Your Speaker or My Snooper? Measuring the Effectiveness of Web Audio Browser Fingerprints

Shekhar Chalise
UNO Cyber Center
University of New Orleans
schalise@uno.edu

Hoang Dai Nguyen
UNO Cyber Center
University of New Orleans
hdnguye5@uno.edu

Phani Vadrevu
UNO Cyber Center
University of New Orleans
phani@cs.uno.edu

ABSTRACT

We conduct the first systematic study of the effectiveness of Web Audio API-based browser fingerprinting mechanisms and present new insights. First, we show that audio fingerprinting vectors, unlike other prior vectors, reveal an apparent fickleness with some users' browsers giving away differing fingerprints in repeated attempts. However, we show that it is possible to devise a graph-based analysis mechanism to collectively consider all the different fingerprints left by users' browsers and thus craft a highly stable fingerprinting mechanism. Next, we investigate the diversity of audio fingerprints and compare this with prior fingerprinting techniques. Our results show that audio fingerprints are much less diverse than other vectors with only 95 distinct fingerprints among 2093 users. At the same time, further analysis shows that web audio fingerprinting can potentially bring considerable additive value to existing fingerprinting mechanisms. For instance, our results show that the addition of web audio fingerprinting causes a 9.6% increase in entropy when compared to using Canvas fingerprinting alone. We also show that our results contradict the current security and privacy recommendations provided by W3C regarding audio fingerprinting.

ACM Reference Format:
Shekhar Chalise, Hoang Dai Nguyen, and Phani Vadrevu. 2022. Your Speaker or My Snooper? Measuring the Effectiveness of Web Audio Browser Fingerprints. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22), October 25–27, 2022, Nice, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3517745.3561435

1 INTRODUCTION

Browser fingerprinting presents a grave threat to the privacy of internet users as it allows user tracking even in private browsing modes. The recent advanced web APIs have tremendously increased the fingerprintable surface area of web browsers. As a result, researchers have extensively focused on measuring and tracking the evolution of browser fingerprints obtained by using APIs such as Canvas and WebGL [6, 10, 16, 29] in order to quantify the scope of the problem. However, despite being used in the wild since 2016 [7, 9], Web Audio API-based fingerprinting has remained a notable absence in such large-scale fingerprint measurement works. There exists no prior work that systematically measures the effectiveness of various Web Audio-based fingerprinting techniques and compares them with existing fingerprinting techniques to gauge their relative importance. In this work, we attempt to fill this important knowledge gap.

Contributions. Our paper's contributions are:

(1) *User Study:* We conducted the first systematic user study dedicated to Web Audio API-based browser fingerprinting by employing 2093 users across the world.
(2) *Feasibility Analysis:* We designed a graph-based fingerprint collation mechanism to overcome an apparent fickleness in the audio fingerprints and showed that it is feasible to use audio fingerprints in development of a stable and scalable browser fingerprinting mechanism.
(3) *Privacy Threat Analysis:* We presented diversity measures for 7 audio fingerprints. We also showed the relative effectiveness of these fingerprints in comparison to (and in conjunction with) other browser fingerprinting vectors such as Canvas, Font and `User-Agent`-based fingerprinting. This will help future browser developers to take informed design decisions regarding privacy protection.
(4) *Impact:* Our work revealed an oversight in the description of the privacy threat posed by Web Audio fingerprinting in W3C's documentation for which we filed an official bug report. Moreover, we have shared our raw datasets and fingerprinting code base upon specific requests from multiple browser vendors (Firefox, Tor and Brave) for addition into their fingerprinting test-suites.

We also make the code we used for our browser fingerprinting measurements publicly available[1].

2 BACKGROUND & SYSTEM DETAILS

The Web Audio API was first introduced in 2011 [22] in order to enable synthesis and processing of audio on the web with support for fine-grained timing controls, real-time sound effects as well as complex visualizations. The use of the API involves the creation of an "Audio Graph" which is a directed graph built by the users to enable arbitrarily complex audio modifications. The atomic components of this graph are the "Audio Nodes" which can represent any audio modules such as audio sources (files, synthesizers etc.), destinations (speakers, offline buffers etc.), modifiers and analyzers.

2.1 Audio Fingerprinting Vectors

We will now describe all the audio fingerprinting vectors whose effectiveness we systematically study in this work. We begin with 3 known audio fingerprinting vectors and then discuss 4 other vectors that we devised for this study.

Dynamics Compressor (DC). It is one of the two audio fingerprinting methods discovered in the wild in [9]. The audio graph for DC is in Fig. 1. The method involves the use of an `OscillatorNode` to

IMC '22, October 25–27, 2022, Nice, France
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9259-4/22/10…$15.00
https://doi.org/10.1145/3517745.3561435

[1] The URL is https://github.com/nguyenhoangdai/audio_fp_code

create a periodic audio waveform in a specific shape and feeding it to a DC Node. It is often used in muscial production to reduce distortion commonly exists in recorded audio samples. The main intuition behind this vector is that there might exist small identifiable differences in the way DC is done in different audio hardware/software stacks of different users. To capitalize on this, this method computes the hash output of DC as the fingerprint.

