MAJĀL

2026 VOLUME 5
UNDERGRADUATE JOURNAL

How Browsers Reveal Themselves: Fingerprinting via the Web Audio API

Sham Homssi,[a] Sudais Arif,[b] and Lujain AlMulaifi[c]

[a] Carnegie Mellon University Qatar, Undergraduate, Computer Science

[b] Carnegie Mellon University Qatar, Undergraduate, Computer Science

[c] Carnegie Mellon University Qatar, Undergraduate, Information Systems

Plain Language Summary

Web browsers can reveal small differences in how they process information, and these differences can be used to create a "fingerprint" that helps distinguish one browser or device from another. This raises privacy concerns because fingerprinting can potentially be used to track people online without relying on cookies or asking for permission. In this study, we tested whether the Web Audio API, a browser feature normally used to generate and process sound, could produce reliable fingerprints across different browsers and devices. We found that simple audio signals, particularly sine and square waves, produced highly consistent and distinctive fingerprints in our experiments, while more complex combinations of frequencies were less reliable. These results show that even ordinary browser features can reveal information about a user's device, highlighting the need to consider fingerprinting risks when designing web technologies and privacy protections.

Abstract

Modern browsers expose rich audio processing capabilities through the Web Audio API as it enables dynamic and immersive sound experiences. However, the tight integration of the API with hardware-level rendering introduces subtle platform-specific variations. These differences can be exploited for browser fingerprinting, an emerging privacy concern in which users are uniquely identified without their knowledge or consent. In this paper, we investigate the feasibility of audio-based fingerprinting by generating controlled frequency signals, analyzing time-domain waveforms via the Web Audio API, and applying hash-based comparisons to assess fingerprint consistency and uniqueness. We design and implement a browser-based pipeline to extract distinguishing features from system-level audio behavior. Our results show that even simple waveform configurations can result in stable and unique identifiers, with browser-specific rendering playing a critical role in fingerprinting effectiveness. These findings call for a re-examination of the privacy implications of seemingly benign web technologies.

Publication Category

Course-based project

Academic Context

This paper developed from an experimental project completed for a Browser Security course. Students were asked to identify a browser security topic, formulate a research question, and investigate it through practical testing. Our group focused on browser fingerprinting through the Web Audio API, building and running experiments across different browsers and devices to examine the stability and uniqueness of the resulting fingerprints. The project was then developed into a formal research paper presenting the motivation, experimental approach, findings, and privacy implications of the study.

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0

Introduction

Modern browsers, such as Google Chrome, Safari, and Opera, expose a rich collection of data consisting of audio processing capabilities through the Web Audio API. A web API (Application Programming Interface) is an interface that enables communication over HTTP or HTTPS, providing protocols for interactions between software applications on the web (Catchpoint, n.d.). The Web Audio API is a high-level JavaScript interface that allows the developer to generate, control, and process audio directly in the browser, enabling dynamic and immersive sound experiences. However, the tight integration of the API with hardware-level rendering, introduces subtle platform specific variations. These differences can be exploited for browser fingerprinting, which is an emerging privacy concern where users are uniquely identified without their knowledge or consent.

Prior work shows browser audio processing can contribute to fingerprinting by producing device-dependent outputs during signal generation and analysis (Englehardt & Narayanan, 2016; Queiroz & Feitosa, 2019). Because these outputs reflect differences in browser implementation and underlying audio hardware, they can serve as a supplementary identification signal. Unlike many purely software-based attributes, such hardware-influenced characteristics are generally harder to spoof, which makes them relevant in the study of browser fingerprinting.

These characteristics raise the question of how much entropy (i.e. randomness and unpredictability) and stability can be extracted from seemingly simple audio signals. Although prior studies have demonstrated the feasibility of Web Audio fingerprinting, they often treat the pipeline as a black box, focusing on the outcome rather than the internal process. Our approach aims to dissect and analyze the impact of different waveform types and configurations on fingerprint reliability, providing a more fine-grained understanding of which features contribute to repeatable and distinct fingerprints.

Additionally, the growing sophistication of anti-tracking tools makes it important to investigate whether Web Audio-based fingerprints can persist under realistic, user-level conditions. By studying common waveforms such as sine, square, triangle, and sawtooth, we aim to highlight which signals are most effective under real-world constraints. Our findings could help inform the development of privacy tools and browser design standards in the future.

