LSBFlip

Brendon Derrick Ferrao[1] and Andrew Ioanoviciu[1]

[1]Rochester Institute of Technology

December 3, 2025

1 Introduction

Online privacy has become an increasingly prominent concern as users grow more aware of how extensively their data can be collected, analyzed, and monetized across the modern web. Recent surveys indicate that approximately 80% of internet users express concern about being tracked online and believe they have little meaningful control over how their personal information is used [1]. In response to rising public scrutiny and regulatory pressure, including the GDPR and CCPA, major web browsers have implemented significant restrictions on traditional tracking mechanisms, such as third-party cookies. For instance, Mozilla Firefox introduced Enhanced Tracking Protection in 2019, blocking known tracking cookies by default [2].

As modern browsers continue to expand in complexity, adding new multimedia capabilities, hardware interfaces, performance optimizations, and accessibility features, the surface area for unintended privacy leaks grows correspondingly. Each new API introduces the potential for entropy that can be used to differentiate users. As a result, browser fingerprinting has become one of the most resilient tracking strategies in an environment where traditional methods are increasingly constrained.

1.1 Background

Modern web browsers incorporate a wide range of mechanisms to improve performance and enable rich application functionality. Since these browser mechanisms store states they can be repurposed into tracking vectors, even if the mechanism was not originally intended to be used for tracking. Traditional tracking methods, such as cookies and local storage, are now heavily monitored and restricted by browsers [3]. As a result, websites have turned to other components of browsers to turn into trackers.

Some of these new tracking vectors include canvas fingerprinting, IndexedDB & WebSQL, cache-based vectors, network mechanisms, and client hints. The contents of this paper will focus entirely on canvas fingerprinting.

For the purposes of this research, several technical concepts are relevant. Hamming distance refers to the minimum number of substitutions requires to transform one string into another. It is commonly used to evaluate the difference between two binary outputs. The Structural Similarity Index Measure (SSIM) is a perceptual metric used to assess the similarity between two images by comparing luminance, contrast, and structure [4]. SSIM has become one of the most popular image quality metrics due to its ability to measure similarity perceived by the human eye rather than pixel by pixel [5]. Finally, the Least Significant Bit (LSB) in a binary value represents the lowest-order bit. Modifications to LSBs are frequently used in digital signal processing and steganography, including the perturbation methods examined in this paper.

1.2 Canvas Fingerprinting

Canvas fingerprinting is a modern tracking technique that generates a unique identifier for a user's device by exploiting subtle differences in how browsers render graphics. Unlike cookies or local storage, canvas fingerprinting does not save any data on the user's machine. Instead, it computes a hash based on the output of an HTML5 canvas element rendered in the browser. Because graphical rendering depends on numerous device and software-specific factors, including operating system, browser engine, GPU model, installed fonts, anti-aliasing behavior, and driver implementations, each device produces a slightly different rendering output, which results in a highly distinctive fingerprint [6]. This technique works even in private browsing modes and persists across sessions since the fingerprint is regenerated rather than stored [7]. Additionally, canvas fingerprinting requires no permission prompts, making it effectively invisible to most users and difficult to

avoid without specialized privacy tools.

Modern browsers have taken varied approaches to mitigating the practice. Privacy-focused browsers such as Brave employ "farbling," a method that randomizes fingerprintable values to reduce uniqueness [8]. Firefox includes a "Resist Fingerprinting" mode, which similarly introduces noise into canvas extraction, while Tor Browser either blocks or warns before a site attempts to read canvas data. Safari has also incorporated recent protections by injecting randomness to limit cross-site fingerprinting [9]. In contrast, Chrome currently provides no built-in fingerprinting resistance, leaving users dependent on extensions or external privacy tools.

Although canvas fingerprinting can be used constructively, such as for fraud detection where persistent device identifiers help detect abnormal login behavior. The same persistence and accuracy that make fingerprinting useful for security also make it attractive for invasive tracking practices. These include cross-site tracking without consent, building advertising profiles, bypassing cookie blockers or VPNs, and even deanonymizing users by correlating fingerprints across platforms [7]. As a result, canvas fingerprinting remains a controversial technique, balancing legitimate security use cases with substantial privacy risks.

2 Related Works

This section will go over the baseline for this project as well as what we contributed by working on this project.

2.1 Paper Overview

Recent research has increasingly focused on uncovering the hidden tracking vectors embedded within modern browsers, with one of the most notable contributions being *Navigating Murky Waters: Automated Browser Feature Testing for Uncovering Tracking Vectors*, published at NDSS. This work introduces *CanITrack*, a framework designed to evaluate how different browser features may be exploited for persistent tracking. CanITrack operates by using modular JavaScript-based read() and write() functions to determine whether specific browser mechanisms retain identifiers across sessions, making it possible to systematically assess tracking behaviors without relying on assumptions about how a mechanism is "supposed" to work.

