FlipStress: Noise Injection Defenses Against CPU-cache-based Web Attacks

Abstract

Side-channel attacks via the CPU-cache exploit hardware behavior to leak sensitive information about user activity. Two prominent forms of such attacks pose a serious privacy threat to web browsing users when they visit an attacker-controlled website. *Targeted deanonymization* allows an attacker to infer a specific user's identity, whereas *website fingerprinting* allows the attacker to infer which websites the user is visiting.

In this paper, we present **FlipStress**, a defense that mitigates these CPU-cache-based attacks by injecting artificial noise to obfuscate the cache patterns and render the side-channel information ineffective. FlipStress is designed to withstand strong attacks that leverage machine learning models to interpret the cache readings. Towards this goal, we start by developing several *stressor* programs that create artificial noise by performing continuous read or write operations on cache-sized data structures. We then consider progressively stronger attackers who incorporate increasingly more information about the defense mechanism into their machine learning pipeline. Conversely, we enhance gradually the defense strategy and converge on FlipStress, which injects artificial noise by switching a randomly picked stressor at regular time intervals.

We implement FlipStress in JavaScript as a browser extension, which makes it convenient to be activated/deactivated on demand without leaving the browser. Based on a comprehensive evaluation, we show that FlipStress effectively reduces attack success rates, bringing targeted deanonymization accuracy down to 57.5% (base rate 50%) and website fingerprinting accuracy down to 6.3% (base rate 1%). The overhead imposed by FlipStress ranges between 31%-226% and is tunable, as the user can control the intensity of the artificial noise by controlling the number of stressor instances. This work represents the first practical noise-based defense against cache-based web attacks available directly within the browser, offering a convenient and effective solution for protecting user privacy.

ACM Reference Format:
. 2024. FlipStress: Noise Injection Defenses Against CPU-cache-based Web Attacks. In *Proceedings of The fifteenth ACM Conference on Data and Application Security and Privacy (CODASPY '25).* ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Side-channel attacks are a class of attacks that exploits indirect information leaked through the physical behavior of hardware to infer sensitive data. Rather than directly attacking cryptographic protocols or network communications, these attacks gather unintended signals — such as timing information, power consumption, or cache usage patterns — revealing insights into the system's activities. *CPU-cache-based side-channel attacks*, a specific subset, take advantage of the CPU's cache behavior to expose private information such as user activity or sensitive data. These attacks can be particularly dangerous because they exploit fundamental hardware characteristics, making them difficult to patch or fully defend against at the software level.

This paper focuses on two prominent forms of such attacks that pose a serious privacy threat to web browsing users: *targeted deanonymization* and *website fingerprinting*. Both attacks exploit CPU-cache-based side channels to classify browsing activity.

Targeted deanonymization (TD) attacks [29, 30, 33, 34] allow an attacker who controls a website to learn whether a specific target user is browsing the website. The attacker only needs to know the target through a public identifier, such as an email address, a LinkedIn user identifier, or an Instagram handle. The attack leverages *state-dependent URLs (SD-URLs)*, which return different responses depending on a user's identity. An example of such SD-URLs are *leaky resources* [29, 33] such as images or videos, which are stored at resource-sharing services such as YouTube, Google Drive or Dropbox, and shared only with a specific target user. The attacker privately shares the resource with the target and then embeds this shared resource into the attack website. If a user visiting the attack website can access the embedded resource, this indicates that the current visitor is the intended target. These attacks are practical and scalable to execute, and can be used as a stepping stone towards executing more sophisticated attacks [34].

There are two main approaches to execute the attack. The first one uses cross-site leaks (XS-leaks) [30], which are a family of mechanisms that exploit behaviors that bypass the same origin policy to enable the attack, such as status code leaks, page content leaks, and header leaks [14]. The second one uses browser-based side channels, such as the CPU cache side channel, which allow a (JavaScript-based) spy process to infer whether the visiting user is able to access the leaky resource. Zaheri et al. [34] showed that the combination of cache attacks and a machine learning pipeline enables the attacker to execute the attack efficiently and scalably.

This work focuses on the second approach, which uses side channels to exploit fundamental hardware characteristics and is thus harder to defend against [1, 2]. In particular, CPU side channel attacks work even if the resource-sharing service does not allow the embedding of its resources, or when browsers disable third-party cookies for embedded resources. In contrast, XS-leaks are being gradually patched by browser vendors and/or resource-sharing providers as they are being discovered.

Website fingerprinting (WF) attacks enable an adversary to infer which websites a user is visiting by analyzing the side-channel data generated as different sites are loaded [12]. Prior work has shown that CPU cache usage is an effective source of side-channel

CODASPY '25, June 04–06, 2025, Pittsburgh, PA, USA.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

data for this attack [16, 26, 27]. Unlike traditional website fingerprinting, which relies on network traffic analysis, cache-based fingerprinting focuses on how the computer's cache behaves during the loading and rendering of web pages. This attack typically involves injecting malicious JavaScript into a webpage, which then monitors cache access times. Each website interacts with the cache in a distinctive way, creating identifiable patterns based on the resources (like images, scripts, or stylesheets) the website loads. By measuring cache occupancy or latency through these patterns, the attacker can "fingerprint" the websites being visited by the user.

Both targeted deanonymization and cache-based website fingerprinting exploit the lack of process isolation in the cache at the micro-architectural level, which in turn allows an attacker to undermine user anonymity and expose browsing activity.

Priorly proposed defenses: Two fundamental approaches can be used to defend against side-channel attacks. The first is prevention, which aims at making attacks theoretically impossible by removing any correlations with the side channel trace. In the context of targeted deanonymization attacks, Zaheri et al [34] proposed Leakuidator+, a browser extension which ensures that cross-site web requests are stripped of cookies and also eliminates a timing side-channel. The extension preserves existing web functionalities such as analytics and tracking. However, this solution requires users to decide whether certain requests are legitimate (such as authentication requests) and mark them as such. This requirement may negatively impact user experience and privacy, as the user needs to spend additional effort and may not always be able to correctly ascertain the legitimacy of requests. Additionally, Chrome's recent transition from Manifest V2 to V3 has also introduced significant changes to how extensions handle network request modifications. Instead of allowing real-time interception and modification of network requests via chrome.webRequest, extensions must now predefine rules governing actions based on specific conditions [20]. This change makes real-time cookie modification significantly more difficult, creating new challenges for extensions like Leakuidator+ that rely on dynamic, on-the-fly decision-making.

The second defensive approach is mitigation, which tries to make attacks impractical by reducing the signal-to-noise ratio of the side-channel trace. Noise injection is a key tactic in this strategy, effectively obfuscating the measurement of the side-channel signals that attackers rely on. In general, noise defenses have the advantage of being easy to apply to existing systems, since the noise is orthogonal to the signal, meaning that adding a noise generator requires require minimal changes to the design of the system being protected. The disadvantage of all noise-based approaches is that they can only reduce the leakage, but not completely prevent it. Thus, an attacker with sufficient measurement and post-processing ability may still be able extract the signal from the noisy trace.

Several works [9, 10, 17, 21, 23, 31, 35] utilize cache flush instructions to disrupt side-channel attacks, strategically clearing cache contents to thwart information leakage. While mitigation defenses have been thoroughly explored in the context of cryptanalysis side-channel attacks, they have not been as widely studied in web-based side-channel attacks. In the context of targeted deanonymization attacks, Zaheri et al. [34] considered a noise-based approach against CPU cache side channels: the user runs CPU stress tests outside the browser. Although they found this approach ineffective, we note that their conclusion only applies to the specific type of noise that was used and cannot be generalized.

Noise-based defenses have been more rigorously studied in the context of cache-based website fingerprinting (WF) attacks. *DefWeb* [26], for instance, leverages self-modifying code with precomputed noise templates during website rendering to mask actual cache usage. However, this approach lacks scalability and is difficult to maintain, as the noise templates are highly dependent on specific micro-architectures, operating systems, and the rendered websites themselves. *Cache Shaping* [16] obfuscates cache patterns through parallel read/write operations involving dummy files on the disk. Whereas both DefWeb and Cache Shaping were designed to mitigate WF attacks, we show that they are ineffective against targeted deanonymization attacks.

This work takes on the challenge of developing **FlipStress**, a practical, portable, and tunable noise-based defense that is generalizable across diverse cache-based attack types. To build this defense, we investigate noise-based approaches to mitigate targeted deanonymization attacks. Our initial exploration focused on the `stress-ng` [6] stress test suite, which includes over 270 CPU stress tests. This investigation shows that different stressors in the suite vary in their effectiveness at mitigating attacks. Some stressors can significantly reduce attack accuracy, both when the attacker is unaware of the defense and when the attacker trains a machine learning model while the stressor is active. Based on this analysis, we identify three high-performing "quality" stressors from the suite and analyze their source code. Additionally, we develop four custom "quality" stressors that perform continuous read and write operations on cache-sized data structures. We implement these seven stressors in JavaScript and package them into a browser extension, enabling users to activate the defense seamlessly within the browser whenever they require protection.

