Cache Occupancy Channel Website Fingerprinting with the aid of Device Fingerprinting

Pietro Costanzi Fantini, *1982805, Sapienza University of Rome*

Abstract—Cache-based Website Fingerprinting (WF) is a powerful side-channel attack that bypasses traditional network-level defences. However, its effectiveness is limited by its high dependency on the target's specific hardware and software environment. This paper demonstrates that this limitation can be overcome by creating an adaptive attacker that first performs passive device fingerprinting to select a pre-trained model tailored to the victim's system. Through a study across multiple CPUs, browsers, and operating systems, we show that while a naive, generic model achieves a low baseline accuracy of at most 49.5%, our proposed adaptive attacker achieves accuracies often exceeding 70% and reaching as high as 96% in some configurations. This work proves that environmental parameters are a key component of a practical cache-based WF attack and highlights the need to consider these adaptive threats when designing future browser and hardware defences.

Index Terms—Website Fingerprinting, Side-Channel Attacks, Cache Occupancy, Device Fingerprinting, Machine Learning.

1 Introduction

In recent years, protecting user data privacy has become a significant challenge. One of the more subtle battlegrounds for this is the modern web browser, where a malicious user can take advantage of vulnerabilities present in features like installed browser extensions, interactions on the website and predictable network traffic to infer certain user characteristics and behaviours. This information can then be used for malicious intent or can be potentially sold to advertisers in order to send more tailored advertisements to users. This has been an issue in the past because of cookie-based tracking on websites, however due to privacy concerns the tracking now occurs in a more subtle and covert way.

A prominent example of this is **Website Fingerprinting** [9] [12], a side-channel attack where an adversary identifies the websites a user visits, even over an encrypted connection. The classic approach attackers use to deploy this kind of attack is by analysing network traffic patterns of users connecting to websites. The idea behind this attack is to collect network traces and determine which website the client visited and this is possible since these might have some distinctive features that make the digital fingerprint of that website recognizable. However, these methods are increasingly challenged by countermeasures like **traffic morphing** and **traffic splitting** [8] [20].

To overcome these defenses, Shusterman et al. [4] proposed a more resilient attack that shifts the focus from the network to the local machine. Their method uses a **cache occupancy side-channel**, where an attacker-controlled website monitors the state of the victim computer's cache, and uses that information to infer the victim's web activity in other tabs of the same browser or even in other browsers. This technique is resistant to network-level defences and can be applicable in scenarios where network-based fingerprinting is known to be less effective, such as when the browser itself caches the contents of the website.

While powerful, the work by Shusterman et al. left a critical question largely unexplored: how much does the attack's effectiveness depend on the target's specific hardware and software? In this paper we prove that by first deploying a passive device fingerprinting attack [15], an attacker can launch a much more successful and precise website fingerprinting attack. We hypothesize that using this device information to select a dedicated pre-trained model will yield significantly higher accuracy than a generic approach that is trained with a collection of memory traces.

To validate this, we use the JavaScript *memorygrammer* from [5] to conduct a closed-world study on five websites, collecting 200 traces each on modern versions of Google Chrome and Firefox. We analyse how the operating system itself may impact the fingerprints by running experiments on both Windows and Linux systems. We also investigate the role of browser-specific timer resolutions, a countermeasure developed in response to attacks like Spectre [13], and show how an attacker must adapt their measurement strategy to the target environment. Ultimately, we propose an adaptive attack model where a initial device fingerprint is used to select an optimal classification model, demonstrating a noticeable increase in attack accuracy. These findings have clear implications for the practical security of modern web browsers and future research into side-channel attacks.

Manuscript delivered on June 28 2025

2 Background

2.1 Website Fingerprinting Attacks

The classic Website Fingerprinting (WF) attack model assumes an adversary positioned on the network path between a client and a server. Much previous work has demonstrated the ability of such an adversary to make inferences about a user's browsing activity by performing statistical analysis on encrypted network traffic [7]. By studying observable metadata such as the size, direction, and timing of data packets, an attacker can construct a recognizable signature, or "fingerprint," for a specific website. The effectiveness of these attacks has led to the development of various countermeasures. The common strategy behind these defences is to obfuscate traffic features by injecting random delays or spurious "cover" traffic, making it more difficult to distinguish one website's traffic pattern from another [18].

Previous work focused on collecting a high number of traces for each website, focusing on building feature-based machine learning models, however, recent work has shifted towards more advanced classification techniques to handle the complexity of modern web traffic. Bhat et al. [16] proposed using a ResNet-based neural network for high-accuracy classification with less data. In a different approach Sirinam et al. [14] utilized N-Shot learning, a method that compares pairs of traces to classify a new trace based on its similarity to known, labelled examples.

This paper will use a **CNN** (Convolutional Neural Networks) model. In particular CNNs are excellent at finding local features or shapes within data. For the *memorygrams* of [4], which are visualizations of cache activity, it learns to spot the specific spikes, dips, and bursts of activity that are characteristic of a particular website. This model are able to achieve a high degree of accuracy both in a closed-world dataset and an open-world dataset of data. The difference between a **closed-world** dataset and an **open-world** dataset, however, due to limited computing power, our work will focus on a closed-world setting.