The study tested 21 different browser features across 126 versions of seven major browsers, including Chrome, Firefox, Safari, Edge, Brave, Opera, and Tor. The results revealed that all major browsers exhibited some level of vulnerability, underscoring the widespread and ongoing challenges in preventing covert tracking. Notably, the researchers identified 13 mechanisms that support third-party tracking, as well as two mechanisms capable of bypassing private browsing protections, which emphasizes that even privacy modes cannot guarantee anonymity. The paper also provided early evidence that components of Google's Privacy Sandbox, particularly Private State Tokens and FLEDGE, can unintentionally introduce new tracking risks despite being promoted as privacy-preserving technologies.

Developed primarily by Mir Masood Ali and released on January 11, 2023, *CanITrack* is implemented largely in JavaScript, aligning with its goal of evaluating client-side browser behaviors in a realistic environment. The authors describe their tool as "an automated testing framework that evaluates browser mechanisms for potential misuse as tracking vectors," highlighting its relevance for both privacy researchers and browser developers seeking to identify and mitigate emerging threats. Overall, this work stands as one of the most comprehensive analyses of browser-level tracking vectors in recent years and provides an important foundation for ongoing research into fingerprinting defenses and privacy-preserving browser design.

2.2 Contribution

This project began with the initial goal of expanding the *CanITrack* framework by incorporating additional browser tracking vectors. However, as we explored the technical requirements in greater depth, it became clear that extending *CanITrack* was beyond our scope and current level of expertise. In response, we pivoted toward developing a more focused and achievable contribution. We chose to create a tool designed to counter canvas fingerprinting. This led to the development of *LSBFlip*, a proof-of-concept defense mechanism that introduces controlled noise into HTML5 canvases to disrupt fingerprint generation. Although we intended to implement *LSBFlip* as a fully functional browser extension, we encountered substantial challenges, most notably restrictive Content Security Policies (CSPs) that prevented our scripts from executing in many testing environments. Despite these limitations, our work remains closely tied to the *CanITrack* baseline and provides a feasible, time-bounded contribution that demonstrates the practicality of interfering with canvas-based tracking through lightweight client-side modifications.

3 Methodology

This section will go more in-depth into the code and how *LSBFlip* works. This starts from canvas generation, and then goes through flipping bits, generating the fingerprint hash, and a test harness that was created to automate testing.

3.1 Canvas Generation

To produce a stable yet highly discriminative fingerprint, we designed a custom HTML5 canvas drawing routine responsible for generating the base image used in all experiments (Figure 1). The goal of this step is to take advantage of the subtle rendering variations that naturally occur across different browsers, GPUs, operating systems, and font engines. Even when drawing the same shapes and text, these low-level differences, such as anti-aliasing artifacts, sub-pixel positioning, gradient interpolation, and font rasterization, introduce small but measurable inconsistencies. These inconsistencies form the entropy that fingerprinting scripts rely on.

Figure 1: Fingerprinting Canvas Example

Our implementation intentionally draws a visually complex scene, including gradients, Bézier curves, multiple font families, rotated text, and layered colored shapes. This mixture of geometric and typographic elements increases the likelihood that each device will render at least some components differently. This canvas output serves as the foundation for both the baseline fingerprints and the modified fingerprints produced by *LSBFlip*.

3.2 Bit Flipping

Once the canvas has been rendered, the script selectively alters the pixel buffer by flipping a configurable number of least significant bits (LSBs) in randomly chosen pixels. These controlled mutations cause significant changes in the final hash value while leaving the visual appearance effectively unchanged to the human eye.

Once the canvas is drawn, the script extracts the raw pixel data and determines how many pixels to alter based on a user-selected flip percentage. Using a random number generator, either non-deterministic like Math.random() or a deterministic seeded PRNG using Mulberry32 we select pixel indices and determine how many LSBs (1 to N) should be flipped per channel. For each chosen pixel, the algorithm toggles specific LSBs using XOR operations, producing subtle numerical changes that dramatically alter the final SHA-256 hash while remaining imperceptible to the human eye. The modified pixel array is then written back to the canvas, ensuring that the resulting fingerprint reflects the introduced noise. An optional visualization overlay highlights the altered pixels for demonstration but does not affect the computed fingerprint.