Next, we consider progressively stronger attacks that incorporate more information about the defense mechanism into the machine learning pipeline used for the attack, and also extend the duration of the attack to capture more information about the target. In complement, we enhance gradually the defense strategy to mitigate the stronger attacks. Notably, we show that a strategy that adds artificial noise by switching to a randomly picked stressor at regular time intervals provides an adequate defense over time, even against the strongest adversaries.

In summary, we make the following contributions in this work:

- We develop seven "quality" CPU stress programs (*i.e., stressors*) in JavaScript, which provide resilience against cache-based web attacks and form the basis of our noise-based defense. We independently develop four of these stressors. For the remaining three, we analyze the source code of `stress-ng` stressors to understand which kind of read and write operations provide effective artificial noise, and subsequently re-implement them in JavaScript.
- We extensively explore the targeted deanonymization attack and defense space. On the attack side, we consider progressively stronger adversarial strategies that feed the cache measurements collected through the side-channel into machine learning models. On the defense side, we enhance gradually the defense strategy by increasing the

amount of noise added (e.g., by increasing the number of stressor instances), incorporating randomness into the generated noise, and periodically changing the type of stressor used to generate noise. Our key findings are synthesized into FlipStress, a defense strategy that adds artificial noise by activating a randomly-selected "quality" stressor at regular time intervals. FlipStress is effective in providing sustained protection over time, even against powerful adversaries, by reducing the attack's success rate to 57.5%. FlipStress remains an effective defense across a variety of system configurations, including multiple operating systems (Linux, Windows, MacOS) and browsers (Chrome, Firefox, Tor).

- We also demonstrate that FlipStress is effective against cache-based website fingerprinting attacks. Specifically, in a closed-world scenario, FlipStress reduces the attack's success rate to as low as 6.3%.
- We implement FlipStress as a browser extension for Chrome, Firefox, and Tor, and make it publicly available so that it can be deployed immediately by any user needing protection. Users can activate or de-activate the defense on demand at the cost of a button click without leaving the browser. In addition, the defense allows users to control the intensity of the generated noise by controlling the number of stressor instances. In general, the more instances of the stressor are active, the more effective the defense becomes, at the cost of incurring more system overhead. To our knowledge, this is the first noise-based defense against cache-based web attacks available in the browser.
  The browser extension, as well as other artifacts, are publicly available at our anonymized artifact repository [3].

Our work suggests that noise-based defenses can provide practical and effective protection against CPU cache-based side-channel attacks. The rest of this paper is organized as follows: Section 2 reviews related work, Section 3 provides necessary background, and Section 4 outlines the threat model. Section 5 explores the TD attack and defense space, detailing our iterative process to develop our proposed defense. Sections 6 and 7 present results for TD and WF attacks, respectively. Section 8 evaluates performance overhead, and Section 9 discusses the results and future directions. We conclude in Section 10.

2 Related Work

Lyu and Mishra [19] survey recent cache-based side-channel attacks and countermeasures. In cache-based website fingerprinting, noise injection has been recently explored as a countermeasure. Cache masking [27] introduces noise by allocating an LLC-sized buffer and repeatedly accessing each cache line in a loop to evict and mask browser activities. Cook et al. [7] proposed generating interrupts by scheduling thousands of activity bursts and network pings at random intervals. Although disrupting cache usage patterns, these methods fail to significantly reduce attack accuracy when attackers retrain their models on defended traces.

DefWeb [26] takes a more advanced approach, using variational autoencoders (VAEs) to generate minimal noise templates and injecting precise noise into cache-based website fingerprints. Although effective in simulations, its reliance on precomputed templates tailored to specific microarchitectures makes it unscalable and challenging to adapt to diverse hardware or evolving website structures. Additionally, DefWeb struggles to obfuscate cache patterns of high-activity websites.

CacheShaping [16] adopts a different methodology by running multiple parallel processes to simulate browser rendering operations. It repeatedly reads data from dummy files into the LLC, evicts existing data, and writes the evicted data back to the files, masking real cache activities. While effective against website fingerprinting (WF) attacks, CacheShaping and the other website fingerprinting defenses have not been tested against targeted deanonymization, leaving their efficacy in such scenarios unverified.

For targeted deanonymization, Zaheri et al. [34] explored noise-based defenses by introducing cache noise through CPU stress tests [6] and web browsing activities. However, this approach proved ineffective when attackers retrained their models on defended cache traces, highlighting the limitations of current noise-based methods against adapted adversaries conducting targeted deanonymization.

3 Background

3.1 CPU-Cache side channels

A cache is a high-speed memory bank that temporarily stores recently accessed memory locations, bridging the performance gap between the processor's speed and the slower main memory. Modern processors typically employ a multi-level cache hierarchy, with the L3 or Last-Level Cache (LLC) being shared across all CPU cores. A cache hit allows for quick retrieval of data; if the data is not found in the cache, a cache miss occurs and the data is retrieved from the slower RAM memory.

The very nature of cache sharing introduces unintended communication channels, or side channels, which can be exploited to leak sensitive information. Side-channel attacks (SCAs) have been remarkably effective in undermining the security of hardware and software implementations in various cryptosystems [11, 18, 31]. These attacks leverage subtle characteristics like timing and power consumption to infer confidential data. Cache-based SCAs are particularly alarming because they exploit minute variations in cache access times to reveal memory access patterns, potentially exposing critical information such as encryption keys. These attacks have been demonstrated across cryptosystems and other domains, as highlighted by Zhou et al. [32], underscoring the urgent need for robust defenses.

3.2 The Cache Occupancy Attack

The Prime+Probe attack [18] is a prominent cache side-channel technique that exploits contention in specific cache sets to infer sensitive information. In this attack, the adversary first "primes" the cache by loading their own data into targeted cache sets. After allowing the victim process to execute, the attacker "probes" these sets to detect whether their data has been evicted, indicating that the victim accessed memory mapped to the same sets. By repeating this process, the attacker can discern the victim's memory access patterns, potentially revealing sensitive information such as encryption keys.

The success of this attack heavily relies on high-resolution timers to distinguish between cache hits and misses. Since the timing differences between accessing data in the cache and in main memory are extremely small — typically around 20 nanoseconds — accurate measurements are crucial. To defend against such timing-based attacks, modern browsers have reduced the resolution of the timers they provide, significantly hindering the effectiveness of attacks like Prime+Probe.

To overcome this limitation, Shusterman et al. [27] proposed the **cache occupancy attack**, which assesses contention across the entire Last-Level Cache (LLC) rather than targeting specific cache sets. This technique involves allocating an LLC-sized buffer and measuring the time required to access the entire buffer. Cache contention caused by victim processes evicting the attacker's buffer introduces measurable delays when reading this buffer, allowing the attacker to detect victim activity. Unlike Prime+Probe, the cache occupancy attack is robust to reduced timer resolution such as those available in web browsers. A variant of the cache occupancy attack, known as **sweep counting**, counts the number of times the buffer can be accessed within a fixed time interval, rather than timing a single pass through the buffer, enabling attacks for even coarser-grained timers, such as those available in the highly-secure Tor Browser.

3.3 The Targeted Deanonymization Attack

Targeted deanonymization attacks are an important class of attacks which threaten user anonymity. An attacker who has complete or partial control over a website seeks to learn the identity of specific target users who are browsing that website. The attacker knows a target only through a public identifier, such as an email address or social media handle.

To mount the attack, the attacker first uses a resource-sharing service like YouTube or Dropbox to bind a resource (e.g., a YouTube video) to the target via their public identifier. This binding can be achieved in two ways depending on which resource-sharing service is used: In the *sharing-based approach*, the attacker privately shares the resource with the target, using their public identifier. Alternatively, in the *blocking-based approach*, the attacker makes the resource publicly accessible, but blocks the target from accessing it.

The attacker then embeds this resource into the website under their control. When a visitor accesses the attacker-controlled website, the attacker observes whether the resource successfully loads. For example, if an embedded YouTube video — shared privately only with the target — loads, it indicates that the visitor is the intended target. In the blocking-based approach, the target is identified by the fact that the resource does not load. Zaheri et al. [34] demonstrate that this attack can be executed in less than one second and remains effective even when resource-sharing platforms restrict embedding and when browsers block third-party cookies.

To determine if the resource was loaded, the attacker uses the cache occupancy attack and measures the time needed to access the attack buffer while the attack page is loaded by the visitor. A long access time indicates the shared resource was loaded, since by doing so the visitor has evicted the attacker's buffer from the cache. To analyze the collected cache timings and make an accurate determination, the attacker employs a machine learning classifier.

