THEME ARTICLE: TOP PICKS FROM THE 2022 COMPUTER ARCHITECTURE CONFERENCES

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

Jack Cook and Jules Drean, *Massachusetts Institute of Technology, Cambridge, MA, 02139, USA*
Jonathan Behrens, *Microsoft, Redmond, WA, 98052, USA*
Mengjia Yan, *Massachusetts Institute of Technology, Cambridge, MA, 02139, USA*

Machine learning has made it possible to mount powerful attacks through side channels that are otherwise challenging to exploit. However, due to the black-box nature of machine learning models, these attacks can be difficult to interpret correctly. Models that simply find correlations cannot be used to analyze the various sources of information leakage behind an attack. This article highlights the limitations of relying on machine learning for side-channel attacks without completing a comprehensive security analysis. We show that a state-of-the-art website-fingerprinting attack powered by machine learning was only partially analyzed. Its authors were misled into believing their attack exploited a cache-based side channel when it actually exploited an interrupt-based side channel. We demonstrate this through a comprehensive analysis, in which we run controlled experiments to rule out alternative hypotheses about the attack's primary source of leakage, and ultimately instrument the attack's code to prove our hypothesis.

Modern applications rely on underlying system hardware and software to remain isolated from one another. However, achieving this isolation is challenging due to the existence of *side channels*. In a side-channel attack, an adversary monitors "side effects" generated by a victim's execution patterns in order to decode a secret. These "side effects" can be observed by the adversary because of contention over resources it shares with the victim, such as caches, branch predictors, and DRAM.

Side-channel attacks are widely effective and have been used to leak cryptographic keys, website browsing activity,[1,2] and user behavior.[3,4] Some recent attacks have employed machine learning techniques, which can simplify development and improve robustness. For relatively complicated applications such as website fingerprinting, document fingerprinting, and acoustic side channels, the relationship between the observed "side effects" and the victim's 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.[1,2,5,6,7,8,9]

Nevertheless, correctly applying machine learning in these settings can be tricky. Machine learning models are good at finding correlations, which enables them to be used regardless of one's understanding of the side channel being exploited. This can lead to the development of powerful attacks that are poorly understood. Without this deeper understanding, the community is left unable to develop effective countermeasures or mechanisms to close the underlying side channels.

In this article, we explore the limitations of using machine-learning techniques in side-channel attacks. Specifically, we show that a recent machine-learning-powered side-channel attack proposed by Shusterman et al.,[1] known as a *cache-based sweep-counting attack*, was not properly analyzed, leading to an incorrect conclusion about the attack's primary cause. While this

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3273457
Date of publication 9 May 2023; date of current version 29 June 2023.

attack was believed to be powered by cache contention, we show that the attack's machine learning model primarily leverages information leaked through system interrupts.

A Mysterious Source of Leakage

At a glance, it may seem unclear why system interrupts would leak any information at all. Due to the complexity of machine-learning-powered side-channel attacks, it took us some time to reach this conclusion. We started with multiple hypotheses about the attack's main source of leakage and ran a series of controlled experiments to disprove them one by one. Once we reached our conclusion about system interrupts, we instrumented the attack's code to prove they were the main source of leakage. In this article, we turn this approach into a framework which can be used to analyze future machine-learning-powered side-channel attacks.

In each of our controlled experiments, we used a modified version of the cache-based sweep-counting attack's code that is stripped of memory accesses. We show that when memory accesses are removed, the attack's accuracy increases in nearly every case, effectively disproving the attack's reliance on the cache. We call this new attack a *loop-counting attack* and evaluate it extensively. Each experiment we run with this new attacker is designed to test one potential hypothesis about the primary source of leakage behind both attacks. This careful approach is necessary due to the opacity of machine-learning models, which discover correlations between "side effects" and secrets without yielding any insights into the signals being exploited.

A System-Interrupt Side Channel

This process led us to find that information is primarily leaked through system interrupts in both attacks. System interrupts are used by all modern computers to perform essential tasks asynchronously, such as sending and receiving data on a network, or interacting with devices such as a keyboard or a GPU. They are specific to each process, meaning interrupts related to one application should not be observed by another. With this in mind, how could they be used to mount a side-channel attack?