2.2 Cache-Based Side-Channel Attacks

In computing systems, multiple programs running on the same processor will share hardware components. This resource sharing, particularly within the micro-architecture, can create covert information leakage pathways known as side channels. Such channels can be exploited by malicious programs to learn secret data from other processes, such as cryptographic keys, user keystrokes, and the memory address layout of a victim program [6]. Among the most well-studied of these are cache-based side-channel attacks, which leverage contention within the processor's cache memory.

Processor caches are designed to mitigate the latency difference between the high-speed CPU and the slow main memory (RAM). A cache is a small, fast memory buffer that stores copies of recently used data from main memory. The majority of modern processors utilize a *set-associative* cache design, in particular Intel. In this architecture, the cache is partitioned into numerous *sets*. Each address in main memory is mapped to one specific set, meaning its data can only be stored within that designated partition of the cache.

When the processor requests data from a memory address, it first checks the cache hierarchy. If the data is present (a cache hit), it is retrieved quickly. If the data is not found (a cache miss), the search continues to the next level of the cache. A miss in the final cache level, known as the **Last Level Cache (LLC)**, necessitates a much slower access to the system's RAM to fetch the data.

The method used in this paper, the **Cache Occupancy Channel**, directly exploits this design. An attacker's script allocates a large buffer in memory (ideally the size of the Last-Level Cache, or LLC) and repeatedly measures the time it takes to access it. When a victim process accesses memory, it may evict the attacker's data from the cache. This forces the attacker's script to retrieve its data from main memory, introducing measurable delays. The time taken to access the buffer is therefore roughly proportional to the number of cache lines the victim has used [3].

2.3 Browser Mitigations and Device Fingerprinting

The discovery of critical vulnerabilities like Spectre [13], a side-channel attack that exploits "*speculative execution*", forced browser vendors to implement widespread mitigations. A primary defence has been the reduction of timer precision. As modern browsers limit the resolution of high-precision timers like *performance.now()*, the precise timing differences needed for many cache attacks become harder to measure. For example, modern versions of Chrome now restrict timer resolution to 100 microseconds, while Firefox has also implemented similar countermeasures [19] [1].

This paper leverages **Device Fingerprinting** [15], a technique that collects information about a device's hardware and software configuration via JavaScript APIs. By knowing which browser and OS the victim is using, an attacker can infer the timer resolution and other system properties, allowing them to adapt their attack, for example, by switching from measuring *probe time* to counting *probe frequency*, to achieve a much higher success rate.

3 Overview of your proposed approach

This work adopts the cross-tab attack model proposed by Shusterman et al. [4], in which an adversary uses phishing techniques or malicious advertising to lure a victim into opening an attacker-controlled website. This malicious page contains a JavaScript-based monitor that uses the cache occupancy channel to collect *memorygrams* while the victim browses legitimate websites in other tabs or browser instances.

While the baseline of the attack is set, our primary contribution is the introduction of a more sophisticated, adaptive attack model. We hypothesize that the effectiveness of a cache-based fingerprinting attack can be increased if the attacker first performs passive device fingerprinting to profile the target's environment. This initial profiling allows the attacker to tune the parameters of the attack to the specific hardware and software configuration of the victim, leading to a more precise and successful classification.

Our proposed adaptive attack consists of two distinct phases:

- **Phase 1 - Passive Device Fingerprinting**: Once the victim visits the malicious site, a lightweight JavaScript script collects key information about the target system. This includes the browser type and version, operating system, and CPU architecture (e.g., Intel or AMD), which can often be inferred from the userAgent string and WebGL rendering information.
- **Phase 2 - Tuned Fingerprinting and Classification**: The information gathered in the first phase is then used to configure the JavaScript *memorygrammer*. As our experiments show, cache management systems and browser timer resolutions differ across environments. By knowing the victim's system architecture and browser, the attacker can select the optimal measurement strategy and then use a adapted, pre-trained classification model that was built using data from a matching environment.

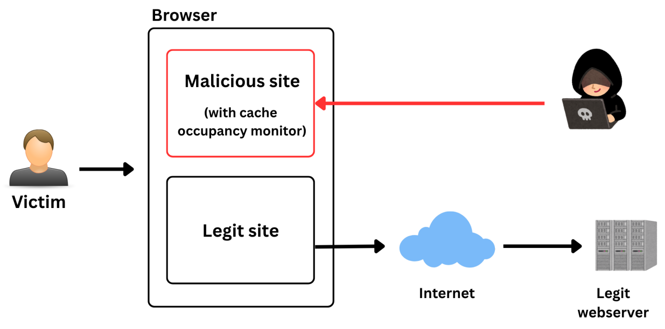

Fig. 1. Attack model

4 Evaluation

To test the hypothesis that an initial device fingerprinting makes the attack more effective, we conducted a series of experiments across different micro-architectures, browsers, and operating systems. This section details our experimental setup, the machine learning methodology, and the results of our analysis.