3.4 The Website Fingerprinting Attack

Website fingerprinting attacks seek to compromise user privacy by using observable activity patterns to identify the websites a user visits. The original website fingerprinting attack assumes an on-path adversary, who leverages statistical analysis of encrypted network traffic metadata — such as packet sizes, timing, and direction — to infer the websites being accessed [12].

Shusterman et al. [27] proposed an alternative attack model which leverages cache-based side channels to bypass network-level defenses. This model assumes that the victim runs a web browser on a target machine and simultaneously accesses both an attacker-controlled site and a sensitive site. The attacker does not need full control over the malicious site; they only need the ability to inject JavaScript code into the victim's browser, thus exploiting the cache occupancy attack channel. By observing cache occupancy patterns, the attacker generates a "fingerprint" — a trace of cache activity over time — which reflects the victim's visited website.

The attack proceeds in two phases: In the *training phase*, the attacker visits a set of target websites and collects fingerprints corresponding to their cache activity. These traces are labeled and used to train a machine learning classifier, which learns to recognize the unique cache signatures associated with different websites. Next, in the *online phase*, when the victim accesses the attacker-controlled website, the malicious JavaScript code collects the fingerprint of the victim's cache activity in real time. These traces are then fed into the trained classifier, which identifies the sensitive website the victim is visiting based on its unique cache signature.

4 Threat Model

The threat model for targeted deanonymization and website fingerprinting attacks involves two parties: a user/victim and an attacker. The user browses the internet using a web browser, which may or may not employ anonymity networks such as VPNs or Tor. The attacker is remote and cannot eavesdrop on the user's network traffic. The user visits an attacker-controlled website containing malicious JavaScript code, which profiles the shared Last-Level Cache (LLC) using the cache occupancy channel [27].

For both attacks, the attacker formulates the problem as a supervised learning task consisting of two phases: a training phase involving offline data collection and model training, and an online phase involving classification of the user's data using the trained model. We assume that the used data for training the model is collected under settings similar to those of the user, including the same operating system, web browser, and LLC size. The attacker can use known techniques to infer this information about the user [25, 28].

For targeted deanonymization, we assume the adversary has some public information about the victim, such as their Twitter handle or email address. We note the adversary does not need to control the resource-sharing service exploited in the attack; they only need to be a registered user of the service. The adversary faces a binary classification problem: distinguishing between traces belonging to the "target" and those from "non-target" users. During the training phase, the attacker collects cache traces labeled as a "target" or "non-target" user. In the online phase, the attacker gathers an unlabeled cache trace from the user and uses the trained model to predict the labels of this trace, classifying the user as a target or a non-target.

For website fingerprinting, the attacker faces a multiclass classification problem. In the training phase, the attacker collects a set of traces corresponding to sensitive websites on a device identical to the victim's. The attacker then trains a multi-class classification model using this dataset. In the online phase, the attacker collects a single trace while the victim's browser renders a website, and this trace is classified using the trained model to identify the website.

5 Targeted Deanonymization: Attacks and Defenses

5.1 Approach Overview

In this section, we perform a systematic investigation of noise-based defenses against cache-based targeted deanonymization attacks. The main research question is whether we can devise a noise-based defense that is resilient against strong attacks. We consider progressively stronger attack strategies and devise defense strategies to mitigate them. This allows us to answer the research question in the positive, and develop a noise-based defense that combines the lessons learned.

At the core of our proposed defense are stress programs, referred to as *stressors*. Each stressor performs read/write operations on memory buffers that are as large as the last-level cache (LLC). Our defense runs while the user is browsing the web, and aims to make cache traces indistinguishable between target and non-target users. We develop seven stressors, which are incorporated into the defense strategies. Section 5.2.2 provides a description of these stressors. In all defense strategies, we use p to denote the number of stressor instances that are running in parallel, where each stressor instance is typically running on a separate CPU core. Having multiple instances introduces non-determinism into cache access patterns, preventing optimizations like prefetching from affecting the CPU's cache replacement policy. The increased randomness reduces the attacker's ability to accurately classify the cache traces, lowering the overall effectiveness of the attack.

We use the term *scenario* to denote the capabilities used by the attacker and the defender (*i.e.*, a pairing of an attack strategy with a defense strategy). As we move through scenarios, we consider progressively stronger attack strategies by incorporating more information about the defense mechanism into the attacker's machine learning pipeline, by extending the attack duration to capture more information about the target, and by assuming a more advanced threat model. Conversely, we progressively enhance the defense strategy by increasing the amount of added noise (e.g., by increasing the stress intensity), by incorporating randomness into the generated noise, and by periodically switching between stressors.

Fig. 1 outlines the progression of the nine scenarios considered. Previous work [34] demonstrated that targeted deanonymization attacks could succeed in as little as one second. Thus, we first examine whether noise-based defense strategies can effectively counter such short-duration attacks. The initial "1s Attack" scenarios 1-3 present foundational approaches, starting with basic attacks like training without stress, countered by defenses such as fixed stressors. As the scenarios progress, attack strategies evolve to include methods like training with known fixed stressors or ensemble models, while defense strategies incorporate techniques like randomness to improve resilience. We then evaluate attacks lasting up to 10 seconds in the "10s Attack" scenarios 4-6, which mirror the methods in scenarios 1-3 but allow for an extended cache trace collection. This introduces new challenges for the defenses to remain effective over time. Finally, scenarios 7-9 introduce our main defense, FlipStress, which disrupts more sophisticated attack strategies by switching to a randomly-picked stressor at regular intervals.

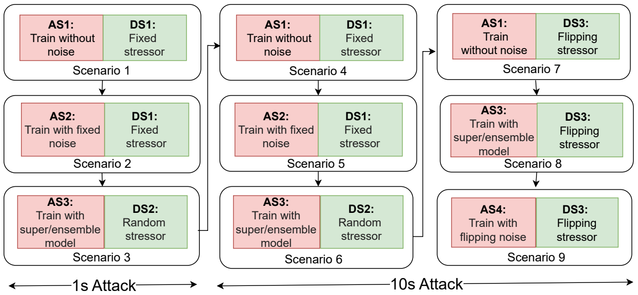


Figure 1: Flow diagram of the attack and defense strategies. We consider nine scenarios, based on four Attack Strategies (AS1-AS4) and three Defense Strategies (DS1-DS3).

5.2 Attack and Defense Methodologies

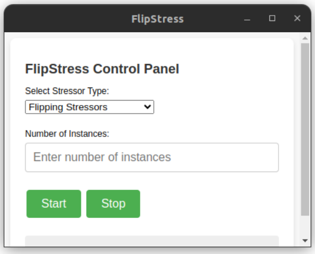
5.2.1 *Attack Methodology.*

We follow the attack methodology used by Zaheri *et al.* [34], which we summarize next. The attack begins with the sharing of a leaky resource. The attacker uploads a YouTube (YT) video to the YouTube sharing service and privately shares the video with the target using the victim's email address. The attacker then embeds the URL of the YT video into a webpage they control. When the target visits this webpage, their browser loads the embedded resource. The attack proceeds in two main phases:

Training Phase: The attacker collects cache activity traces for when the YT video is loaded and when it is not. These traces are used to train a machine learning classifier to identify the cache signature associated with successfully loading the YT video.

Online Phase: The victim visits the attacker-controlled webpage that loads the YT video. As the video is loaded and rendered, the attack script continuously takes cache measurements on the victim's computer. These measurements are then fed into the trained classifier to determine whether the user is the target. The attack's effectiveness is determined both by its accuracy and by the *attack duration* parameter, which represents the time for which the attacker collects cache traces. Our cache occupancy code is based on the PP0 repository [22].

The Attack Variant: We consider the basic attack variant that embeds a shared YouTube video using an <iframe> in the Chrome browser. Although Zaheri *et al.* [34] showed that the targeted deanonymization attack has a large attack surface, including additional attack variants, additional resource-sharing services (including Twitter, LinkedIn, TikTok, Facebook, Instagram and Reddit), and additional browsers (including Firefox and Safari), we argue that a successful defense against this basic scenario will also be applicable for other embedding methods, other resource-sharing services, and other browsers. This is because the underlying principles of the attack and defense mechanisms are sufficiently general across



FlipStress Control Panel

Select Stressor Type: Flipping Stressors

Number of Instances: Enter number of instances

Figure 2: The FlipStress browser extension defense: Users can select the type of stressor, the number of stressor instances, and can activate/deactivate the defense.

different browsers and resource-sharing services. Additionally, web standards and consistent browser behaviors indicate that defenses effective in one context are likely to be effective across others. We provide evidence for this claim later in this section, where we show our defense remains effective on a variety of system configurations (*i.e.*, combination of OS, browser, and resource-sharing service).