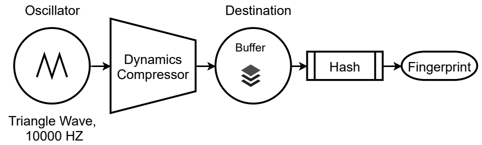


Figure 1: Dynamics Compressor (DC) Method

Fast Fourier Transform (FFT). The other audio fingerprinting method discovered in [9] is in Fig. 2 where the intuition is to make use of characteristic differences existing in the Fast Fourier Transformation (FFT) calculations performed by the web browsers when requested to transform a simple audio signal from time domain to frequency domain. This is accomplished with the help of an `AnalyserNode` and a `ScriptProcessorNode` after which the FFT output is sent to a hash function to produce the final output. Note that this method sends the audio signal to a `GainNode` whose gain (volume) is set to zero before forwarding to the speakers in order to make the fingerprinting undetectable to the end user.

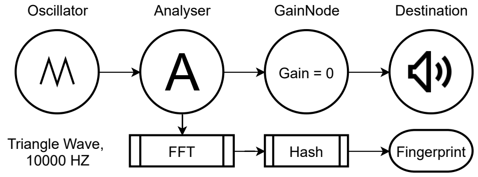


Figure 2: Fast Fourier Transform (FFT) Method

Hybrid (DC + FFT). The authors of [9] also developed another audio fingerprinting method (called "hybrid") that simply combines both DC and FFT methods in an attempt to increase the amount of "fingerprintability" [28] as is depicted in Fig. 6 in Appendix (Fig. 6). We obtained the code samples for all these three vectors from [28] and used them.

New Audio Fingerprinting Vectors. We also wanted to explore the possibility of being able to improve the known audio fingerprinting vectors. For this, we built four different vectors by extending the hybrid (DC + FFT) vector. The intuition behind this to make an attempt to add various complexities to the signals that are used in the fingerprinting methods and measure if this increases the diversity of the fingerprints. As this ultimately only caused a marginal impact on the diversity, we only describe these vectors at a high-level here and defer the details to Appendix B for interested readers.

The first vector we built was *Merged Signals* vector, which modifies the Hybrid vector by using a combination of four geometric shaped signals provided by the Web Audio API instead of a single shape as described above. *Custom Signal* is the next vector we built which uses a custom shaped signal instead of the predefined signal shapes of the web API. Finally, we also built two more vectors to see if the process of amplitude or frequency modulation can further improve the diversity of Hybdrid audio fingerprint vectors. We refer to these as *Amplitude* and *Frequency Modulation* vectors.

2.2 Experimental Setup

We set up a single web page to host the fingerprinting code that implements all the 7 vectors we discussed above. Along with this, we also included other browser fingerprinting vectors such as Canvas and JS-based Font by leveraging code from [1]. For any fingerprinting vector to be effective, it needs to have *stability* which means that the same user/browser pair should result in the same fingerprint even if fingerprinted repeatedly. To measure this, we designed our study's web page to repeatedly run the same audio fingerprinting code multiple times. This allowed to us to collect multiple fingerprints for each vector from each participant and evaluate the stability aspects of the audio fingerprinting mechanisms. To decide the number of repetitions, we ran pilot experiments during which we profiled our code in commodity machines. We noticed that by setting the number of iterations to 30, our entire fingerprinting code ran for about 30 to 60 seconds on these test machines. Since this amount of time matched the planned time for each volunteer in our study, we used 30 as the number of iterations. As a result, our web page was set up to collect a total of 210 audio fingerprints (30 iterations, 7 vectors) from each user.

The fingerprinting website was built with 5.8K lines of TypeScript code using the Angular 11.0.4 framework and the Cloud Firebase database. We also wrote about 10K lines of Python code for the fingerprint analysis presented in this paper. The web site is built to run all the fingerprinting code in the background after informing the visitor about the study and seeking consent from them in the form of a click.

2.3 Participants

Our study was conducted for 76 days during the months of March to May 2021. During this time, we recruited 2093 unique participants for our study with the help of Amazon's MTurk platform (2064) as well as our social circles (29). We had a very diverse participant pool covering as many as 57 different countries. Among those countries, the United States, India, Brazil and Italy were the most frequent with each of them having at least 100 participants. From the `User-Agent` HTTP headers, we inferred that our participants used different browsers such as Google Chrome, Mozilla Firefox as well as several Chrome-based browsers such as Microsoft Edge, Opera, Samsung Internet, Silk, and Yandex. Firefox was used by about 9.6% of the participants while the remaining 90.4% all used Chrome-based browsers. Our study also included all major OS families such as Windows (78.5%), Android (6.9%), MacOS (9.4%) and Linux (5.2%).