4.1 Experimental Setup

A dataset was collected from five target websites using two machine configurations: a high-performance laptop with an Intel® Core™ i7-1260P processor with an 18MB LLC, and an older desktop computer with an Intel® Core™ i7-7700K processor and an 8MB LLC. On each machine, we collected 200 cache traces per website using both Google Chrome (v. 137) and Firefox (v. 139). This process was performed on both Windows and Linux operating systems to create a diverse set of environmental conditions.

The *memorygrams* were collected over a 30-second period with a sampling period of 2 ms, resulting in traces of 15,000 samples each. It is important to note that the timer resolution of modern browsers is coarsened as a security measure [19] [1].

4.2 Classification Methodology

To perform website fingerprinting on cache occupancy traces, a deep learning pipeline was used using Python with the **TensorFlow** and **Keras** libraries. It consists of four stages:

- **Data Loading**: The script first loads all the collected **.json** files, where each file contains a single *memorygram* trace and the name of the website it belongs to.
- **Data Preprocessing**: Each trace is normalized so that the model focuses on the shape of the cache activity. Website names are also converted into a numerical format that the model can understand (classes).
- **Model Design**: We use a **Convolutional Neural Network (CNN)**, a type of deep learning model that is excellent at finding patterns in sequence data. The model is designed to automatically extract the most important features from the *memorygrams*.
- **Training and Evaluation** The dataset is split, with 80% used to train the model and 20% for testing. Finally, the trained model's performance is measured on the test data to get a realistic measure of its accuracy.

4.3 Attack Scenarios

The core of our evaluation is a comparison between two attacker models:

- **Naive Attacker**: This model represents an adversary who collects cache traces from various sources but does not perform any preliminary device fingerprinting. To simulate this, we trained a "generic" CNN model on a dataset containing a mix of traces from all our tested environments.
- **Adaptive Attacker**: This model represents our proposed approach. Here, the adversary first runs a device fingerprinting script to identify key system properties like the browser, OS, and CPU vendor. Based on this profile, the attacker then selects an adapted, pre-trained model that was built exclusively on data from a matching environment.

TABLE 1
Comparison of Classification Performance for Naive vs. Adaptive Attack Models Across Different Environments

OS	**CPU**	**LLC**	**Browser**	**Naive Attacker**			**Adaptive Attacker**		
				Accuracy	Recall	F1-Score	Accuracy	Recall	F1-Score
Windows	i7-1260P	18MB	Chrome v137	48.0%	48.0%	47.0%	73.5%	73.0%	74.0%
Windows	i7-1260P	18MB	Firefox v139	49.5%	49.0%	50.0%	72.3%	73.0%	73.0%
Windows	i7-7700K	8MB	Chrome v137	23.0%	20.0%	9.0%	76.5%	77.0%	76.0%
Windows	i7-7700K	8MB	Firefox v139	33.6%	34.0%	32.0%	77.2%	77.0%	77.0%
Linux	i7-1260P	18MB	Chrome v137	22.3%	22.0%	21.0%	70.5%	70.0%	71.0%
Linux	i7-1260P	18MB	Firefox v139	25.1%	25.0%	23.0%	51.3%	51.0%	51.0%
Linux	i7-7700K	8MB	Chrome v137	34.6%	35.0%	23.0%	96.4%	96.0%	96.0%
Linux	i7-7700K	8MB	Firefox v139	20.0%	20.0%	7.0%	89.5%	90.0%	90.0%

The key idea behind this approach is that the naive attacker will simply pass the data he collected to the CNN and get a **baseline accuracy** from the model. The adapted adversary on the other hand will have performed a device fingerprinting analysis beforehand and will know how to perform an initial classification of the cache traces. By using this data on the model he will be able to get a **improved accuracy**, which will be greater than the baseline accuracy.

The malicious JavaScript code that is set up to perform the device fingerprinting is able to extract key information of the victim's system like:

- **userAgent**: This is a key piece of information since it explicitly tells the attacker what browser the victim has and use a model trained on that specific one used by the victim.
- **OS**: Another key piece of information that can help the attacker narrow down the model to use in order to get a high accuracy.
- **cpuCores**: This is useful information when setting up the JavaScript *memographer* since a high cpu core count might indicate a higher-end CPU with a larger LLC, thus having to tweak some parameters within the JavaScript *memographer* in order to get a more precise cache trace.
- **CPU vendor**: Another key piece of information since this might indicate Intel or AMD and because CPU vendors have different cache management systems, the attacker might need to create a model ad-hoc for different micro architectures.

4.4 Results and Discussion

The results of our experiments are summarized in Table 1. As shown, the **adaptive attacker**, who leverages device fingerprinting to select a adapted model, achieves an improvement in accuracy, recall, and F1-score compared to the **naive attacker**.