5.2.2 Defense Methodology: Stressors. The defense is implemented as a browser extension that runs stressor programs continuously throughout the user's browsing session. The stressors are implemented as JavaScript web workers. In a browser environment, web workers provide a means to run concurrent scripts in background threads, independent of the main execution thread. Each web worker operates with its own memory space, much like how forked processes do in a native environment. The number of web workers corresponds to the parameter p, representing the degree of parallelism. This parallelism is crucial for generating significant cache contention, thus maximizing the effectiveness of stressors to disrupt the attacker's ability to analyze the cache patterns. Fig. 2 shows the browser extension defense.

We designed and implemented seven stressors to create different types of memory access patterns, each tailored to stress the system in unique ways. For the first three stressors, we turned our attention to the `stress-ng` stress suite [6]. In prior work, Zaheri *et. al.* [34] attempted to add artificial noise by using `stress-ng` stressors, but the stressors they considered were found to be ineffective. We re-examined Zaheri et. al.'s claims regarding `stress-ng` as a source of noise. By broadening the pool of considered stressors, we were able to identify three `stress-ng` stressors which provide resilience. We examined their source code, and re-implemented them in JavaScript as part of our defense. We developed the other four stressors independently. The seven stressors are: *Read Buffer, Write Buffer, Read Linked List, Write Linked List, Stream, VM,* and *Memcpy.* These are described in more detail in Appendix B and their source code is provided in the FlipStress code repository [3].

5.2.3 Experimental Setup. We conducted most of the experiments using the Chrome browser, on a Lenovo ThinkPad P14s laptop with an Intel Core i7-1260P CPU running Ubuntu 22.04.1 LTS. To demonstrate the universality of our defense, we used several additional system configurations, which include the Firefox and Tor browsers, and Windows OS. These are all detailed in Appendix A.

The attack page contains a YouTube (YT) video that is privately shared with the target. The page also contains a JavaScript script that takes CPU cache measurements. We automated the cache measurements collection using Selenium. We use supervised machine learning to analyze the cache measurement data. In general, to build our data sets, we collect 200 cache occupancy samples while the attack page is loaded — 100 for the target and 100 for the non-target — a subset of which is used to train the classifier, which is then used to predict whether a user loading the attack page is the targeted victim. We used logistic regression for the exploratory portion of the experiments. We also validated the effectiveness of FlipStress using a convolutional (CNN) and a long short-term memory (LSTM) neural network model. The parameters used for these classifiers are provided in Appendix A. Data analysis was performed using Scikit-Learn v1.5.2 [4] and Tensorflow v2.17.1 [5] with Python v3.10.12 on Google Colab [24].

5.3 Attack and Defense Strategies

In this section, we aim to systematically evaluate the effectiveness of different defensive measures against a range of potential attacks. To do so, we explore 9 specific scenarios based on *4 attack strategies (AS)* and *3 defense strategies (DS)*, as illustrated in Fig. 1.

Guided by the findings of Zaheri et al. [34], we initially set the attack duration to *1 second*, which was previously identified as sufficient for mounting a successful attack. An effective defense can theoretically reduce attack accuracy to approximately 50%, equivalent to a random guess by the attacker. Higher accuracy values suggest that the attacker can still extract meaningful information and perform deanonymization to some extent.

5.3.1 ***Scenario 1: AS1+ DS1 (1s):***

AS1 — Train without noise: The initial attack strategy evaluates an attacker using a model trained on clean cache traces, simulating an environment without defensive noise.

DS1 — Fixed stressor: The defender employs a single, consistent stressor that injects continuous noise into the cache. This strategy aims to disrupt the attacker's model by masking cache traces with persistent noise, challenging the model's classification capabilities.

No.	Stressor	p = 1	p = 2	p = 4
1	Read Buffer	99.00%	98.00%	90.50%
2	Write Buffer	50.00%	50.00%	50.00%
3	Read Linked List	50.00%	50.00%	50.00%
4	Write Linked List	50.00%	50.00%	50.00%
5	vm	50.00%	50.00%	50.00%
6	Stream	50.00%	50.00%	50.00%
7	Memcpy	99.50%	82.50%	50.00%
	Average	64.07%	61.50%	55.79%

Table 1: Attack accuracy for Scenario 1 (AS1 + DS1, 1s).

The interplay between AS1 and DS1 is quantified in Table 1. We observe that not all stressors demonstrate equal effectiveness. More specifically, stressors 2 through 6 significantly stand out in their capacity to disrupt the attack and consistently reduce attack accuracy to 50% across all levels of p. The average accuracy over all seven stressors indicates a clear trend: as the number of stressor instances, p, increases, the effectiveness of the defense also improves.

5.3.2 *Scenario 2: AS2 + DS1 (1s):*

In Scenario 2, we introduce a more advanced threat model by assuming that the attacker possesses detailed knowledge of the defensive measures in place via DS1.

AS2 — Train with fixed noise: The attacker has precise knowledge of the fixed stressor used by the defender. Leveraging this information, the attacker trains a model using cache traces collected in the presence of this specific stressor.

No.	Stressor	p = 1	p = 2	p = 4
1	Read Buffer	95.00%	99.00%	94.50%
2	Write Buffer	91.50%	55.00%	52.50%
3	Read Linked List	99.50%	92.00%	57.50%
4	Write Linked List	86.00%	69.00%	64.50%
5	vm	94.50%	46.00%	52.00%
6	Stream	59.00%	61.50%	64.50%
7	Memcpy	99.50%	99.00%	82.00%
	Average	89.29%	74.50%	66.79%

Table 2: Attack accuracy for Scenario 2 (AS2 + DS1, 1s).

The results in Table 2 show that AS2 proves to be a superior attack strategy compared to AS1. For p=1 and p=2, the defense is ineffective for most stressors. However, as the value of p increases to 4, there is a notable decline in the attack accuracy. This scenario starkly contrasts with the simpler attack model in AS1, where lower values of p might still offer substantial protection.

5.3.3 *Scenario 3: AS3 + DS2 (1s):*

In this scenario, we elevate the defense strategy to DS2, in which the defender employs a stressor selected at random from the set of available stressors.

DS2 — Random stressor: The defender deploys a randomly chosen stressor, which operates continuously. The randomness introduces uncertainty for the attacker, as they can no longer predict which specific noise will be present in the cache traces.

Note that we do not evaluate the combination of AS2 and DS2 because AS2 relies on the attacker training with a known, fixed stressor. Since DS2 employs a random stressor, the premise of AS2 is invalidated; an attack model trained on a specific stressor cannot effectively generalize to the varying conditions imposed by DS2.

AS3 — Train with super/ensemble model: To counter the unpredictability introduced by DS2, the attacker adopts more advanced techniques that aggregate traces from multiple stressors, which we divide into two approaches - AS3a and AS3b.

- *AS3a — Super model:* The attacker aggregates cache traces from all stressor types and trains a single "super model". This model aims to increase predictive accuracy and robustness by pooling data from various stressor-influenced cache traces. This approach allows the attacker to detect patterns across different noise conditions, attempting to classify effectively even under a random environment imposed by DS2.
- *AS3b — Ensemble model:* The attacker builds separate models for each stressor's cache traces. This strategy leverages the strengths of multiple models, each assessing individual stressor-influenced data independently, which collectively contribute to the final decision based on a majority vote.

Attack Strategy	$p = 1$	$p = 2$	$p = 4$
AS3a + DS2	75.29%	69.43%	58.29%
AS3b + DS2	85.71%	77.14%	62.14%

Table 3: Attack accuracies for Scenario 3 (AS3 + DS2, 1s).

The results presented in Table 3 demonstrate that both the super model (AS3a) and the ensemble model (AS3b) achieve higher attack accuracies at lower values of p, even without knowledge of the specific stressor used by the defender. However, as the number of stressor instances p increases to 4, both strategies notice a noticeable decline in attack accuracy. This indicates that increasing the number of instances of stressors remains an effective way to mitigate even these stronger attacks.

5.4 Extending the Attack Duration

So far, we have explored attack and defense strategies within a constrained attack duration of 1 second. Our results show that DS2 is effective against all attack strategies when the number of stressor instances (p) is sufficiently high. However, in real-world scenarios, an attacker could extend the observation period, collecting cache traces over a longer duration, thereby increasing the likelihood of a successful attack.

In this section, we examine the implications of extending the attack duration from 1 second to 10 seconds. We revisit the three scenarios discussed earlier — now referred to as Scenarios 4-6 — with the only difference being the extended attack duration. This allows us to assess how a longer observation window impacts the effectiveness of both the attack and defense strategies.