In this paper, we explore the viability of audio-based fingerprinting using the Web Audio API. We developed a browser-based experiment using HTML and JavaScript to synthesize waveform signals and analyze them across multiple browsers. Our goal is to assess fingerprint quality using two primary metrics: stability (consistency of fingerprint across runs) and uniqueness (ability to distinguish users).

Background

To understand how the Web Audio API can be exploited for device fingerprinting, it is essential to examine the technical foundations and architectural design that enable both its powerful audio processing capabilities and the privacy vulnerabilities that emerge from its hardware interactions.

Defined by the World Wide Web Consortium (W3C) specification, the Web Audio API has a dual-threaded architecture consisting of a control thread and a render thread. The control thread runs the JavaScript code of the web page, interacts with the API, and manages user-facing changes to the audio graph, while the render thread performs low level audio processing. The separation of this architecture ensures that JavaScript code responding to user actions does not interrupt real-time audio from playing smoothly.

This dual-threaded architecture is not unique to this specific project, but rather, represents the fundamental architecture requirements of the Web Audio API as a whole. All web audio implementations must separate graph configuration (control thread) from real-time audio rendering (render thread) to ensure the safe and predictable audio processing (Matuszewski & Rottier, 2023). Thus, while there are varying processors and implementation details, the two-threaded architecture model is a universal pattern across all web audio applications.

Running within the web page's JavaScript environment, the control thread is responsible for the high-level audio components, including AudioContext, and the user application space where the audio graph is built and configured (Matuszewski & Rottier, 2023). As part of the high-level audio components, the OscillatorNode, a type of AudioScheduledSourceNode, defines the periodic waveform by specifying its parameters (i.e. frequency, detune, and type) (MDN Web Docs, 2024). It's used by developers to create constant tones and is often the starting point of

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0

an audio signal chain (MDN Web Docs, 2024). The OscillatorNode's output is then routed through the GainNode, which is typically the next component and defines the volume of the generated tone. This audio processing module applies a gain factor to its input signal to adjust its amplitude (MDN Web Docs, 2023). After modulating the output level, it is routed to the last stage of an audio graph, the DestinationNode. This node does not process audio directly, but rather it represents the output device or recording context, serving as the endpoint where the fully rendered audio is delivered to the user's system (MDN Web Docs, 2025).

Once the graph is defined by the control thread, it is transmitted via the message bus to the render thread. In this constrained environment, the corresponding audio processors (i.e., the OscillatorProcessor, GainProcessor, and DestinationMixer) execute the low-level mathematical computations, manage the AudioGraph, and run the render loop that processes audio frames (Matuszewski & Rottier, 2023). The OscillatorProcessor generates the actual waveform samples using the parameters previously defined by the OscillatorNode. The GainProcessor applies the dynamic volume adjustments using the gain parameters defined by the GainNode and performing per-sample mathematical computations (MDN Web Docs, 2023). Lastly, on the render thread, the DestinationMixer is the final stage in the audio processing pipeline, combining all processed audio streams before passing them to the hardware backend (Matuszewski & Rottier, 2023).

While the Web Audio API defines the structure of the audio graph, the actual sound generation and numerical computation depends solely on the physical hardware of the user's device. Once the audio reaches the render thread, the processing gets carried out by the system's audio backend (i.e., Core Audio on macOS, WASAPI on Windows). Because the audio content depends on the device's sound card, the output reflects hardware characteristics (Matuszewski & Rottier, 2023). Even devices with the same model and version exhibit manufacturing inconsistencies in their audio hardware which results in variations in rendered signals.

The variations which enable fingerprinting happen during the process of rendering the audio such as differences in amplitude values produced during waveform rendering (GainProcessor), making them suitable for device fingerprinting (Matuszewski & Rottier, 2023).

The deep interaction between the WebAudio API and the device's hardware creates subtitles and device-specific variations in audio signal rendering. They can be employed to steal a user's privacy on the web by collecting relevant features of a user's device. Recent research has shown the effects of generating audio signals and utilizing this API to track users with signal processing (Englehardt & Narayanan, 2016). Unlike traditional software-based fingerprinting, hardware-dependent characteristics such as these are harder to spoof, making it a promising tool for device identification (Queiroz & Feitosa, 2019). These findings highlight the potential of the Web Audio API as a vector for fingerprinting and a more persistent tracking measure. Through the evolution of browser security, recognizing and mitigating such high-level mechanisms has become essential when it comes to efforts to preserve user privacy across the Web.