The *memorygrams* themselves reveal why this is the case. While some work suggests static websites are harder to fingerprint, the more critical factor is the stability of a site's overall structure, as noted by Juárez et al. [10]. Our visualizations show that even for dynamic sites like *BBC.com* and *Amazon.com*, the initial page loading process creates a consistent and recognizable pattern of cache activity. In the *memorygrams* in particular, the lighter parts indicate moments of HIGH cache activity, where the browser's rendering process causes a high number of cache evictions. Conversely, darker parts indicate periods of LOW cache activity. The unique sequence of these periods forms the website's fingerprint.

From the *memorygrams* we can definitely notice a trend in regards to the cache activity. On the faster system with the Intel i7-1260P the memory accesses are faster and aren't being picked up as much as the slower system with the Intel i7-7700K, that has more distinguishable and defined *memorygrams*. The lowest precision the model was able to achieve was when loading websites on Firefox in a Linux environment on the faster system with the i7-1260P. By looking at the specific *memorygram* of this particular test (Figure 6), this data makes sense since all the memory traces are not particularly distinctive and the model had issues when trying to identify them.

In contrast, the highest precision was recorded when loading websites on Chrome in a Linux environment on the slower system with the i7-7700K. By looking at the specific *memorygram* of this particular test (Figure 7), this data makes sense since all the memory traces very distinctive and the model was precise when trying to identify them.

5 Related work

The field of website fingerprinting has been extensively studied, and the main focus of previous work was to analyse the network traces of target systems in order to classify them and de-anonymize user activity [7]. The effectiveness of this approach can be limited by various network-level countermeasures designed to obfuscate these patterns, often by injecting cover traffic or padding packets [18]. Moreover, factors like natural network noise and "*concept drift*", where websites change over time, can worsen the accuracy of these network-based attacks [10].

To address these issues, Shusterman et al. [4] proposed a new form of attack that moves from the network to the local machine, monitoring the CPU's cache occupancy as a side channel. This method has proven more resilient to both traffic-shaping defences and the effects of browser caching. Our work builds directly upon this foundation. To our knowledge, no prior scientific paper has combined a **cache occupancy attack** with a preliminary **device fingerprinting**

analysis to create an adaptive attack model that selects a specialized classifier.

The primary defences against cache-based attacks have focused on mitigating contention-based techniques. One major area of research is cache randomization. For example, Werner et al. developed ScatterCache [11], a design that pseudo-randomly remaps cache lines to hide them from an attacker. While effective against contention, a systematic evaluation by Chakraborty et al. [5] demonstrated that many state-of-the-art randomized cache designs, including ScatterCache and CEASER, remain vulnerable to occupancy-based attacks, which they proved by performing a full AES key recovery.

Other defence strategies have also been proposed. **Cache partitioning**, often supported by hardware features like *Intel's Cache Allocation Technology* (CAT) [17], aims to isolate processes from one another to prevent cross-core information leakage. Perhaps the most critical defence relevant to our browser-based attack is the reduction of timer precision. In response to the discovery of speculative execution attacks like Spectre [13], modern browsers now deliberately coarsen the resolution of high-precision timers such as *performance.now()* [2], making it harder for malicious scripts to measure the minute timing differences required for many side-channel attacks.

6 Future Works

While our study confirmed that an adaptive attacker is significantly more effective, there are several interesting possibilities for future research that could use these findings.

An obvious next step would be to expand the architectural testing. Our work was done on Intel CPUs, but as AMD and ARM-based processors become more prevalent, particularly in mobile devices, it is important to understand how this attack performs on their particular cache designs. The differences in cache design and physical layout on these architectures would likely produce very different fingerprints, presenting new challenges and opportunities for the attacker.

Another area for exploration is the model architecture itself. We used a CNN-based model, but other architectures might be better suited for this type of sequence data. Investigating models like Transformers, which are designed to capture long-range dependencies, could potentially bring to even more accurate classifiers.

Finally, a key limitation in any fingerprinting study is the size of the dataset. While our 200 traces per site were sufficient to prove our hypothesis, a much larger-scale data collection, with thousands of traces, would allow the models to learn more robust and generalizable features. It's likely that with more data, even the lower-performing models in our study could see a significant boost in accuracy.

7 Conclusions

The results from our experiments show a contrast between a naive attacker and our proposed adaptive attacker. The **naive model**, which was trained on a mixed dataset without any prior knowledge of the target environment, struggled significantly, achieving a baseline accuracy of at most **49.5%**. This confirms that environmental factors like the operating system, CPU architecture, and browser choice create fingerprints that are too varied for a single generic model to learn effectively.

In contrast, the **adaptive attacker**, which first identifies the target's environment and then selects an adapted model, demonstrated a noticeable increase in performance. We saw accuracy rates as high as **96.4%** accuracy in certain configurations. These results validate our hypothesis: knowing the specific characteristics of a target system is a crucial prerequisite for launching a successful cache-based fingerprinting attack.