No.	Stressor	p = 1	p = 2	p = 4
1	Read Buffer	100.00%	100.00%	57.00%
2	Write Buffer	50.00%	50.00%	50.00%
3	Read Linked List	50.00%	50.00%	50.00%
4	Write Linked List	50.00%	50.00%	50.00%
5	vm	50.00%	50.00%	50.00%
6	Stream	50.00%	50.00%	50.00%
7	Memcpy	98.00%	58.00%	50.00%
	Average	64.00%	58.29%	51.00%

Table 4: Attack accuracy for Scenario 4 (AS1 + DS1, 10s).

5.4.1 *Scenario 4: AS1 + DS1 (10s):*

The results presented in Table 4 indicate that, in this particular scenario, the longer attack duration does not significantly benefit the attacker. Specifically, the consistent attack accuracy of 50% for stressors 2 through 6 remains unchanged compared to the 1-second scenario. This consistency implies that these stressors are highly effective against AS1, regardless of the attack duration.

5.4.2 *Scenario 5: AS2 + DS1 (10s):*

The results presented in Table 5 show that extending the attack duration increases the attack accuracy. This suggests that a more extended observation period allows the attacker to gather more data, thereby effectively countering the fixed stressor defense (DS1). As p increases, the attack accuracy generally decreases, but to a lesser extent Scenario 2. This suggests that while increasing p introduces

No.	Stressor	p = 1	p = 2	p = 4
1	Read Buffer	100.00%	100.00%	97.00%
2	Write Buffer	98.00%	84.50%	65.50%
3	Read Linked List	99.50%	99.00%	83.00%
4	Write Linked List	98.00%	96.00%	73.50%
5	vm	100.00%	67.00%	70.00%
6	Stream	63.50%	61.50%	57.00%
7	Memcpy	100.00%	99.00%	85.00%
	Average	94.14%	86.71%	75.86%

Table 5: Attack accuracy for Scenario 5 (AS2 + DS1, 10s).

more noise, the extended time lets the attacker refine their model, mitigating some of the defense's effectiveness. The results indicate that while DS1 can disrupt shorter attacks, it may not be robust enough for a longer duration.

5.4.3 *Scenario 6: AS3 + DS2 (10s):*

The results presented in Table 6 indicate that in Scenario 6, the attack accuracy consistently improves compared to Scenario 3. This improvement aligns with the trend observed in the preceding scenario, suggesting that the super and ensemble models in AS3 significantly benefit from the extended attack duration.

Attack Strategy	$p = 1$	$p = 2$	$p = 4$
AS3a + DS2	82.29%	77.14%	70.50%
AS3b + DS2	92.85%	93.57%	75.00%

Table 6: Attack accuracies for Scenario 6 (AS3 + DS2, 10s).

Overall, the findings from Scenarios 4 through 6 imply that both DS1 and DS2 are inadequate in defending against sophisticated attack strategies such as AS3 when the attack duration is extended. This underscores the necessity for a more robust defense mechanism to counter advanced attacks over more extended periods.

5.5 The Flipping Stressor Defense (FlipStress)

This section introduces our primary defense strategy: the Flipping Stressor Defense, or FlipStress, which mitigates extended attack durations by dynamically altering the noise patterns throughout the attack window. FlipStress periodically changes the active stressor, aiming to disrupt the attacker's ability to collect consistent cache traces over more extended periods, thereby mitigating the increased risk of extended observation times.

5.5.1 *Scenario 7: AS1 + DS3 (10s):*

DS3 — Flipping stressor: In this defense strategy, we introduce a more unpredictable countermeasure by switching to a randomly-picked stressor at regular intervals of 0.5 seconds[1]. This unpredictable change in the stressor environment makes extracting meaningful patterns from the cache traces significantly harder.

Table 7 shows that in Scenario 7, which combines AS1 with DS3, the defense consistently scores 50% accuracy. This suggests that the random and frequent switching of stressors effectively neutralizes the attack, reducing it to a random guess.

p = 1	p = 2	p = 4
50.00%	50.00%	50.00%

Table 7: Attack accuracy for Scenario 7 (AS1 + DS3, 10s).

5.5.2 *Scenario 8: AS3 + DS3 (10s):*

In this scenario, we evaluate the effectiveness of the flipping stressor defense (DS3) against AS3a (super model) and AS3b (ensemble model) over a 10-second attack duration. Table 8 shows that DS3 is successful against this strong attack, maintaining low accuracy levels.

Attack Strategy	$p = 1$	$p = 2$	$p = 4$
AS3a + DS3	64.00%	55.50%	63.50%
AS3b + DS3	50.00%	50.00%	50.00%

Table 8: Attack accuracy for Scenario 8 (AS3a-b + DS3, 10s).

5.5.3 *Scenario 9: AS4 + DS3 (10s):*

AS4 — Train with flipping noise: In this attack strategy, the attacker adjusts their strategy to match the defender's approach, where the active stressor is flipped randomly at regular intervals. Therefore, the attacker trains the model in the presence of flipping noise.

p = 1	p = 2	p = 4
79.00%	69.00%	64.00%

Table 9: Attack accuracy for scenario 9 (AS4 + DS3, 10s).

Table 9 shows that AS4 represents the best attack considered so far against DS3. However, despite the attack's effectiveness at lower values of p, the DS3 flipping defense remains robust, significantly reducing the attack's success as p increases to 2 and 4.

Given the results across Scenarios 7-9, DS3 consistently demonstrates its strength in mitigating attacks, even against the most effective strategies like AS4. The ability of DS3 to maintain low attack accuracies, particularly as stress intensity increases, confirms that it is the most effective defense strategy among those tested.

Table 10 summarizes all attack and defense strategies.

AS1	Train without noise
AS2	Train with fixed noise (fixed stressor)
AS3	Train with aggregated noise: 3a — Super model (combining all stressors) 3b — Ensemble model (separate model for each stressor, majority vote)
AS4	Train with flipping noise
DS1	Fixed stressor (single fixed stressor)
DS2	Random stressor (single randomly picked stressor)
DS3	Flipping stressor (switch to a randomly picked stressor at regular intervals)

Table 10: Summary of Attack Strategies (AS) and Defense Strategies (DS)

Table 11 summarizes the attack accuracies across various scenarios with different attack durations and defense strategies. We note

[1] We selected a 0.5 second switching interval based on experimental observations. In earlier scenarios, we observed that attack accuracies remained relatively low with shorter attack durations, such as 1 second, in Scenarios 1-3. This suggests that brief time frames effectively maintain the defensive advantage – the shorter interval means the attacker has less time to gather consistent data before the stressor changes.

No.	Scenario	p = 1	p = 2	p = 4
1	AS1 + DS1 (1s)	64.07%	61.50%	55.79%
2	AS2 + DS1 (1s)	89.29%	74.50%	66.79%
3	AS3a + DS2 (1s)	75.29%	69.43%	58.29%
	AS3b + DS2 (1s)	85.71%	77.14%	62.14%
4	AS1 + DS1 (10s)	64.00%	58.29%	51.00%
5	AS2 + DS1 (10s)	94.14%	86.71%	75.86%
6	AS3a + DS2 (10s)	82.29%	77.14%	70.50%
	AS3b + DS2 (10s)	92.85%	93.57%	75.00%
7	AS1 + DS3 (10s)	50.00%	50.00%	50.00%
8	AS3a + DS3 (10s)	64.00%	55.50%	63.50%
	AS3b + DS3 (10s)	50.00%	50.00%	50.00%
9	AS4 + DS3 (10s)	79.00%	69.00%	64.00%

Table 11: Attack accuracy for all 9 scenarios.

that extending the attack durations generally results in higher attack accuracy. It is worth noting that the DS3 strategy consistently mitigates the attack, maintaining low accuracy levels even over an extended attack duration. Whereas AS4 shows some effectiveness against DS3 for low values of p, it struggles as the defense intensity increases, reinforcing FlipStress as the most effective overall defense strategy.

6 Results: Targeted Deanonymization

In the previous section, we laid the groundwork for understanding the effectiveness of our defense by focusing exclusively on an attack pipeline that uses Logistic Regression (LR). This initial exploration aimed to demonstrate the iterative process that led to the development of FlipStress. Building on these insights, this section seeks to rigorously evaluate FlipStress. We start by shifting the focus to using more advanced machine learning models. We then assess the defense's resilience against prolonged attacks, compare its effectiveness with existing defenses, and demonstrate its universality across different system configurations.