Research Question: Can a stable and repeatable audio fingerprint be generated using the Web Audio API and SHA256 hashing of Captured audio data?

Related Work

Research on Web Audio fingerprinting was first documented in detail by Englehardt and Narayanan (2016) in their large-scale study of online tracking. Their analysis showed that certain websites employed AudioContext operations, including OscillatorNode and DynamicsCompressorNode, to derive numerical outputs that varied across devices. Although audio was only one of several fingerprinting vectors they measured, the study showed that websites were actively invoking Web Audio-related operations to derive numerical outputs that could contribute to browser fingerprints. This made clear that Web Audio was already being used in practice as part of broader fingerprinting pipelines.

A more focused investigation was carried out by Queiroz and Feitosa (2019), who proposed one of the earliest dedicated Web Audio fingerprinting methods. Their work constructs Web Audio graphs using nodes such as OscillatorNode, GainNode, AudioDestinationNode, and AnalyserNode, and systematically compares two interfaces: AudioContext and OfflineAudioContext. Through a series of controlled experiments, they show that real-time pipelines based on AudioContext tend to be unstable across repeated runs, whereas OfflineAudioContext yields stable sets of samples for periodic waves (sine, square, triangle, and a custom waveform). In a deployment with

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0

over one hundred participants, their method was able to distinguish classes of devices (combinations of browser, operating system, and device type), but only a small number of devices received unique fingerprints. They conclude that while Web Audio exposes hardware-related variation, it is not sufficient on its own and is best used in combination with other fingerprinting techniques.

The most extensive modern evaluation appears in the IMC 2022 paper "Your Speaker or My Snooper?" (Chalise et al., 2022). This study collected Web Audio fingerprints from approximately 2,093 participants and evaluated seven different audio-based fingerprinting vectors. The authors reported three major findings. First, audio fingerprints can be inconsistent: some browsers produce more than one fingerprint on the same device, a phenomenon they termed "fickleness." Second, only 95 distinct fingerprints were observed in the entire dataset, indicating limited diversity relative to the number of users. Third, when combined with existing vectors such as Canvas, audio contributed an average entropy increase of about 9.6%. To address fingerprint inconsistency, the authors introduced a graph-based matching approach that allowed multiple related fingerprints to be clustered into a stable identifier.

Additional large-scale research supports the idea that audio is primarily effective when combined with other browser features rather than used independently. FP-Tracer (2024), a fine-grained fingerprinting detection study applied to more than 80,000 domains, confirms that AudioContext-related attributes are actively collected on the web and appear in higher-entropy fingerprinting vectors (Boussaha et al., 2024). While FP-Tracer does not compute dedicated entropy estimates for audio alone, its findings show that audio attributes contribute to complex, multi-attribute fingerprints alongside canvas, storage, and navigator properties. Together with the entropy-focused results of IMC 2022, these observations reinforce that audio is best understood as a supplementary vector that increases the distinctiveness of composite fingerprints rather than as a strong standalone identifier.

Recent work presented at the Web Conference (WWW) 2023 further broadens the understanding of browser fingerprinting by discovering previously unknown APIs used in fingerprinting scripts (Su & Kapravelos, 2023). Although the focus of the study is not specifically on Web Audio, the system identifies several audio-related interfaces among its expanded list of 231 fingerprinting APIs. The key contribution of this work is methodological: it introduces an automated hybrid analysis pipeline that reveals new fingerprinting vectors created by emerging browser features, demonstrating that the fingerprinting surface continues to grow as modern APIs evolve.

Relationship to our Work

Our study builds on these foundational results but differs in scope and methodology. Although audio was only one of several fingerprinting vectors they measured, the study showed that websites were actively invoking Web Audio-related operations to derive numerical outputs that could contribute to browser fingerprints. This made clear that Web Audio was already being used in practice as part of broader fingerprinting pipelines. However, our focus is distinct in three ways. First, we perform a detailed comparison of four waveform types, sine, square, triangle, and sawtooth, whereas earlier studies typically use sine waves or do not examine waveform choice in depth. Second, we conduct our evaluation on recent versions of major browsers (Chrome 138, Firefox 140, Safari 18.5), which incorporate newer audio engines and updated processing behavior. Third, we introduce a composite multi-frequency signal (440 Hz, 880 Hz, 1760 Hz) to examine whether increasing spectral complexity improves fingerprint uniqueness. Our results show that this approach can reduce stability, contrasting with assumptions in earlier work such as IMC 2022.