Interestingly, the performance of the adaptive attack still varied across different environments, with the highest success rate observed on the Linux system with the older i7-7700K processor. This suggests that our adaptive model is a clear improvement, however, the system's parameters like its OS, the browser used, and the hardware produces distinct and sometimes more challenging fingerprints to detect.

Our work demonstrates that the threat of cache-based website fingerprinting is not just theoretical but can be made practical and relevant. By adding a preliminary device fingerprinting stage, an attacker can overcome many issues that would otherwise worsen a simpler attack. This highlights the need for better defence mechanisms that account for not just network traffic, but also the information leaked by the very hardware our browsers run on. Future work could expand on this by testing a wider range of architectures, such as ARM, and by developing countermeasures that directly target this adaptive attack model.

References

[1] "Performance: now() method." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Performance/now

[2] L. W. J. 3, "Aligning timers with cross origin isolation restrictions," 2018. [Online]. Available: https://blog.mozilla.org/security/2018/01/03/mitigations-landing-new-class-timing-attack/

[3] B. P. Alex H. Williams and N. Maheswaranathan, "Discovering precise temporal patterns in large-scale neural recordings through robust and interpretable time warping," in *Neuron*, 2020, pp. 246–259, Volume 105, Issue 2. [Online]. Available: https://www.cell.com/neuron/fulltext/S0896-6273(19)30894-3?dgcid=raven_jbs_etoc_email

[4] Z. A. Anatoly Shusterman and E. Croitoru, "Website fingerprinting through the cache occupancy channel and its real world practicality," in *IEEE Transactions on Dependable and Secure Computing*, 2020, pp. 2042–2060. [Online]. Available: 10.1109/TDSC.2020.2988369

[5] N. M. Anirban Chakraborty and S. Saha, "Systematic evaluation of randomized cache designs against cache occupancy," in *Proceedings of the 28th USENIX Security Symposium.*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2310.05172

[6] A. S. Dag Arne Osvik and E. Tromer, “Cache attacks and countermeasures: The case of aes,” in *Lecture Notes in Computer Science*, 2006, pp. 1–20. [Online]. Available: https://link.springer.com/chapter/10.1007/11605805_1
[7] R. W. Dominik Herrmann and H. Federrath, “Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier,” in *Proceedings of the 2009 ACM workshop on Cloud computing security*, 2009, pp. 31–42. [Online]. Available: https://doi.org/10.1145/3399742
[8] T. K. Eric Chan-Tin and J. Kim, “Website fingerprinting attack mitigation using traffic morphing,” in *IEEE 38th International Conference on Distributed Computing Systems*, 2018, pp. 1575–1577. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes
[9] J. Hayes and G. Danezis, “k-fingerprinting: A Robust Scalable Website Fingerprinting Technique,” in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1187–1189. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes
[10] S. A. Marc Juarez and G. Acar, “A critical evaluation of website fingerprinting attacks,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 263–274. [Online]. Available: https://doi.org/10.1145/2660267.2660368
[11] T. U. Mario Werner and L. Giner, “Scattercache: Thwarting cache attacks via cache set randomization,” in *Proceedings of the 28th USENIX Security Symposium.*, 2019, pp. 675–679. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/werner
[12] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, “Website fingerprinting at internet scale,” 2016. [Online]. Available: https://nymity.ch/tor-dns/pdf/Panchenko2016a.pdf
[13] J. H. Paul Kocher and A. Fogh, “Spectre attacks: exploiting speculative execution,” in *Communications of the ACM*, 2020, pp. 93–101, Volume: 63, Issue: 7. [Online]. Available: https://doi.org/10.1145/3399742
[14] N. M. Payap Sirinam and M. S. Rahman, “Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1131–1148. [Online]. Available: https://doi.org/10.1145/3319535.3354217
[15] W. S. Qiang Xu, Rong Zheng and Z. Han, “Device fingerprinting in wireless networks: Challenges and opportunities,” in *IEEE Communications Surveys & Tutorials*, 2018, pp. 94–104, Volume: 18, Issue: 1. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7239531
[16] D. L. Sanjit Bhat and A. Kwon, “Var-cnn: A data-efficient website fingerprinting attack based on deep learning,” in *Proceedings on Privacy Enhancing Technologies*, 2019, pp. 292–300. [Online]. Available: https://doi.org/10.2478/popets-2019-0070
[17] J. S. Vicent Selfa and L. Eeckhout, “Application clustering policies to address system fairness with intel’s cache allocation technology,” in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017. [Online]. Available: 10.1109/PACT.2017.19
[18] T. Wang and I. Goldberg, “Walkie-talkie: An efficient defense against passive website fingerprinting attacks,” in *Proceedings of the 26th USENIX Security Symposium*, 2017, pp. 1375–1377. [Online]. Available: https://doi.org/10.1145/3399742
[19] Y. Weiss and E. Kitamura, “Aligning timers with cross origin isolation restrictions.” [Online]. Available: https://developer.chrome.com/blog/cross-origin-isolated-hr-timers
[20] A. M. Wladimir De la Cadena and J. Hiller, “Trafficsliver: Fighting website fingerprinting attacks with traffic splitting,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1971–1973. [Online]. Available: https://doi.org/10.1145/3372297.3423351