Advanced Machine Learning Models. We test the effectiveness of FlipStress against deep learning models, specifically Convolutional Neural Networks (CNN) [15] and Long Short-Term Memory networks (LSTM) [13]. These models, known for capturing complex patterns in data, present a more rigorous evaluation of our defense under adversarial conditions.

p	LR	CNN	LSTM
1	79.0 ± 10.91%	99.0 ± 2.0%	88.0 ± 7.48%
2	69.0± 4.90%	94.0 ± 5.39%	86.5 ± 7.43%
4	64.0 ± 8.60%	81.5 ± 8.08%	79.5 ± 9.07%
6	50.5 ± 10.83%	82.0 ± 8.72%	89.0 ± 7.00%
8	52.5 ± 11.67%	74.5 ± 10.59%	76.0 ± 11.58%
10	57.0 ± 7.14%	57.5 ± 11.46%	62.5 ± 8.14%
12	49.0 ± 7.68%	56.0 ± 10.68%	57.5 ± 11.01%

Table 12: Attack accuracy for FlipStress.

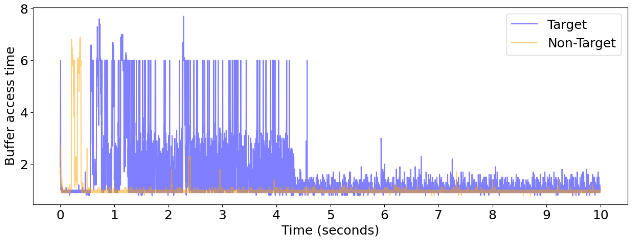


Figure 3: Buffer access time for target and non-target with no noise.

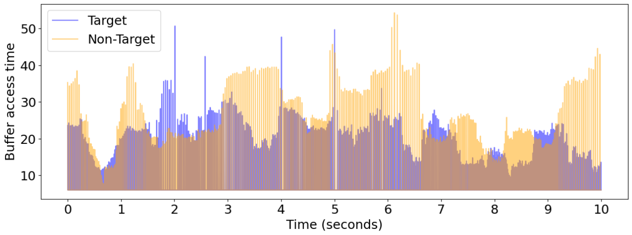


Figure 4: Buffer access time for target and non-target with FlipStress, when p=12.

Table 12 shows the attack accuracy when using different machine learning models, as the number of stressor instances, p, increases. Appendix A details the hyperparameters used for these models. CNN and LSTM outperform LR across all p values. While p=4 was a strong starting point against LR, achieving meaningful reductions in the performance of advanced models like CNN and LSTM required increasing p further up to 12 instances.

Figures 3 and 4 illustrate the impact of FlipStress on the attacker's buffer access times. Without noise introduced by FlipStress, as seen in Figure 3, the buffer access times for the target and non-target scenarios are easily distinguishable. In contrast, the application of FlipStress, as illustrated in Figure 4, introduces an unpredictable pattern in the buffer access times measured by the attacker. By rapidly switching and altering cache usage, FlipStress disrupts the attacker's cache readings, making them highly irregular. As a result, the buffer access times become randomized, with the target exhibiting higher readings at some times and the non-target at other times. This obfuscation effectively conceals the underlying patterns, rendering them much more challenging for machine learning models to classify accurately.

Resilience Against Prolonged Attacks. In practical attack scenarios, the attacker is not constrained to short durations such as 10 seconds. To evaluate the sustainability of FlipStress, we tested its performance over a prolonged attack duration of 60 seconds. Table 13 shows that FlipStress continues to disrupt adversarial classification even during extended attacks effectively. Figure 5 shows that the attack accuracy remains relatively constant after an initial rise and some fluctuations. This empirically reinforces FlipStress as a reliable long-term defense mechanism.

p	LR	CNN	LSTM
12	65.5% ± 11.06%	57.5% ± 7.83%	63.5% ± 9.23%

Table 13: FlipStress attack accuracy for 60 second attacks.

Comparison with existing defenses. Most prior research into noise-based countermeasures against cache-based attacks on the web has been done in the context of website fingerprinting (WF). However, the underlying principle of these defenses — introducing noise to disrupt the attacker's ability to extract meaningful patterns

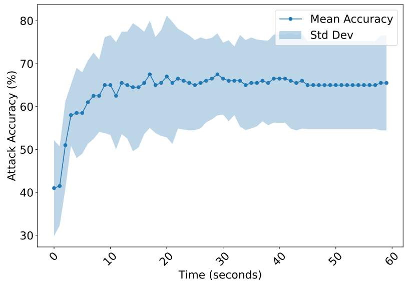


Figure 5: FlipStress attack accuracy over time (using LR).

in cache timings — can be extended to targeted deanonymization. Table 14 compares FlipStress with two such existing defenses, adapted for the targeted deanonymization attack. Since the attacker's website is unknown to the defender, we used the variant of DefWeb [26] that generates practical noise with random noise templates across all 12 physical cores. For CacheShaping [16], we utilize all 12 physical cores while performing read/write operations on 128 files. FlipStress outperforms these defenses.

Defense	LR	CNN	LSTM
DefWeb [26]	85.5% ± 6.5%	95% ± 4.15%	95% ± 4.15%
Cache Shaping [16]	62.5% ± 6.8%	99% ± 1.5%	100% ± 0%
FlipStress	49.0 ± 7.68%	56.0 ± 10.68%	57.5 ± 11.01%

Table 14: Targeted Deanonymization attack accuracy against various defenses (FlipStress, DefWeb, CacheShaping), under various models (LR, CNN, LSTM) (number of stressor instances p=12).

Defense Universality Across Diverse System Configurations. To assess the universality of FlipStress, we tested it across different operating systems (Linux, Windows, MacOS) and browsers (Chrome, Firefox, Tor) under the setups described in Appendix A. Table 15 shows that our proposed defense FlipStress remains effective under these diverse configurations.

7 Results: Website Fingerprinting

To evaluate the effectiveness of FlipStress against the website fingerprinting attack, we conduct a closed-world evaluation, a widely accepted methodology for assessing fingerprinting defenses. Similar to Son *et al.* [26], our evaluation uses 100 websites from Alexa's most visited websites list [8] and collects 100 traces from each website, for a total of 10,000 traces. We used an attack duration of 30s, with cache measurements at a sampling period of 2ms.

Table 16 compares FlipStress with DefWeb [26] and CacheShaping [16] as a defense against website fingerprinting attacks. The results for DefWeb (variant using practical noise generation) and CacheShaping are sourced directly from their respective studies. The second column of the table specifies the number of CPU cores utilized in each study. In particular, the reported accuracies for DefWeb and CacheShaping are based on scenarios in which the systems used 100% of their available physical cores. To ensure a fair comparison, we also evaluate FlipStress under 100% core utilization, specifically using $p = 12$ instances, corresponding to the maximum number of physical cores available on the test machine used for our experiments. The term "top-5 accuracy" refers to the percentage of instances where the correct website is among the top 5 predictions made by the attacker model. This metric provides a more lenient measure of attack success than strict top-1 accuracy, reflecting scenarios where attackers may prioritize a small set of possible targets.

FlipStress outperforms DefWeb and CacheShaping by achieving a significantly lower attack accuracy of 6.3%, compared to 28.8% for DefWeb and 10.9% for CacheShaping with Chrome and Linux. To explain the effectiveness of FlipStress, Appendix C includes additional insights about the impact of FlipStress on the attacker's buffer access times.

Defense	No. of Cores/Instances (Utilization)	CNN
DefWeb [26]	6 (100%)	28.8%
CacheShaping [16]	8 (100%)	10.9%
FlipStress	12 (100%)	6.3%
FlipStress (top-5)	12 (100%)	23%

Table 16: Comparison of defenses against the Website Fingerprinting attack, using the CNN model.

8 Performance Overhead

In this section, we evaluate the performance overhead introduced by our proposed defense at varying levels and compare it with existing defenses. To measure the overhead, we adopt the methodology proposed by Son et al.[26], which focuses on assessing the impact of the defense on website loading times. More precisely, we activate the defense and monitor the beginning and completion of the website rendering process by recording the difference in timestamps from the *performance.timing.navigationStart* and *performance.timing.loadEventEnd* functions. While Son et al. [26] evaluated overhead using 30 websites, we extend this assessment to 100 websites based on the Alexa most visited website list [8] to obtain a more robust measurement of performance impact, ensuring our results capture a wider range of website behaviors and variations in load times. Our results are an average of over three independent runs for each website.

p	Average Loading Time (ms)	Overhead Percentage
0	1,848	0%
1	2,425	31%
2	2,435	32%
4	3,351	81%
6	3,537	91%
8	4,393	138%
10	5,015	171%
12	6,035	226%

Table 17: Performance overhead of FlipStress with a varying number of stressor instances (averaged over 3 runs).