In sum, while prior work primarily evaluates Web Audio fingerprinting at large scale or treats the audio pipeline as a single monolithic vector, our contribution lies in isolating and comparing the fingerprinting properties of specific waveform configurations on modern browser engines. This offers a finer grained understanding of how individual signal characteristics influence stability and uniqueness.

Methods

Experimental Design

We implemented a browser-based experiment in HTML and JavaScript to generate a fingerprint for each browser/audio-device combination. During the test runs, we synthesized four waveform types (sine, square, triangle, and sawtooth) at 440 Hz and captured their time-domain representations, which is the shape of each audio signal expressed as amplitude values sampled over time, using the Web Audio API. Specifically, the waveform was collected through an AnalyserNode using getByteTimeDomainData(), which outputs the rendered signal as a byte array. This array was

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0

then preprocessed and hashed using SHA-256 to produce a fingerprint. The process was repeated 50 times for each configuration to evaluate fingerprint stability and uniqueness. All measurements in this study used AudioContext for real-time rendering, rather than OfflineAudioContext. While OfflineAudioContext renders audio to an in-memory buffer in a fully deterministic, clock-independent manner, AudioContext processes audio in real time and is therefore subject to system scheduling and audio driver timing. Prior work by Queiroz and Feitosa (2019) found that AudioContext-based pipelines tend to produce unstable fingerprints for this reason. In our implementation, however, perfect stability was achieved across all single-frequency trials. We attribute this to the noise-reduction preprocessing applied in step 3, rounding and scaling each byte value before hashing, which absorbs minor system-level timing variations before they can affect the final hash. This result should therefore be interpreted as specific to our controlled pipeline rather than as a general property of AudioContext-based fingerprinting.

Fingerprinting Process

The steps to creating these fingerprints are as follows:

1) Audio generation: For each waveform, an OscillatorNode, which is a Web Audio API node that generates periodic waveforms (sine, square, triangle, sawtooth) at a configurable frequency, produces a 440 Hz tone for approximately 0.2 seconds.

2) Frequency extraction: The AnalyserNode is configured with an FFT size of 2048. The Fast Fourier Transform (FFT) is an algorithm that decomposes a time-domain signal into its constituent frequency components; the FFT size determines both the frequency resolution and the length of the time-domain capture buffer (Dewesoft, n.d.). Each resulting frequency bin represents a discrete range of the frequency spectrum. In the single-frequency pipeline, however, no FFT transformation is applied to the captured data, getByteTimeDomainData() is called instead, which returns 2048 time-domain amplitude samples directly as unsigned 8-bit integers (values 0–255), representing the instantaneous shape of the waveform over time. The FFT size parameter is used here solely to set the buffer length.

3) Extraction: The waveform buffer is extracted and each byte value is rounded and scaled to reduce the sensitivity to system-level noise.

4) Hashing: The combined array of values is then hashed using SHA-256 which provides cryptographic hashing functionalities.

5) Repetition: The fingerprinting process is repeated 50 times for each waveform type. Each hash is logged to analyze consistency and variation of generated fingerprints.

To clarify the complete data processing flow: the fingerprint is derived directly from the time-domain byte buffer captured by getByteTimeDomainData(), not from FFT-bin magnitudes. Although the AnalyserNode is configured with an fftSize of 2048, which sets the capture buffer length, no frequency-domain transformation is applied to this buffer in the single-frequency pipeline. The multi-frequency pipeline, by contrast, uses getFloatFrequencyData() to extract FFT-bin magnitudes around target frequencies before hashing. Algorithm 1 summarizes the full processing flow for the single-frequency case (Table 1).

Mathematical Framework

1) Frequency-to-Bin Mapping: If a target frequency f is mapped to an FFT representation with size N and sample rate f_s, the corresponding bin index is computed as

$$bin\ index\ = \frac{f\ \cdot\ N}{f_s}$$

This formula identifies the FFT bin associated with a target frequency, adapted from Center for Computer Research in Music and Acoustics (n.d.).

2) Uniqueness of a Fingerprinting System: The uniqueness metric for evaluating fingerprinting effectiveness is calculated as:

$$\text{Uniqueness}\quad = 1 - \frac{u-1}{U-1}$$

u = Users with the same hash
U = Total users
Note: u = users with the same hash; U = total users.
Adapted from Queiroz and Feitosa (2019).