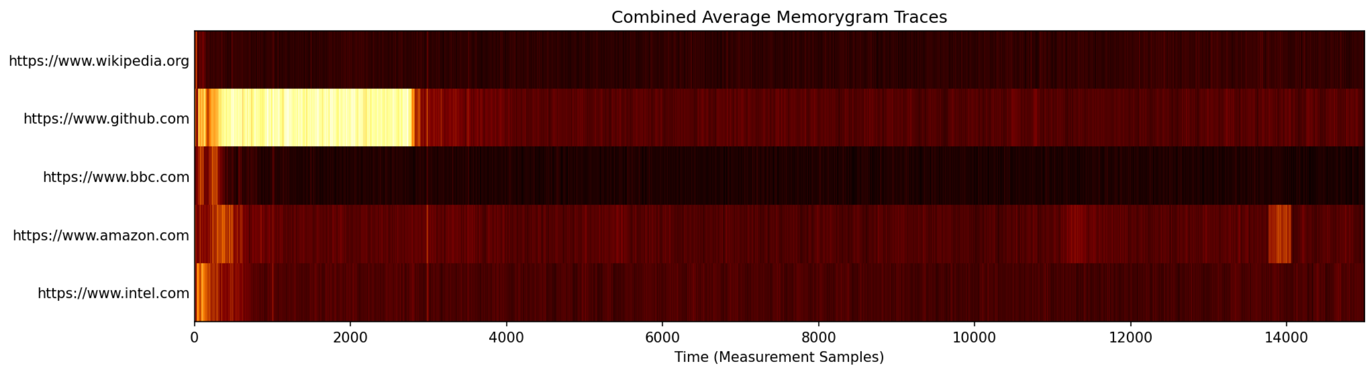

Fig. 2. Google Chrome traces performed on the system with the Intel i7-1260P on Linux

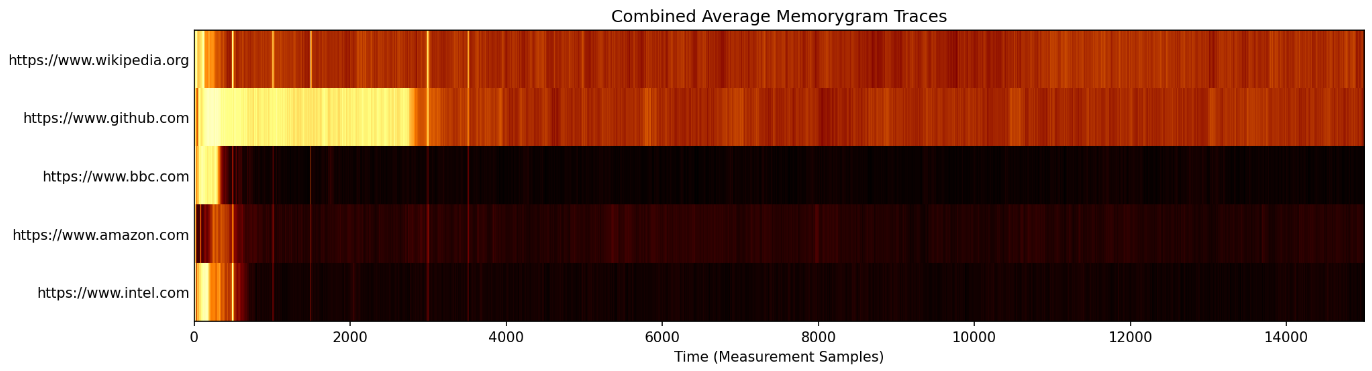

Fig. 3. Google Chrome traces performed on the system with the Intel i7-7700K on Linux