3.3 Hash Calculation

In order to generate a hash we must first convert the canvas into a string format. We do this by exporting the Canvas to an Identifying String by using the canvas.toDataURL() function. This converts the rendered pixel buffer into a deterministic Base64-encoded PNG string. Even a single-bit change in any pixel would result in a completely different hash. PNG encoding differences (e.g., metadata chunks, compression decisions) are also indirectly influenced by the pixel buffer, which amplifies platform differences.

The script hashes the Base64 string using hashStringToHex() function using SHA-256 to ensure a stable representation of rendering differences, minimizing collisions,and producing a consistent hex string suitable for comparison, A fallback (FNV-1a) is used where crypto.subtle is unavailable, preserving determinism.

3.4 Testing Harness

A test harness was created using *Node.js* to automate the *LSBFlip* testing. It systematically evaluates the behavior of *LSBFlip* over a wide range of parameter configurations in order to understand its impact on canvas fingerprinting. To accomplish this the script automates a browser environment using Puppeteer where it repeatedly renders the fingerprinting canvas under different flipping percentages (0%-100%) and LSB-flip depths (0-6 bits). For each configuration, the harness loads the test page, sets the *LSBFlip* controls, and captures the canvas output. Each canvas output is converted to a PNG buffer, hashed using SHA-256, and stores alongside other metadata. This enables the harness to quantify key fingerprinting metrics such as the number of unique hashes produced, the repetition rate, and the Hamming distance from a baseline hash. The script also generates visual artifacts, including individual sample images and averaged composite images, to help identify how the perturbations influence the perceptual appearance of the canvas.

Beyond hash-level comparisons, the test harness performs more detailed image-based analyses to evaluate the perceptual similarity and structural changes induces by *LSBFlip*. The test harness computes the mean pixel difference between each average canvas and a baseline image. The harness also incorporates SSIM to measure how the modified images resemble the unmodified canvas. For each test setting the script aggregates all of the computed metrics and writes them to a JSON and CSV file. This allows us to extract the information to visualize it. Overall, the test harness functions as an automated, reproducible framework that quantifies *LSBFlip*'s effects across hundreds of thousands of configurations. This enables the project to empirically assess how different flipping intensities influence fingerprint randomness and perceptual integrity.

4 Results

The results of over 350,000 test cases were compiled and visualized below.

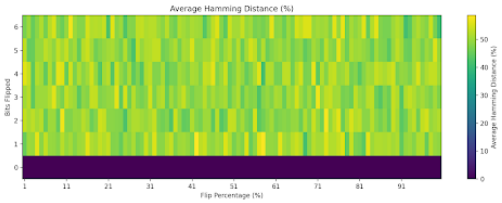

Figure 2: Average Hamming Distance for Flip Percentage vs LSBs Flipped

Figure 2 shows how the average Hamming distance changes as different percentages of LSB flips are applied across multiple bit positions. The row for zero flipped bits remains at zero, indicating no deviation from the baseline image. Once any LSB position from 1 to 6 is modified, however, the Hamming distance quickly rises and stays between roughly 40% and 55% across all flip percentages. This pattern suggests that even small levels of perturbation are enough to make the resulting hashes appear effectively random compared to the original. The absence of a clear gradient along the x-axis also indicates that the specific percentage of pixels altered matters less than the act of flipping a bit itself, reinforcing the idea that minimal noise is sufficient to disrupt canvas fingerprinting.

Figure 3 presents the mean pixel difference across varying flip percentages and bit positions. As expected, the row corresponding to zero flipped bits remains at zero, indicating that the baseline image is unchanged. For all other bit positions, the mean pixel difference increases

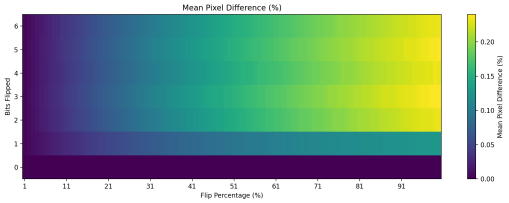

Figure 3: Mean Pixel Difference for Flip Percentage vs LSBs Flipped

smoothly as the flip percentage rises, forming a clear gradient from left to right. Higher bit positions generally produce larger differences, since flipping more significant bits results in larger changes to pixel values. Despite this upward trend, the overall magnitude of the pixel differences remains relatively small (generally staying below 0.25%), suggesting that the visual appearance of the canvas remains largely intact even when substantial percentages of LSBs are flipped. This pattern highlights how LSB perturbations can significantly disrupt fingerprinting signals while preserving the human-visible content of the image.