Environmental Setup

- Environment: Modern web browsers — Google Chrome 138.0 (stable, released June 24 2025); Mozilla Firefox 140.0 (stable, released June 24 2025) ; Safari 18.5 (build 20621.2.5, released May 12 2025)
- Audio Sample Rate: 48000 Hz
- FFT Size: 2048
- Frequencies: 440 Hz

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0

TABLE 1. Algorithm 1. Single-Frequency Audio Fingerprinting Pipeline.

Algorithm 1: Single-Frequency Audio Fingerprinting Pipeline

Input: waveType $\in$ {sine, square, triangle, sawtooth}, frequency = 440 Hz, trials = 25

Output: SHA-256 fingerprint hash per trial

1. Create AudioContext *(real-time rendering)*
2. Create OscillatorNode (type = waveType, frequency = 440 Hz)
3. Create GainNode (gain = 0.8)
4. Create AnalyserNode (fftSize = 2048)
   // fftSize sets the buffer length only; // no FFT transformation is applied
5. **Connect: OscillatorNode → GainNode → AnalyserNode → AudioContext.destination**
6. Start oscillator at context.currentTime + 0.05
7. Stop oscillator after 0.2 seconds
8. Wait 250 ms for waveform to stabilize
9. Call AnalyserNode.getByteTimeDomainData (buffer)
   // Returns 2048 unsigned 8-bit integers [0–255]
   // representing instantaneous amplitude over time
10. for each byte *b* in buffer: *b′* ← **round**(*b*) and scale to reduce noise sensitivity
11. Concatenate all *b′* values into a single array
12. **Compute hash ← SHA-256(array)**
13. Log hash; repeat steps 1–12 for each trial
14. **Compute stability ← (count of most frequent hash) / trials**

- Sampling Window: ±10 bins around each frequency
- Number of Trials: 50 runs per test
- Waveform Types: Sine, Square, Triangle, Sawtooth
- Format: CSV format

Results & Discussion

To evaluate the effectiveness of audio-based fingerprinting through the use of the Web Audio API, we conducted a series of tests and experiments across multiple waveforms and frequency strategies. This section presents the findings from both the single-frequency waveform tests and a secondary multi-frequency sine test. Our analysis focuses on two key fingerprinting attributes: stability (consistency of fingerprints across trials) and uniqueness (ability to differentiate browser–hardware combinations).

Although we collected data from multiple users, the primary source of fingerprint variation is the browser and hardware configuration, not the person. Users were included to provide diversity across devices, operating systems, and browser builds, but user identity itself does not influence the fingerprint. All reported differences therefore reflect differences in machine-level audio rendering pipelines.

TABLE 2. Single-Frequency Waveform Fingerprinting Results.

Waveform	Avg Stability	Avg Improved Uniqueness	Unique Hashes	Total Tests
Sine	1.00	0.875	12	600
Square	1.00	0.917	12	600
Triangle	1.00	0.500	12	600
Sawtooth	1.00	0.667	12	600

Single Frequency Waveform Results

The first set of tests evaluated fingerprints generated using four distinct waveform types: sine, square, triangle, and sawtooth. Each waveform was tested on multiple browsers and hardware configurations. Each browser–hardware–waveform combination underwent 50 identical fingerprinting runs.

Uniqueness was calculated using the following normalization metric:

$$\text{Uniqueness:} \quad 1 - \frac{u-1}{U-1}$$

where u = number of configurations producing the same hash U = total configurations Adapted from Queiroz and Feitosa (2019).

Stability was defined as the proportion of trials producing the most common hash for that configuration:

$$\text{Stability} = \frac{\text{Most frequent hash count}}{50}$$

A score of 1.00 therefore indicates that all 50 runs produced identical fingerprints.

From these results, we observed that square and sine waves offered the highest uniqueness values (around 0.9), making them strong candidates for browser and device identification. Triangle had notably low uniqueness (0.5),

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0

indicating frequent collisions across configurations. Sawtooth waves performed moderately with mid-range uniqueness (0.667). All four waveforms achieved perfect stability in our experimental setting, suggesting that the rendering pipeline used here was highly consistent under controlled conditions. Because our implementation used AudioContext, this result should be interpreted as a property of our setup rather than as a universal characteristic of all Web Audio fingerprinting pipelines.