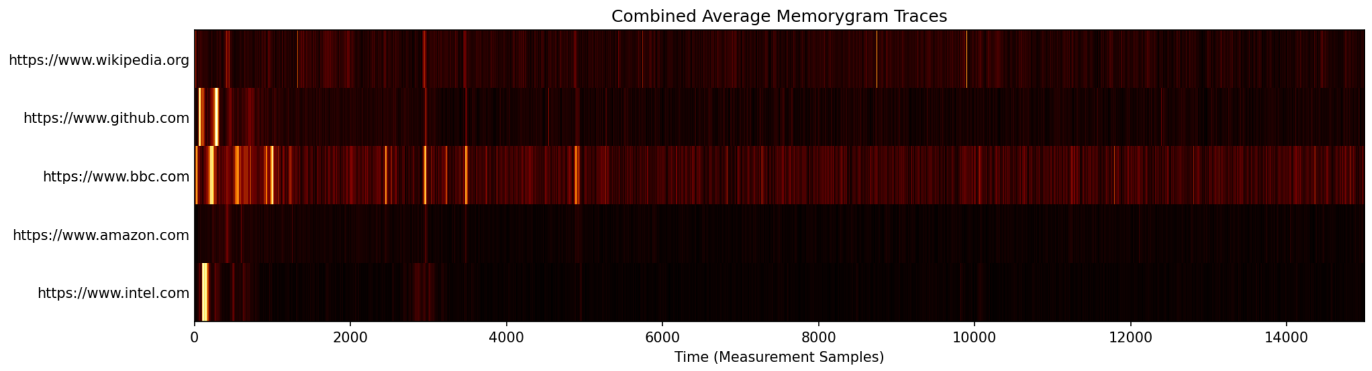

Fig. 4. Google Chrome traces performed on the system with the Intel i7-1260P on Windows

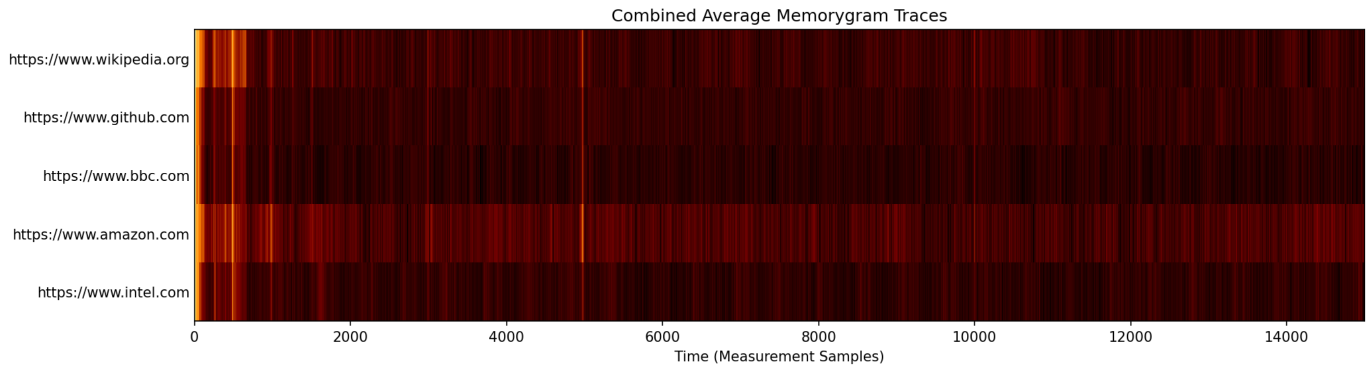

Fig. 5. Google Chrome traces performed on the system with the Intel i7-7700K on Windows