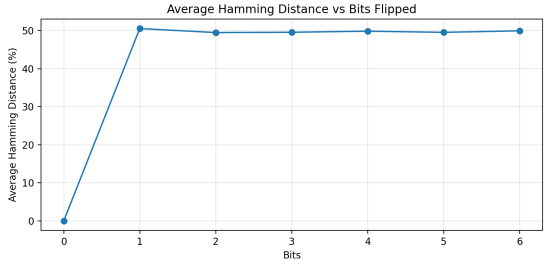

Figure 4: Average Hamming Distance vs LSBs Flipped

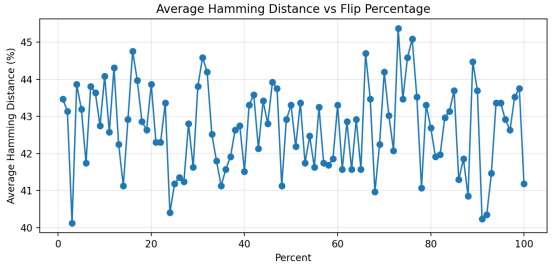

Figure 5: Average Hamming Distance vs Flip Percentage

Figure 4 plots the average Hamming distance for different numbers of LSBs flipped. Here, the distance jumps sharply once a single bit is flipped and then stabilizes near 50% for higher bit counts. This behavior is expected, since flipping even one LSB introduces substantial ran-

domness, and additional flipped bits do not significantly increase the overall disagreement. Figure 5 shows the average Hamming distance as the flip percentage increases. The values fluctuate around roughly 42–44%, with no strong upward or downward trend, suggesting that randomly flipping different proportions of LSBs tends to produce a fairly consistent level of bit disagreement. Together, these results indicate that LSB perturbations rapidly drive the fingerprint signal toward randomness, regardless of the precise flip percentage or number of bits modified.

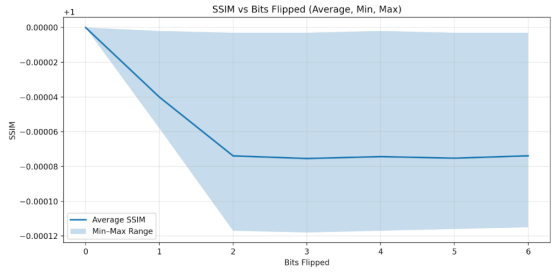

Figure 6: SSIM vs LSBs Flipped

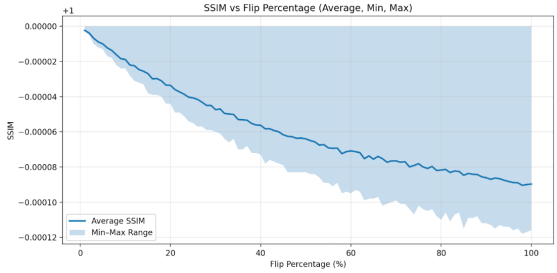

Figure 7: SSIM vs Flip Percentage

Figure 6 shows how SSIM changes as the number of flipped LSBs increases. SSIM drops sharply when moving from zero to two flipped bits, after which the decline plateaus and remains relatively stable through six bits. Although the min–max range broadens with additional flipped bits, the overall SSIM values stay close to zero, indicating that even multiple LSB flips only slightly impact perceptual image quality.

Figure 7 presents SSIM as a function of the percentage of flipped pixels. Here, SSIM decreases smoothly and almost linearly as the flip percentage rises from 0% to 100%. The widening min–max band suggests increasing variability across images at higher flip rates, but the average degradation remains modest. Taken together, these results show that substantial perturbation can be introduced while preserving perceptual quality. Based on all of the trends, we recommend flipping 74% of pixels with 1 LSB flipped each, which provides strong disruption of embedded signals while maintaining visually negligible distortion.

5 Discussion

This section will discuss the implications of *LSBFlip* as well as the limitations we came across while working on this project. Future steps, such as completion of the Chrome extension for *LSBFlip*, will also be discussed here.

5.1 Implications

The findings of this study highlight that browser fingerprinting remains one of the most effective and least visible forms of online tracking. Unlike cookies or other conventional mechanisms, fingerprinting techniques operate passively and often without the user's awareness, making them difficult to monitor or regulate. The proposed *LSBFlip* method demonstrates a promising countermeasure against canvas fingerprinting by introducing controlled randomness into the rendered output. By flipping bits within the HTML canvas, the technique disrupts the determinism on which fingerprint generation relies, resulting in hashes that are inconsistent and therefore unsuitable for long-term tracking. This suggests that lightweight, client-side perturbation methods may offer a viable path toward mitigating fingerprint stability without requiring major architectural changes to web browsers.