Multi-Frequency Composite Sine Fingerprinting

To explore whether a richer spectrum could yield more discriminative fingerprints, we conducted a second experiment using composite sine tones at 440Hz, 880Hz, and 1760Hz. These three tones were summed into a single multi-frequency signal and processed using the same audio rendering and FFT hashing pipeline as in the single-frequency tests. Importantly, this experiment used the exact same browser and hardware configurations as the single-frequency experiment, ensuring direct comparability.

TABLE 3. Multi-Frequency Sine Fingerprinting Results

Browser	Avg Stability	Avg Improved Uniqueness	Unique Hashes	Total Tests
Chrome	0.78	0.61	9	50
Firefox	0.74	0.55	8	50
Safari	0.76	0.59	9	50

Unlike the deterministic behavior of the single-frequency signals, the composite sine test produced nondeterministic results where stability varied across different browsers. Each configuration again underwent 50 trials.

Compared to the single-frequency experiment, both stability and uniqueness dropped across all browsers. Despite containing more spectral information, composite tones did not translate to more reliable or more unique fingerprints.

This partially contradicts previous results [3], which suggest that broader signals improve entropy. We hypothesize three contributing factors: (1) modern browsers increasingly apply DSP normalization, limiting device-specific variation; (2) multi-frequency signals accumulate floating-point rounding noise, which reduces stability; and (3) post-2020 audio engine updates prioritize consistency across platforms, suppressing fine-grained spectral differences that earlier studies relied on.

Cross-Experiment Analysis

Taken together, our findings show that while the Web Audio API can produce stable and unique fingerprints under controlled settings, effectiveness varies significantly depending on the waveform and browser used. Simpler signals such as sine and square waves consistently achieved high uniqueness and perfect stability (Table 2), whereas composite multi-frequency signals showed degraded stability and lower uniqueness (Table 3). This highlights a trade-off between signal complexity and fingerprint reproducibility, and points to the dominant role of browser-specific audio rendering pipelines.

- Simplicity outperformed complexity: Isolated square and sine waves yielded the best trade-off between entropy and stability.
- Browser effects dominate: Fingerprints reflect browser–hardware characteristics, not user identity.
- Repeatability is fragile under complexity: Multi-frequency audio introduces nondeterministic floating point behavior that degrades stability.

Choice of SHA-256 Hashing

We apply SHA-256 to the FFT magnitude vector to generate the final fingerprint. This choice is motivated by practicality: raw FFT vectors are high-dimensional floating-point arrays that require thresholding, alignment, and similarity metrics to compare reliably, whereas SHA-256 provides a compact and architecture-independent identifier that is trivial to store and compare across trials. While SHA-256 exhibits strong avalanche behavior and small numerical changes can produce completely different outputs, the Web Audio pipeline is sufficiently deterministic for single-frequency signals that this does not reduce stability. Instead, the sensitivity of SHA-256 amplifies consistent device and browser-specific spectral differences, improving uniqueness without harming repeatability.

The trade-off is that hashing discards fine-grained similarity information; however, because our goal is to evaluate stability and collision behavior rather than degrees of spectral similarity, a cryptographic hash provides a clean and standardized way to summarize the FFT output.

Conclusion

This study explored the viability of audio-based browser fingerprinting using the Web Audio API and analyzed both standard waveform signals and complex composite tones.

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0

Through systematic testing across multiple browsers and users, we evaluated fingerprint quality using two key metrics: stability and uniqueness.

Our findings demonstrated that simple waveform signals, particularly sine and square waves, produced perfect stability and high uniqueness, making them effective for fingerprinting. In contrast, triangle and sawtooth waves exhibited more frequent hash collisions, suggesting that the waveform shape has a notable effect on fingerprint entropy. These results align with previous work but offer a more detailed breakdown of waveform-specific behavior.

Interestingly, our introduction of multi-frequency composite sine signals, intended to increase entropy, produced mixed results. Although this method preserved a moderate degree of uniqueness, it introduced non-determinism into the fingerprinting pipeline, reducing overall stability. This indicates that signal complexity does not necessarily yield stronger fingerprints and may, in fact, amplify sensitivity to rendering inconsistencies in browser audio engines.