Intuitively, when an interrupt is triggered, it needs to be handled by an *interrupt handler*, a piece of kernel code that can run on the same core as the attacker's program. The attacker program will then be interrupted, and its instruction throughput should decrease. Figure 1 shows this process in action. In the white regions, the victim and attacker programs are both

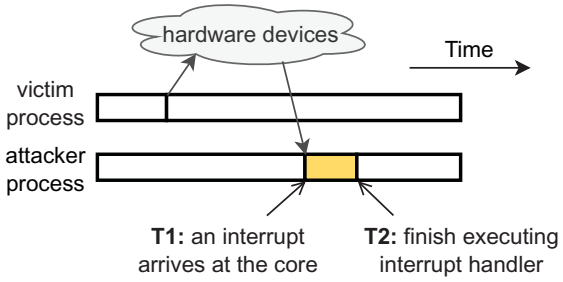

FIGURE 1. Example interrupt-based side channel.

able to execute freely on separate cores. However, when the victim accesses a shared resource, such as a network card or graphics card, it generates an interrupt that is processed on the attacker's core. In the yellow region, the attacker is preempted as the interrupt is processed, and the attacker's instruction throughput decreases. A particularly attentive attacker may record the time and notice a small jump, equal to $T2 - T1$, once the interrupt has been processed. Observing many of these jumps over time effectively allows the attacker to monitor the victim's execution patterns, and ultimately decode the victim's secrets.

To prove the connection between system interrupts and information leakage in both attacks, we finally instrument the Linux kernel. In our analysis, and for the first time in the literature, we find that *nonmovable interrupts*, such as softirqs and rescheduling interrupts, play a crucial role in leaking application information. This finding is important as nearly all prior research[3,10] studying interrupt-based side channels 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 nonmovable interrupts may require major system redesigns.

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 methods for analyzing side channels.

THE LOOP-COUNTING ATTACK

In this section, we build an attacker program that is almost identical to the sweep-counting attack[1] 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.

Attack Description

We show pseudo-code for a sweep-counting attacker in Figure 2(a) and our loop-counting attacker in Figure 2(b). In both algorithms, the attacker takes a 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 ms. 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 2(a)] 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 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.

```
int Trace[T*1000];
loop {
  count = 0;
  t_begin = time();

  do {
    // count iterations
    count++;
    // memory accesses
    for (i=0; i<N; i++) {
      tmp = buffer[i*64];
    }
  } while(time()–t_begin
      < P);

  Trace[t_begin] = count;
}
```

(a)

```
int Trace[T*1000];
loop {
  count = 0;
  t_begin = time();

  do {
    // count iterations
    count++;

    No memory
    accesses

  } while(time()–t_begin
      < P);

  Trace[t_begin] = count;
}
```

(b)

FIGURE 2. Pseudocode for both attacks. (a) Sweep-counting attack. (b) Loop-counting attack.

Trace Examples

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

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 ms.

We collect each trace by running the loop-counting attack code from Figure 2(b) for 15 s while the browser loads and renders a website in another tab. The shades in the trace correspond to counter values ranging from approximately 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 s, with spikes in activity around 5 and 10 s. Visual cues such as these

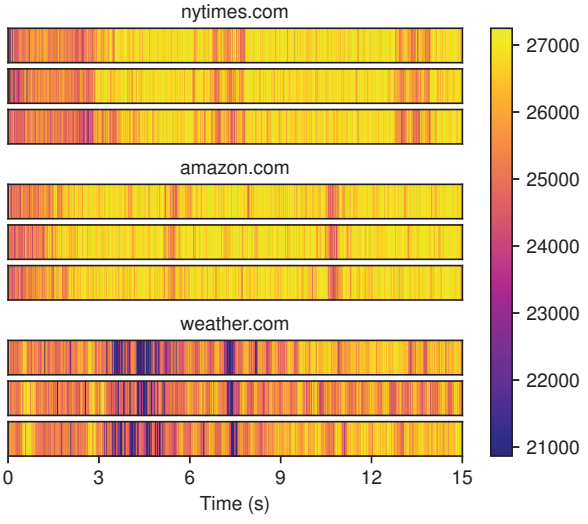

FIGURE 3. Example loop-counting traces collected over 15 s. Darker shades indicate smaller counter values and lower instruction throughput.

