007/978-3-030-62974-8\\_11](https://doi.org/10.1007/978-3-030-62974-8_11) Series Title: Lecture Notes in Computer Science.

- [49] Haoyu Ma, Jianwen Tian, Debin Gao, and Chunfu Jia. 2021. On the Effectiveness of Using Graphics Interrupt as a Side Channel for User Behavior Snooping. *IEEE Transactions on Dependable and Secure Computing* (2021).
- [50] Housseem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. 2016. Breaking Cryptographic Implementations Using Deep Learning Techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer.
- [51] Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. 2020. CopyCat: Controlled Instruction-Level Attacks on Enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 469–486. <https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-copycat>
- [52] 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*. ACM, Toronto Canada. <https://doi.org/10.1145/3243734.3243831>
- [53] 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*.
- [54] 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 *CCS*.
- [55] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache Attacks and Countermeasures: The Case of AES. In *CT-RSA*.
- [56] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale. In *NDSS*.
- [57] 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*.
- [58] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In *USENIX Security*.
- [59] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. 2018. On the Performance of Convolutional Neural Networks for Side-Channel Analysis. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer.
- [60] Vera Rimmer, Davy Preuvenciers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *Proceedings 2018 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA. <https://doi.org/10.14722/ndss.2018.23105>
- [61] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *CCS*.
- [62] Ronald L Rivest. 1991. *Cryptography and Machine Learning*. In *International Conference on the Theory and Application of Cryptology*. Springer.
- [63] Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, and Stefan Mangard. 2017. KeyDrown: Eliminating Keystroke Timing Side-Channel Attacks. *arXiv:1706.06381 [cs]* (June 2017). <http://arxiv.org/abs/1706.06381> arXiv: 1706.06381.
- [64] 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. *arXiv:2103.04952 [cs]* (March 2021). <http://arxiv.org/abs/2103.04952> arXiv: 2103.04952.
- [65] 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 *USENIX Security*.
- [66] Richard Sites. 2021. *Understanding Software Dynamics*. Addison Wesley.
- [67] Raphael Spreitzer, Veelasha Moonsamy, Thomas Korak, and Stefan Mangard. 2017. Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices. *IEEE Communications Surveys & Tutorials* (2017).
- [68] Xiaoxiao Tang, Yan Lin, Daoyuan Wu, and Debin Gao. 2018. Towards Dynamically Monitoring Android Applications on Non-Rooted Devices in the Wild. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*. ACM, Stockholm Sweden. <https://doi.org/10.1145/3212480.3212504>
- [69] Eran Tromer, Dag Arne Osvik, and Adi Shamir. 2010. Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology* (2010).
- [70] J.T. Trostle. 1998. Timing Attacks Against Trusted Path. In *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No.98CB36186)*. <https://doi.org/10.1109/SECPRI.1998.674829> ISSN: 1081-6011.
- [71] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2017. SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In *SysTEX*.
- [72] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Toronto Canada. <https://doi.org/10.1145/3243734.3243822>
- [73] Pepe Vila and Boris Köpf. 2017. Loophole: Timing Attacks on Shared Event Loops in Chrome. In *USENIX Security*.
- [74] 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)*.
- [75] 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*.
- [76] Tao Wang and Ian Goldberg. 2016. On Realistically Attacking Tor with Website Fingerprinting. *Proceedings on Privacy Enhancing Technologies* (2016).
- [77] Tao Wang and Ian Goldberg. 2017. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *26th USENIX Security Symposium (USENIX Security 17)*.
- [78] Yao Wang, Andrew Ferraiuolo, and G Edward Suh. 2014. Timing Channel Protection for a Shared Memory Controller. In *HPCA*.
- [79] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher Fletcher, Roy Campbell, and Josep Torrellas. 2019. Attack Directories, Not Caches: Side Channel Attacks in a Non-Inclusive World. In *S&P*.
- [80] Yuval Yarom and Katrina Falkner. 2014. Flush+Reload: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security*.
- [81] Yuval Yarom, Daniel Genkin, and Nadia Heninger. 2017. CacheBleed: A Timing Attack on OpenSSL Constant Time RSA. *JCEN* (2017).

## A CLOSED-WORLD WEBSITES DATASET

1688.com	6.cn	adobe.com
alibaba.com	aliexpress.com	alipay.com
amazon.com	aparat.com	apple.com
babytree.com	baidu.com	bbc.com
bing.com	booking.com	canva.com
chase.com	cnblogs.com	cnn.com
csdn.net	daum.net	detik.com
dropbox.com	ebay.com	espn.com
etsy.com	facebook.com	fandom.com
force.com	freepik.com	github.com
godaddy.com	gome.com.cn	google.com
grammarly.com	hao123.com	haosou.com
xinhuanet.com	huanqiu.com	ilovepdf.com
imdb.com	imgur.com	indeed.com
instagram.com	intuit.com	jd.com
kompas.com	linkedin.com	live.com
mail.ru	medium.com	microsoft.com
msn.com	myshopify.com	naver.com
netflix.com	nytimes.com	office.com
ok.ru	okezone.com	panda.tv
paypal.com	pikiran-rakyat.com	pinterest.com
primevideo.com	qq.com	rakuten.co.jp
reddit.com	rednet.cn	roblox.com
salesforce.com	savefrom.net	sina.com.cn
slack.com	so.com	sohu.com
spotify.com	stackoverflow.com	taobao.com
telegram.org	tianya.cn	tiktok.com
tmall.com	tradingview.com	tribunnews.com
tumblr.com	twitch.tv	twitter.com
vk.com	walmart.com	weibo.com
wetransfer.com	whatsapp.com	wikipedia.org
wordpress.com	yahoo.com	youtube.com
yy.com	zhanqi.tv	zillow.com
zoom.us		