Looking ahead, we suggest three potential directions for future research. First, future work could investigate multimodal fingerprinting by combining the Web Audio API with other APIs such as the Canvas API and font detection via CSS. This approach would allow researchers to target multiple subsystems within the user device and potentially increase fingerprint entropy, raising additional privacy concerns. Second, a comprehensive privacy impact study could evaluate how API combinations affect user identification and resistance to anti-fingerprinting techniques. As most browser APIs were not designed with fingerprinting in mind, their hardware-level exposure presents a need for a unified framework to govern API behavior and establish fingerprint-ability boundaries.

Finally, another promising direction is to explore the use of physical hardware, specifically the interaction between speaker input and microphone input, as a potential source of stable device-specific fingerprints. This would open a new line of inquiry into physical-world fingerprinting and its repeatability across different environments and contexts. Understanding how fingerprint stability behaves across rooms, times of day, and device usage scenarios will be crucial to responsibly advancing this line of research.

Together, these future directions aim to deepen our understanding of audio-based and multi-modal fingerprinting systems while addressing the ethical concerns they raise for user privacy on the modern Web.

Acknowledgements

We thank Professor Hanan Hibshi for her consistent support, thoughtful feedback, and enthusiasm throughout the development of this project. Her expertise in browser security was instrumental in shaping our direction and deepening our understanding of the topic.

We also acknowledge the use of OpenAI's ChatGPT, which assisted in building the browser-based fingerprinting website and refining LaTeX graph visualizations. Its guidance helped streamline technical explanations, improve presentation clarity, and accelerate the implementation process.

Additional thanks to the teaching team and our classmates in 14-828 for the collaborative discussions, feedback, and encouragement that contributed to the final results of this work. We also thank the anonymous expert reviewers for *Majāl*, who helped us improve our manuscript.

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0

References

Boussaha, S., Hock, L., Bermejo, M., Cuevas Rumin, R., Cuevas Rumin, A., Klein, D., Johns, M., Compagna, L., Antonioli, D., & Barber, T. (2024). FPtracer: Fine-grained browser fingerprinting detection via taint-tracking and entropy-based thresholds. *Proceedings on Privacy Enhancing Technologies, 2024*(3), 540–560. https://petsymposium.org/popets/2024/popets-2024-0092.php

Catchpoint. (n.d.). *Web API vs. REST API: Comparing RESTful and non-RESTful web APIs.* https://www.catchpoint.com/api-monitoring-tools/web-api-vs-rest-api

Center for Computer Research in Music and Acoustics. (n.d.). *CCRMA.* Stanford University. https://ccrma.stanford.edu/

Chalise, S., Nguyen, H. D., & Vadrevu, P. (2022). Your speaker or my snooper?: Measuring the effectiveness of web audio browser fingerprints. In *Proceedings of the 22nd ACM Internet Measurement Conference* (pp. 349–357). ACM. https://doi.org/10.1145/3517745.3561435

Dewesoft. (n.d.). *FFT spectrum analysis (Fast Fourier Transform).* https://training.dewesoft.com/online/course/fft-spectral-analysis

Englehardt, S., & Narayanan, A. (2016). Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1388–1401). ACM. https://doi.org/10.1145/2976749.2978313

Matuszewski, B., & Rottier, O. (2023). The Web Audio API as a standardized interface beyond web browsers. *Journal of the Audio Engineering Society, 71*(11), 790–801. https://www.aes.org/e-lib/browse.cfm?elib=22344

MDN Web Docs. (2023, April). *GainNode - Web APIs.* Mozilla. https://developer.mozilla.org/en-US/docs/Web/API/GainNode

MDN Web Docs. (2024, July). *OscillatorNode - Web APIs.* Mozilla. https://developer.mozilla.org/en-US/docs/Web/API/OscillatorNode

MDN Web Docs. (2025, April). *AudioDestinationNode - Web APIs.* Mozilla. https://developer.mozilla.org/en-US/docs/Web/API/AudioDestinationNode

Queiroz, J. S., & Feitosa, E. L. (2019). A web browser fingerprinting method based on the Web Audio API. *The Computer Journal, 62*(8), 1106–1120. https://academic.oup.com/comjnl/article/62/8/1106/5298776

Su, J., & Kapravelos, A. (2023). Automatic discovery of emerging browser fingerprinting techniques. In *Proceedings of the ACM Web Conference 2023* (pp. 2178–2188). ACM. https://doi.org/10.1145/3543507.3583333

DOI: 10.48762/z470-dk61| Published by Carnegie Mellon University Library Publishing Service | CC BY 4.0