As observed from Table 17, the baseline average loading time without the defense is approximately 1,848 ms. The overhead escalates with the number of cores utilized, reaching up to 6,035 ms, or 226%, when all physical cores are used. In other words, websites

OS	Browser	Resource	LR(%)	CNN(%)	LSTM(%)
Linux	Chrome	YouTube video	49 ± 7.68	56 ± 10.68	57.5 ± 11.01
	Chrome	Google Drive video	49 ± 5.25	55 ± 10.50	68.5 ± 8.90
	Firefox	YouTube video	45 ± 5.55	54 ± 9.81	65 ± 9.81
	Firefox	LinkedIn video	51.5 ± 5.50	65 ± 12.00	68 ± 11.00
	Tor	YouTube video	52 ± 5.69	51 ± 11.42	55.93 ± 7.80
Windows	Chrome	YouTube video	48 ± 7.88	56 ± 5.11	73 ± 7.76
	Firefox	YouTube video	51 ± 5.45	55.78 ± 5.65	70 ± 8.90
MacOS	Chrome	YouTube video	62 ± 5.57	70.5 ± 11.06	63.5 ± 7.43

Table 15: FlipStress Attack Accuracy Across Different System Configurations (system details provided in Appendix A).

load 3.26 times slower when our defense is in effect. Our defense allows users to balance security and performance by adjusting the number of instances, enabling a customizable trade-off.

Defense	Overhead Percentage
DefWeb [26]	63%
CacheShaping [16]	119%
FlipStress	226%

Table 18: Performance overhead of FlipStress compared with other existing defenses, when using 12 cores.

Table 18 compares the performance overhead of FlipStress with DefWeb and CacheShaping when utilizing all 12 physical cores in our system. For DefWeb, we use a random noise template while rendering each of the 100 websites. As anticipated, our defense incurs a higher performance overhead than other existing solutions. This additional overhead is due to the increased noise required to strengthen the defense's effectiveness — a necessary trade-off for enhanced security.

9 Discussion

Targeted deanonymization shares conceptual similarities with website fingerprinting, as both attacks exploit cache-based side channels to classify browsing activity. However, due to its binary classification nature, targeted deanonymization presents a unique challenge for defenses. In website fingerprinting, the attacker typically faces a multi-class problem involving distinguishing among 100 or more websites, where the complexity lies in differentiating among numerous classes. In contrast, targeted deanonymization simplifies the attacker's objective to distinguish between two classes — whether the user is the target or not — significantly reducing the complexity of the classification problem. This presents a more significant challenge for defenders because even slight variations in cache patterns can enable successful classification in targeted deanonymization, making it more difficult to thwart.

The difficulty of defending against targeted deanonymization is further amplified by the attackers' flexibility in creating the attack page. In contrast with website fingerprinting, where the attackers aim to identify public websites whose form and content are known to the defender ahead of time, in the targeted deanonymization scenario, attackers can use different leaky resources, dynamically load these resources at varying times, or alter their website's structure to generate diverse cache traces. This adaptability allows attackers to bypass deterministic defenses, such as DefWeb or CacheShaping, which rely on predefined templates or predictable patterns to obfuscate cache behavior.

The success of FlipStress stems from two key factors: its *randomness* and the *higher level of noise* it generates. First, FlipStress employs seven distinct stressors that switch randomly and rapidly. This randomness introduces significant variability in cache access patterns, making it challenging for attackers to identify consistent features for classification. Second, FlipStress generates higher noise levels, as reflected in its average buffer access time of *23.12 ms*, compared to *3.04 ms* for DefWeb and *10.03 ms* for CacheShaping while using the same number of instances or physical cores. This additional noise is necessary to dominate the attacker's measurements, ensuring that their data is influenced more by the defense mechanisms than the target's actual cache behavior. Together, these factors prevent accurate classification by machine learning models, reducing the effectiveness of targeted deanonymization and website fingerprinting attacks. Moreover, FlipStress's adaptability makes it more scalable across diverse attack scenarios, as it does not rely on specific assumptions about the attacker's behavior, the websites being visited, or the micro-architectural details of the victim's system.

As a noise-based defense, FlipStress still shares the fundamental limitations of other noise-based defenses: it only reduces the signal-to-noise ratio rather than completely eliminating the side-channel leakage. Thus, as our results show, the accuracy of the more advanced attack strategies we evaluated remains higher than the base rate, even though the application of FlipStress significantly reduces it. FlipStress also incurs computational overhead, particularly at higher intensity levels (e.g., $p = 12$), which could impact the user experience in resource-constrained environments. Future work could focus on optimizing the balance between noise generation and system overhead. One potential direction is to develop adaptive mechanisms that adjust noise levels based on real-time monitoring of system performance or detected attack signals. This approach would allow FlipStress to maintain high levels of protection while minimizing performance impacts.

10 Conclusion

In this paper, we introduced FlipStress, a novel defense mechanism against CPU cache-based side-channel attacks on the web, specifically targeted deanonymization and website fingerprinting.

By introducing noise injection through randomized stressors, FlipStress significantly reduces attack success rates, even in scenarios involving advanced adversarial strategies and machine learning models. Its implementation as a browser extension ensures ease of deployment across systems and user accessibility while offering tunable system overhead to balance security with performance.

References

[1] 2022. XSLeaks Summit 2022. https://tinyurl.com/xsleakssummit2022.

[2] 2023. XSLeaks Summit 2023. https://tinyurl.com/xsleakssummit2023.

[3] 2024. FlipStress Artifact Repository. https://anonymous.4open.science/r/FlipStress-566F/

[4] 2024. Scikit-learn: Machine Learning in Python. https://scikit-learn.org/

[5] 2024. TensorFlow: An end-to-end open source machine learning platform. https://www.tensorflow.org

[6] Colin King. [n. d.]. stress-ng. https://wiki.ubuntu.com/Kernel/Reference/stress-ng.

[7] Jack Cook, Jules Drean, Jonathan Behrens, and Mengjia Yan. 2022. There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 204–217. https://doi.org/10.1145/3470496.3527416

[8] ExpiredDomains.net. [n. d.]. Alexa Top Websites. https://www.expireddomains.net/alexa-top-websites/. Accessed: 2022-10-10.

[9] Michael Godfrey and Mohammad Zulkernine. 2013. A Server-Side Solution to Cache-Based Side-Channel Attacks in the Cloud. In *2013 IEEE Sixth International Conference on Cloud Computing*. 163–170. https://doi.org/10.1109/CLOUD.2013.21

[10] Michael Godfrey and Mohammad Zulkernine. 2014. Preventing cache-based side-channel attacks in a cloud environment. *IEEE Transactions on Cloud Computing* 2, 4 (2014), 395–408. https://doi.org/10.1109/TCC.2014.2358236

[11] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721* (San Sebastián, Spain) *(DIMVA 2016)*. Springer-Verlag, Berlin, Heidelberg, 279–299. https://doi.org/10.1007/978-3-319-40667-1_14

[12] Andrew Hintz. 2002. Fingerprinting websites using traffic analysis. In *International workshop on privacy enhancing technologies*. Springer, 171–178.

[13] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[14] Lukas Knittel, Christian Mainka, Marcus Niemietz, Dominik Trevor Noß, and Jörg Schwenk. 2021. XSinator.com: From a Formal Model to the Automatic Evaluation of Cross-Site Leaks in Web Browsers. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 1771–1788.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[16] Haipeng Li, Nan Niu, and Boyang Wang. 2022. Cache Shaping: An Effective Defense Against Cache-Based Website Fingerprinting. In *CODASPY 2022 - Proceedings of the 12th ACM Conference on Data and Application Security and Privacy*. https://doi.org/10.1145/3508398.3511500

[17] Tuo Li and Sri Parameswaran. 2022. FaSe: Fast Selective Flushing to Mitigate Contention-based Cache Timing Attacks. In *Proceedings - Design Automation Conference*. https://doi.org/10.1145/3489517.3530491

[18] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *Proceedings - IEEE Symposium on Security and Privacy*, Vol. 2015-July. https://doi.org/10.1109/SP.2015.43

[19] Yangdi Lyu and Prabhat Mishra. 2018. A Survey of Side-Channel Attacks on Caches and Countermeasures. *Journal of Hardware and Systems Security* 2, 1 (2018). https://doi.org/10.1007/s41635-017-0025-y

[20] Manifest-v3 2022. Replace blocking web request listeners. https://developer.chrome.com/docs/extensions/develop/migrate/blocking-web-requests.

[21] M. Asim Mukhtar, Maria Mushtaq, M. Khurram Bhatti, Vianney Lapotre, and Guy Gogniat. 2020. FLUSH + PREFETCH: A countermeasure against access-driven cache-based side-channel attacks. *Journal of Systems Architecture* 104 (2020). https://doi.org/10.1016/j.sysarc.2019.101698

[22] Yossi Oren. 2021. PP0 GitHub Repository. https://github.com/Yossioren/pp0.