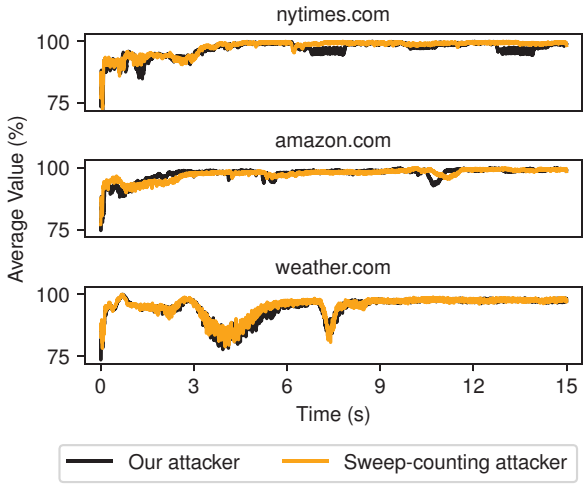

FIGURE 4. Normalized trace values averaged over 100 runs while loading three websites. Traces collected with both attackers are strongly correlated with each other.

indicate that loop-counting traces can be used as fingerprints to distinguish different websites.

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.

COMPARING THE TWO ATTACKS

In this section, we demonstrate that the sweep-counting attack by Shusterman et al.[1] 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[2] and find that our attack outperforms it in all but one experimental configuration.

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 presented by Shusterman et al.[2] 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.

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 websites that the victim will visit. In our experiments, the attacker aims to distinguish the victim's accesses of 100 different websites. By contrast, 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 "nonsensitive" websites. The attacker only knows the set of sensitive websites and aims to precisely identify the victim's access to these websites. When the victim visits a nonsensitive website, the attacker should simply report "nonsensitive."

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 evaluate our attack on four commercial web browsers: Chrome, Firefox, Safari, and Tor Browser.

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-s traces when attacking Chrome, Firefox, and Safari, and 50-s traces when attacking Tor Browser.

Closed-World Results

In the closed-world setup, our attack predicts which website was visited out of 100 possible websites. We directly compare the accuracy of our attack to the accuracy of the cache-based sweep-counting attack from Shusterman et al.[2] Note that website content has changed since Shusterman et al. published their work,[2]

TABLE 1. Classification accuracy obtained with JavaScript loop-counting attacker.

Browser	Operating system	Closed world		Open world	
		Loop-counting attack	Cache attack[2]†	Loop-counting attack	Cache attack[2]†
Chrome 92	Linux	**96.6**±0.8	91.4±1.2	**97.2**±0.3	86.4±0.3
Firefox 91	Linux	**95.3**±0.7	80.0±0.6	**96.4**±0.8	87.4±1.2
Safari 14	macOS	**96.6**±0.5	72.6±1.3	**96.7**±0.6	80.5±1.0
Tor Browser 10	Linux	**49.8**±4.2	46.7±4.1	62.9±2.4	62.9±3.3

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

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, our attack is stronger for both experimental setups. For example, when attacking Chrome running in Windows in a closed-world setup, our attack achieves an accuracy of 92.5%, while the cache-based attack only achieves a success rate of 80.0%. Notably, 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-five accuracy, the rate at which the correct website is one of the model's top-five predictions, is 86.4%.

Open-World Results

In our open-world setup, we add 5000 "nonsensitive" 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 "nonsensitive" class with all 5000 open-world traces. This experimental design is identical to the design used by Shusterman et al.[2]

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 Shusterman et al.[2] 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.

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 the next section.

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

THE PRIMARY SOURCE OF LEAKAGE

At this point, we can rule out caches and the rest of the memory hierarchy as the primary source of leakage in these attacks. Now, we can investigate other hypotheses to determine the true primary source. For each potential side channel, we design a controlled experiment where we add isolation to close the side channel and observe changes in the attack's accuracy. Figure 5 gives an example of the process used to test and disprove these different hypotheses.

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 2(b) in Python.

Table 2 shows classification accuracy when evaluating the Python attack code on Chrome 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 interrupt request (IRQ) interrupts, and running the attacker process and the victim process in two separate virtual machines (VMs).

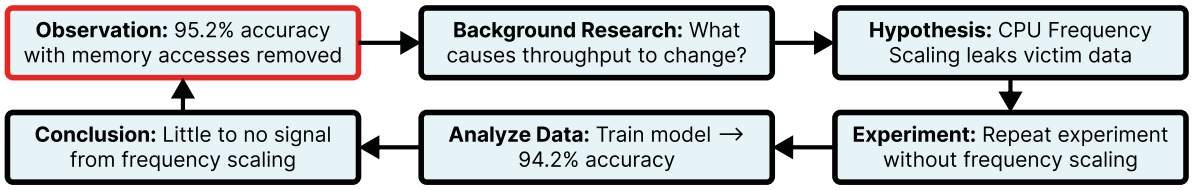

FIGURE 5. Illustration of our experimental procedure. We start at the top-left and repeatedly run new controlled experiments to learn more about our loop-counting attack.