5.2 Limitations

Despite its potential, *LSBFlip* faces several practical limitations. First, implementing the technique in a deployable form, such as a browser extension, proved challenging due to the restrictive nature of modern Content Security Policies (CSPs). Many websites tightly control the execution of inline scripts or modifications to rendering surfaces, which limits the feasibility of injecting the necessary code at runtime. This constraint significantly reduced the ability to prototype a functional extension within the project's timeframe. Additionally, because techniques like LSB flipping directly alter canvas output, they may inadvertently break certain website functionalities. Finally, user awareness of fingerprinting countermeasures remains low, meaning that even effective tools may face adoption challenges if they introduce noticeable side effects or require technical configuration.

5.3 Future Work

Future work on *LSBFlip* should focus primarily on developing a fully functional browser extension, as this would allow the technique to operate seamlessly for end users. While initial attempts to build such an extension were hindered by strict Content Security Policies (CSPs), further exploration of alternative injection strategies or browser-specific extension APIs may help bypass these restrictions without violating security requirements. Achieving a stable extension framework would also enable the integration of broader tracking protections. For example, *LSBFlip* could be expanded to block or perturb additional vectors such as cookies and local storage, IndexedDB, cache-based fingerprints, network-level identifiers, new browser APIs, and Client Hints. Incorporating these features would allow the tool to serve as a more comprehensive fingerprinting defense rather than targeting canvas-based tracking alone.

Another important direction involves enhancing the realism of the noise introduced into the canvas. In its current form, the randomized output may appear synthetic and potentially suspicious to fingerprinting scripts or browser integrity checks. Future research should explore generating "hardware-plausible noise" that mimics the subtle imperfections found in real GPUs, creating fingerprints that appear natural rather than artificially manipulated. This would improve both the stealth and effectiveness of the technique. Combined, these advancements would bring *LSBFlip* closer to a deployable, user-facing tool capable of mitigating multiple forms of modern web tracking.

6 Conclusion

Browser fingerprinting continues to pose a substantial threat to user privacy as traditional tracking mechanisms become increasingly restricted. This project set out to evaluate and disrupt one of the most resilient fingerprinting vectors, Canvas Fingerprinting through the development of *LSBFlip*, a lightweight perturbation-based defense. Our experimental results, drawn from more than 350,000 test cases, demonstrate that even minimal manipulation of least significant bits introduces enough randomness to invalidate stable fingerprint generation while being indistinguishable from the original image. These findings highlight a critical insight: fingerprinting defenses do not require heavy obfuscation or complex system-level controls; small, targeted modifications are sufficient to break determinism without degrading user experience.

While *LSBFlip* proved effective in controlled environments, real-world deployment poses nontrivial challenges. Modern Content Security Policies restricted our ability to implement the technique as a fully functional browser extension, and certain websites may rely on precise canvas output for legitimate purposes. These limitations highlight the gap between theoretical fingerprint resistance and deployable privacy tools. Nonetheless, this work provides a concrete foundation for further exploration into client-side fingerprint obfuscation techniques and reiterates the need for built-in browser support to counter tracking mechanisms that operate without user consent.

With continued refinement particularly in extension development,and stealthier noise modeling perturbation-based defenses may become a viable component of a broader strategy to protect users from covert cross-site tracking. This work contributes a measurable step toward that goal and reinforces the importance of ongoing research into privacy-preserving browser technologies.

References

[1] Brooke Auxier, Lee Rainie, Monica Anderson, Andrew Perrin, Madhu Kumar, and Erica Turner. Americans and privacy: Concerned, confused and feeling lack of control over their personal information, 11 2019.

[2] Dave Camp. Firefox now available with enhanced tracking protection by default plus updates to facebook container, firefox monitor and lockwise | the mozilla blog, 06 2019.

[3] Tom Kemp. Understanding website tracking and how to limit it | privacy.ca.gov, 07 2025.

[4] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612, 04 2004.

[5] Illya Bakurov, Marco Buzzelli, Raimondo Schettini, Mauro Castelli, and Leonardo Vanneschi. Structural similarity index (ssim) revisited: A data-driven approach. *Expert Systems with Applications*, 189:116087, 03 2022.

[6] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. Browser fingerprinting: A survey, 11 2019.

[7] Kaitong Lin, Huazhu Cao, and Amin Milani Fard. Browser fingerprint detection and anti-tracking, 2025.

[8] Brave. Fingerprinting defenses 2.0, 05 2020.

[9] Apple. Safari privacy overview learn how the safari web browser protects your privacy, 11 2019.