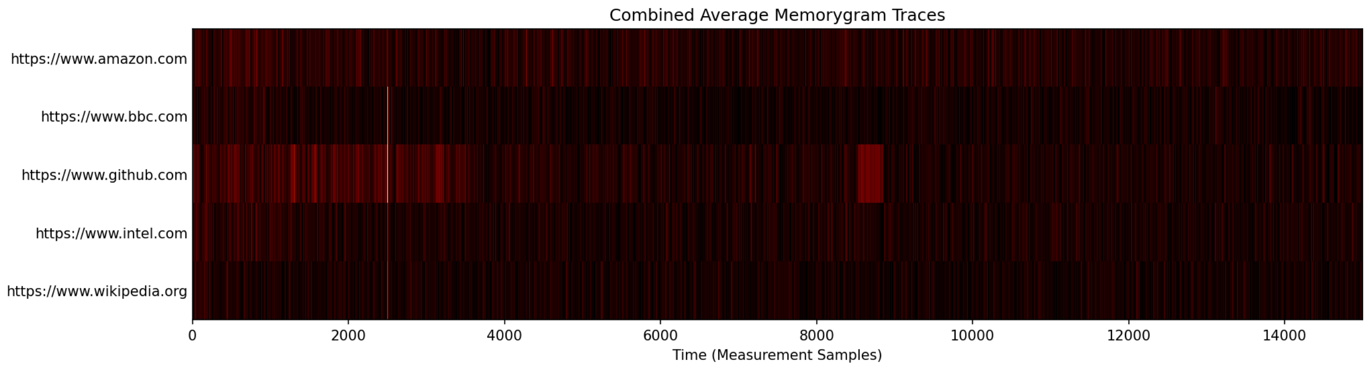

Fig. 6. Firefox traces performed on the system with the Intel i7-1260P on Linux

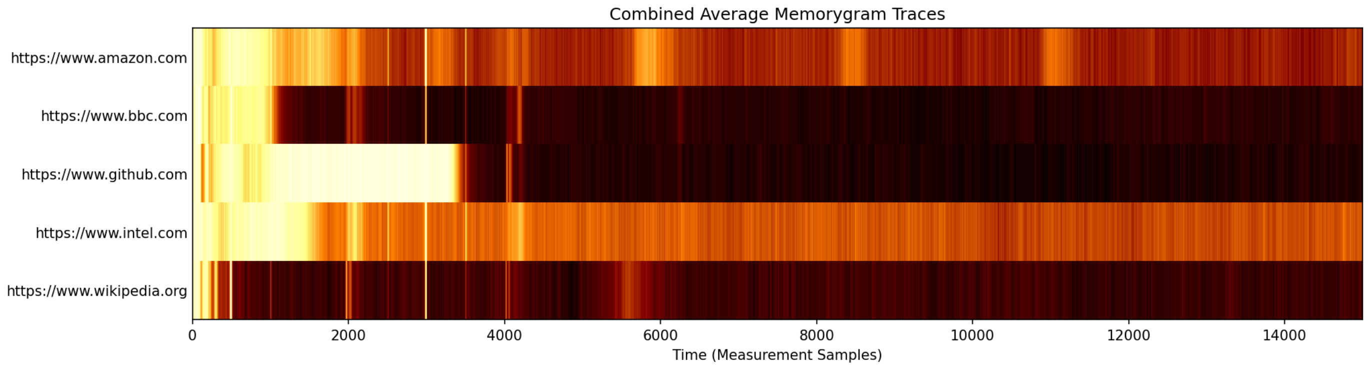

Fig. 7. Firefox traces performed on the system with the Intel i7-7700K on Linux

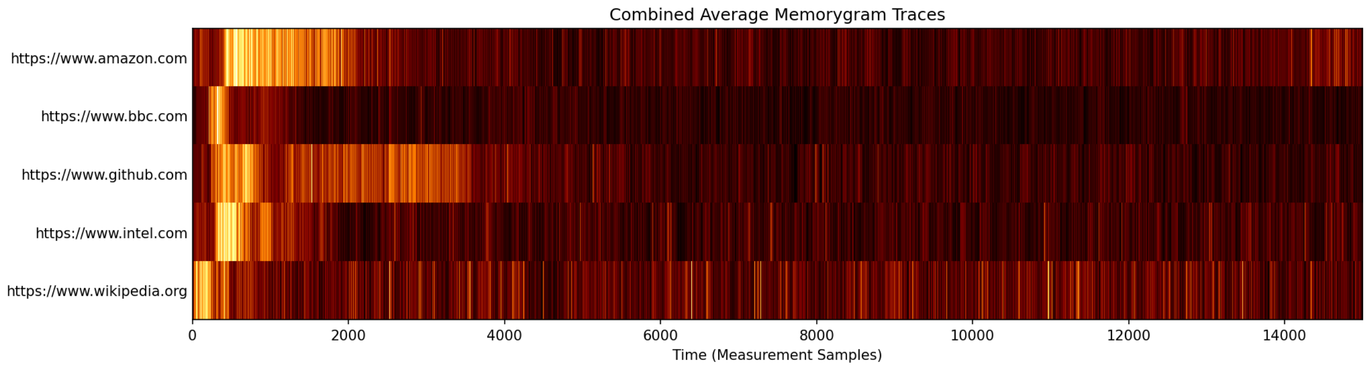

Fig. 8. Firefox traces performed on the system with the Intel i7-1260P on Windows

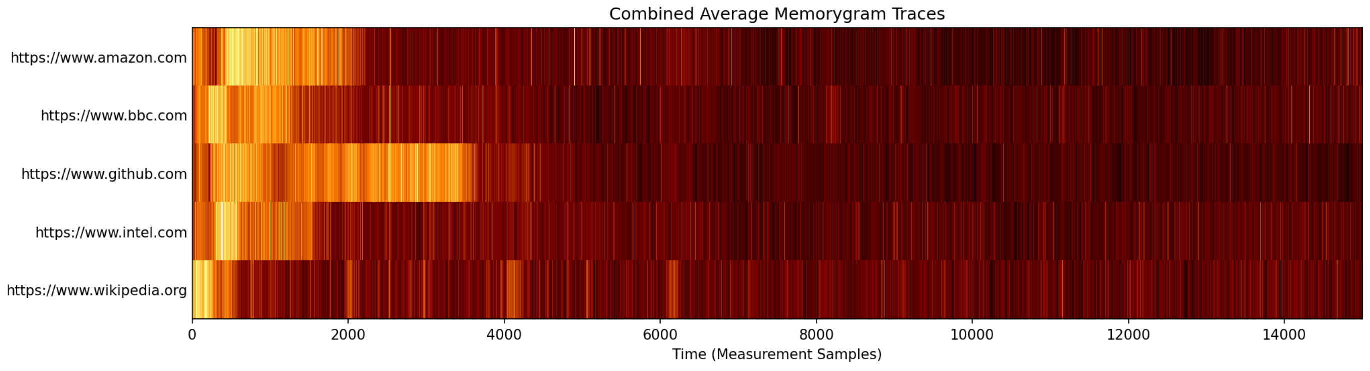

Fig. 9. Firefox traces performed on the system with the Intel i7-7700K on Windows