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 by running an experiment with this feature disabled. We observe a small decrease of 1% in the top-one accuracy, which indicates that our attack is still highly effective even without any signal due to frequency scaling.

Run on 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. 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

There exist many types of system interrupts. For *movable* interrupts, such as graphics interrupts, network interrupts, and SATA interrupts, there is no restriction on where these interrupts should be processed. However, *nonmovable interrupts*, such as timer interrupts, softirqs, rescheduling interrupts, and translation lookaside buffer (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-one and top-five 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 nonmovable interrupts might play an important role in leaking the victim's activity.

TABLE 2. Classification accuracy obtained with Python loop-counting attacker under various isolation mechanisms.

Isolation mechanism	Top-one accuracy	Top-five 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%

Run in Separate VMs

Finally, we evaluate how our interrupt-based side channel performs when the attacker and victim are running in separate VMs, in addition to all isolation mechanisms from our previous configurations. In this configuration, we observe a slight increase of 3.4% in the top-one accuracy. This observation contradicts our expectation, as one would expect the stronger isolation offered by VMs would reduce the effectiveness of our attack. One 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 2: Current isolation mechanisms are insufficient to completely mitigate the loop-counting attack, which monitors instruction throughput for website fingerprinting.

With these results, nonmovable interrupts appear to be the primary source of leakage in both attacks. Because nonmovable interrupts cannot be isolated nor deactivated, it is impossible to mount a simple experiment and try to disprove this hypothesis. Instead, we

need to use a different method to prove that nonmovable interrupts are responsible for the information leakage in these attacks.

IDENTIFYING THE UNDERLYING SIDE CHANNEL

In this section, we use the instrumentation functionalities provided by Linux to analyze the loop-counting attack in more detail and uncover the underlying side channels. In particular, Linux allows setting kprobes and tracepoints at specific points in the kernel binary. When execution reaches these points, Linux calls 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 application programming interface of eBPF byte code.

Analysis Tool

We use eBPF to monitor system interrupts as they are processed on each core, logging the timestamp and root cause of each one. We then compare the timing of each interrupt to the timings of "jumps," or large and sudden increases in the local time, that can be observed by our attacker. Our attacker watches for these jumps 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 virtual dynamic shared object (`vDSO`). 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.

Analysis Results

Figure 6 reports the average time spent in interrupt handlers during 100 runs from three popular websites, sample traces for which were presented in Figure 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 nonmovable interrupts. Notice that the amount of time spent handling interrupts closely matches the appearance of the traces in Figure 3. For instance, most interrupt-handler activity when loading nytimes.com happens in the first 4 s, while for amazon.com, we observe spikes at 5 and 10 s.

Different websites can even trigger different types of nonmovable 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.

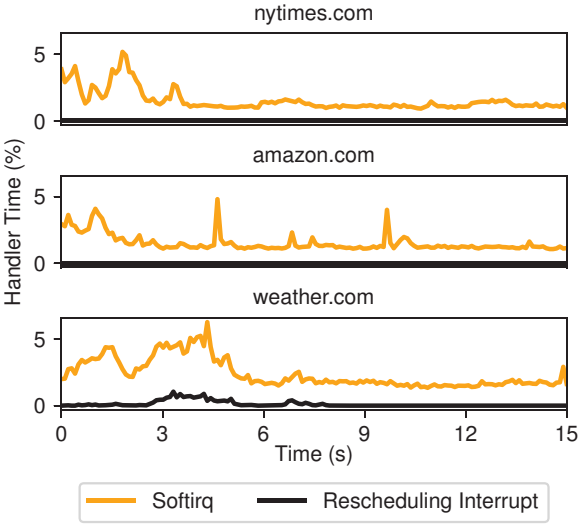

FIGURE 6. Percentage of time spent processing interrupts averaged over 100 runs while loading three different websites.

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 3:** 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 nonmovable 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 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 nonmovable interrupts and other system events requires instrumenting the kernel at a further depth than allowed by eBPF. Since it is infeasible to isolate nonmovable 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 4: Nonmovable interrupts, such as softirqs and rescheduling interrupts, leak information from victim processes. Blocking information leakage from these interrupts may require major system redesigns.