[23] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology* (San Jose, CA) *(CT-RSA'06)*. Springer-Verlag, Berlin, Heidelberg, 1–20. https://doi.org/10.1007/11605805_1

[24] Google Research. 2024. Google Colaboratory. https://colab.research.google.com/

[25] Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In *NDSS*. The Internet Society.

[26] Son Seonghun, Dipta Debopriya Roy, and Gulmezoglu Berk. 2023. DefWeb: Defending User Privacy against Cache-based Website Fingerprinting Attacks with Intelligent Noise Injection. In *Proceedings of the 39th Annual Computer Security Applications Conference* (Austin, TX, USA) *(ACSAC '23)*. Association for Computing Machinery, New York, NY, USA, 379–393. https://doi.org/10.1145/3627106.3627191

[27] Anatoly Shusterman, Zohar Avraham, Eliezer Croitoru, Yarden Haskal, Lachlan Kang, Dvir Levi, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. 2021. Website Fingerprinting Through the Cache Occupancy Channel and its Real World Practicality. *IEEE Transactions on Dependable and Secure Computing* 18, 5 (2021), 2042–2060. https://doi.org/10.1109/TDSC.2020.2988369

[28] Anatoly Shusterman, Zohar Avraham, Eliezer Croitoru, Yarden Haskal, Lachlan Kang, Dvir Levi, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. 2021. Website Fingerprinting Through the Cache Occupancy Channel and its Real World Practicality. *IEEE Trans. Dependable Secur. Comput.* 18, 5 (2021), 2042–2060.

[29] Cristian Alexandru Staicu and Michael Pradel. 2019. Leaky images: Targeted privacy attacks in the web. In *Proceedings of the 28th USENIX Security Symposium*.

[30] Avinash Sudhodanan, Soheil Khodayari, and Juan Caballero. 2020. Cross-Origin State Inference (COSI) Attacks: Leaking Web Site States through XS-Leaks. In *Network and Distributed Systems Security (NDSS) Symposium*. https://doi.org/10.14722/ndss.2020.24278

[31] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Security Symposium*.

[32] YongBin Zhou and DengGuo Feng. 2005. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. https://eprint.iacr.org/2005/388

[33] Mojtaba Zaheri and Reza Curtmola. 2021. Leakuidator: Leaky Resource Attacks and Countermeasures. In *Proc. of the 17th EAI International Conference, SecureComm 2021*, Vol. 399. Springer, 143–163.

[34] Mojtaba Zaheri, Yossi Oren, and Reza Curtmola. 2022. Targeted Deanonymization via the Cache Side Channel: Attacks and Defenses. In *Proceedings of the 31st USENIX Security Symposium, Security 2022*.

[35] Yinqian Zhang and Michael K. Reiter. 2013. Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *Proceedings of the ACM Conference on Computer and Communications Security*. https://doi.org/10.1145/2508859.2516741

A Additional Experimental Details

System Information. Table 19 lists the information on the hardware, browsers, and operating systems used to evaluate our defense.

Property	**Details**
Windows/Linux System	
Device Model	Lenovo ThinkPad P14s Gen 3
Processor	12th Gen Intel(R) Core(TM) i7-1260P, 12 Cores
L1 Cache	448 KB L1i, 640 KB L1i
L2 Cache	9 MB
L3 Cache	18 MB
OS	Ubuntu 22.04.1 LTS / Windows 11 Pro
macOS System	
Device Model	Apple M3 Max
Processor	16-cores (12 P-cores, 4 E-cores)
L1 Cache	192 KB L1i, 128 KB L1d (P-cores) 128 KB L1i, 64 KB L1d (E-cores)
L2 Cache	32 MB (P-cores), 4 MB (E-Cores)
OS	macOS Sonoma 14.5
Browsers	
Google Chrome	124.0.6367.91
Firefox	132.0
Tor	14.0.3

Table 19: System Information

Hyperparameters	CNN	LSTM
Optimizer	Adam	Adam
Learning rate	0.001	0.001
Batch size	32	32
Activation function	relu	relu
Dropout	0.3	0.7
Pool size	3	4
Epoch	40	40
Kernel	32, 64, 128	32, 256, 256

Table 20: Hyperparameters for CNN and LSTM models

Machine Learning Classifier Parameters. The CNN and LSTM neural network model was used with the hyperparameters described in Table 20. The Logistic Regression classifier was used with 1000 max iterations.

B Description of Stressor Programs

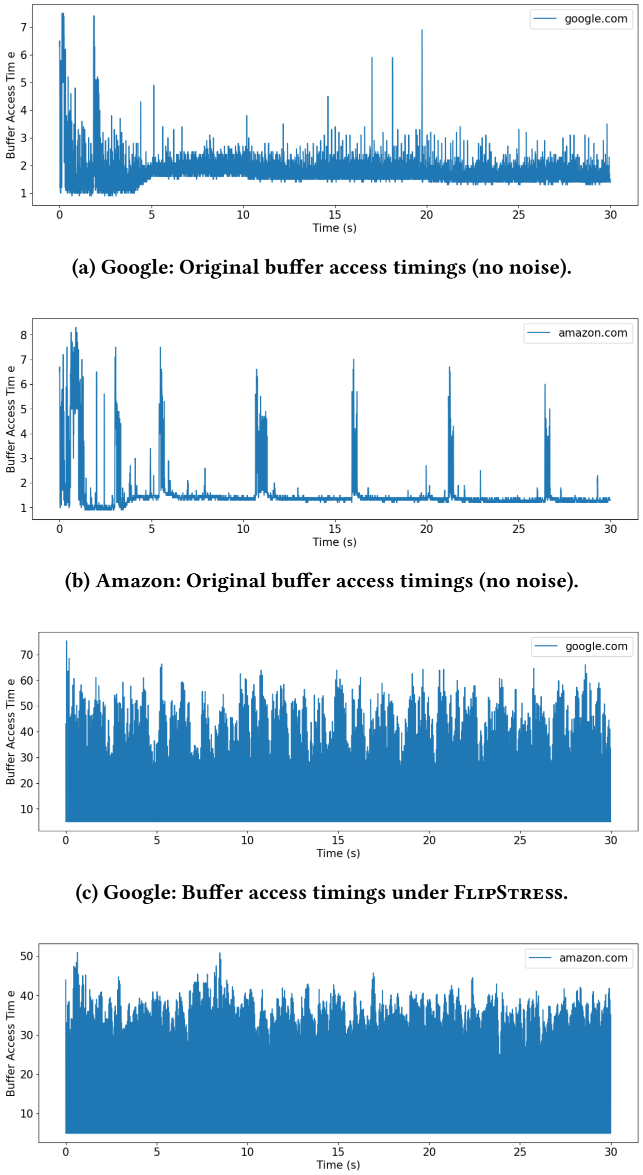
The seven stressors that were implemented as part of the FlipStress defense are:

(1) *Read Buffer*: This stressor repeatedly reads values from two large buffers, which are the size of the LLC buffer, filled with random data.
(2) *Write Buffer*: Similar to the Read Buffer, this stressor writes incremented values to two large buffers.
(3) *Read Linked List*: This stressor uses two large linked lists and traverses them to read data into a global variable.
(4) *Write Linked List*: This stressor also uses linked lists but writes incremented values to each node.
(5) *Stream*: This stressor implements a simplified version of the STREAM benchmark, performing a series of operations like copying, scaling, and adding values across large arrays, each the size of the LLC.
(6) *VM*: This stressor allocates a LLC-sized buffer and performs bitwise operations that systematically clear bits. The continuous manipulation of bits within the memory region places stress on the cache and memory.
(7) *Memcpy*: This stressor tests memory copying operations by implementing naive versions of memcpy and memmove. It continuously copies and moves data between large buffers, stressing the cache and memory.

C Additional Results for the FlipStress Defense Against the WF Attack

To illustrate the effectiveness of FlipStress in mitigating the website fingerprinting attack, we examine the buffer access timings of two representative websites: Google and Amazon. Figure 6 shows the original buffer access patterns and the patterns under FlipStress noise for both websites.

The original buffer access timings for Google (Figure 6a) and Amazon (Figure 6b) exhibit distinct patterns, with access times ranging from 1 to 8 ms. These unique fingerprints make these websites vulnerable to fingerprinting attacks. However, as seen in Figures 6c and 6d, FlipStress effectively introduces random high-frequency noise, elevating access times (10 to 70 ms) and obfuscating the original patterns. This obfuscation renders the patterns indistinguishable, thus significantly enhancing privacy and thwarting fingerprinting attacks.



(a) Google: Original buffer access timings (no noise).

(b) Amazon: Original buffer access timings (no noise).

(c) Google: Buffer access timings under FlipStress.

(d) Amazon: Buffer access timings under FlipStress.

Figure 6: Comparison of buffer access timings for Google and Amazon with and without FlipStress noise