CONCLUSION

This paper highlights the importance of thoroughly analyzing machine-learning-powered side-channel attacks. We presented a framework which can be used to analyze future side-channel attacks in depth.

We first constructed our "loop-counting" attack by removing memory accesses from a similar cache-based attack. We showed our attack achieves higher accuracy, ruling out cache-based side channels as a primary source of leakage. We then performed several controlled experiments in which we closed potential side channels, such as frequency scaling and CPU contention, and observed that our attack's accuracy was not significantly impacted. Finally, we instrumented the Linux kernel to prove that both attacks exploit interrupt-based side channels. Our analysis showed that it is virtually impossible to isolate all interrupts from a specific CPU core, since many nonmovable interrupts are needed for the system to function properly.

We hope this work will inspire the research community to perform more in-depth analysis of side-channel attacks as these are essentials to understand the security of our system and to build efficient defense mechanisms.

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 lab mates for their support. This work was supported in part by NSF Grant CNS-2046359, AFOSR Grant FA9550-20-1-0402, and the MIT-IBM Watson AI Lab.

REFERENCES

1. A. Shusterman, A. Agarwal, S. O'Connell, D. Genkin, Y. Oren, and Y. Yarom, "Prime+Probe 1, JavaScript 0: Overcoming browser-based side-channel defenses," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 2863–2880.
2. A. Shusterman et al., "Robust website fingerprinting through the cache occupancy channel," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 639–656, doi: 10.1109/TDSC.2020.2988369.
3. M. Lipp, D. Gruss, M. Schwarz, D. Bidner, C. Maurice, and S. Mangard, "Practical keystroke timing attacks in sandboxed JavaScript," in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*, 2017, pp. 191–209, doi: 10.1007/978-3-319-66399-9_11.
4. Y. Oren, V. Kemerlis, S. Sethumadhavan, and A. Keromytis, "The spy in the sandbox: Practical cache attacks in JavaScript and their implications," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1406–1418, doi: 10.1145/2810103.2813708.
5. G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: A first study," *J. Cryptogr. Eng.*, vol. 1, no. 4, pp. 293–302, Dec. 2011, doi: 10.1007/s13389-011-0023-x.
6. L. Lerman, G. Bontempi, and O. Markowitch, "A machine learning approach against a masked AES: Reaching the limit of side-channel attacks with a learning model," *J. Cryptogr. Eng.*, vol. 5, no. 2, pp. 123–139, Jun. 2015, doi: 10.1007/s13389-014-0089-3.
7. P. Lifshits et al., "Power to peep-all: Inference attacks by malicious batteries on mobile devices," in *Proc. Privacy Enhancing Technol.*, 2018, pp. 141–158, doi: 10.1515/popets-2018-0036.
8. H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Proc. 6th Int. Conf. Secur., Privacy, Appl. Cryptogr. Eng.*, 2016, pp. 3–26, doi: 10.1007/978-3-319-49445-6_1.

9. H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: GPU side channel attacks are practical," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 2139–2153, doi: 10.1145/3243734.3243831.
10. X. Tang, Y. Lin, D. Wu, and D. Gao, "Towards dynamically monitoring android applications on non-rooted devices in the wild," in *Proc. 11th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2018, pp. 212–223, doi: 10.1145/3212480.3212504.

JACK COOK is with the Massachusetts Institute of Technology, Cambridge, MA, 02139, USA. His research interests include machine learning interpretability, natural language processing, and human–computer interaction. Contact him at cookj@alum.mit.edu.

JULES DREAN is a Ph.D. student at the Massachusetts Institute of Technology, Cambridge, MA, 02139, USA. His research interests include side-channel attacks and trusted execution environments. Contact him at drean@mit.edu.

JONATHAN BEHRENS is a senior software engineer at Microsoft, Redmond, WA, 98052, USA. His research interests include operating systems and distributed systems. Behrens received his Ph.D. degree from the Massachusetts Institute of Technology. Contact him at fintelia@gmail.com.

MENGJIA YAN is an Assistant Professor in the Electrical Engineering and Computer Science Department, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA. Her research interests include computer architecture and hardware security, with a focus on side-channel attacks and defenses. Yan received her Ph.D. degree from the University of Illinois at Urbana-Champaign. Contact her at mengjia@csail.mit.edu.