3 FEASIBILITY ANALYSIS

3.1 Preliminary Analysis

When conducting a preliminary analysis of the results for stability, we observed that the Web Audio API-based fingerprints have some "fickleness" with some users' browsers leaving more than 20 different fingerprints among the 30 iterations we make for each vector. These numbers are shown in the "Max." column of Table 1. This phenomenon appears unique to the Web Audio API-based fingerprinting as other HTML5 APIs abused for fingerprinting such as Canvas and WebGL [17] have been shown to be very stable and do not change unless there is a browser upgrade. Among the "Max." values, the

Vector	DC	FFT	Hybrid	Custom Signal	Merged Signals	AM	FM
Min.	1	1	1	1	1	1	1
Max.	1	21	18	18	21	26	24
Mean	1.0	1.81	2.08	2.08	2.92	4.28	4.33

Table 1: # Distinct fingerprints across 30 iterations for each user

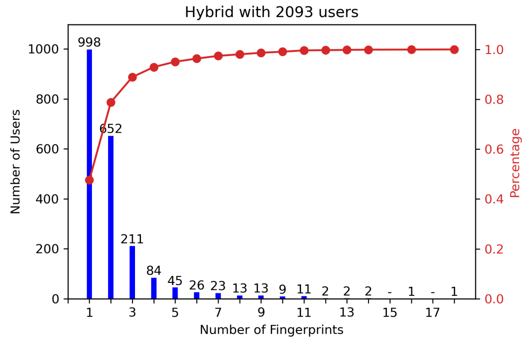


Figure 3: CDF and Bar plot for the distribution of distinct Hybrid (DC+FFT) audio fingerprints.

Dynamics Compressor (DC) vector stands out in the table as it results in only one stable fingerprint for each of the 2093 users across all 30 iterations. All the other vectors including the Hybrid vector are showing varying number of fingerprints (of at least 18 or more) across different iterations for some of the users. As the FFT is the only difference between Hybrid and DC vectors (see Section 2.1), it is likely that FFT calculations are what are causing this apparent instability in the extracted fingerprints.

At the same time, the other columns in Table 1 which show the minimum and mean number of fingerprints obtained from a user's browser reveal that there are users who only left one fingerprint among all 30 iterations. It is to be noted that the "Min." value is 1 for all rows in the table. Furthermore, the "Max." value for any row in the column is only 26 and not 30 even though the number of iterations of fingerprinting is 30. This shows that there are some fingerprints that are repeating for every vector across every user. Fig. 3 shows the distribution of fingerprint numbers for Hybrid audio vector[2]. It shows that the number of distinct fingerprints for most users is simply one or two thus indicating high degree of stability for most users. All of this shows that there is a degree of stability in all of these vectors.

3.2 Fingerprint Collation

Inspired by the above, we devise a simple graph-based approach to combine all the various fingerprints in the 30 iterations into a single fingerprint. For every fingerprinting vector, we build a separate undirected bipartite graph in which every user and every elementary fingerprint is represented by a node. For example, Fig. 4 represents a hypothetical graph for a particular vector after collecting 9 elementary fingerprints (eFP_1 to eFP_9) across 4 users (U_1 to U_4). In this graph, all fingerprint nodes are connected to all the user nodes that they were associated with during the fingerprint collection process. In order to collate the fingerprints, we simply consider each *connected component* in the graph to be representation of each **collated fingerprint**. Thus, the number of connected components is the number of distinct collated fingerprints and each user in a particular component can be considered to have the same fingerprint. In our example, we thus end up with 3 distinct fingerprints for the 4 users with users U_1 and U_2 having the same fingerprint while users U_3 and U_4 having a unique fingerprint that does not collide with any other user's. Thus each connected component can also be considered to be a cluster of users (**user cluster**) with colliding fingerprints.

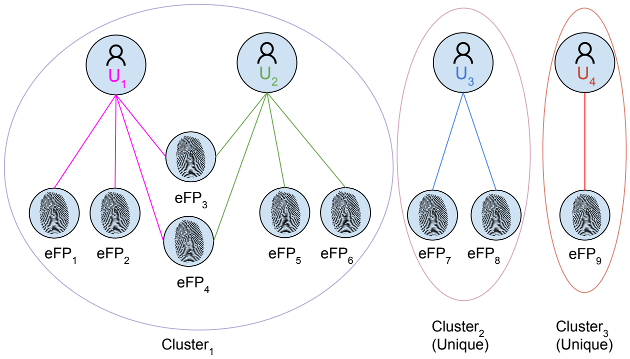


Figure 4: Our approach for collating multiple fingerprints into a single fingerprint

It is to be noted that with our proposed method, as we obtain fingerprints of more users, new collisions can pop up between users who were previously considered to be having distinct fingerprints. For example, consider a new user U_5 who has elementary fingerprints, eFP_8 and eFP_9. This merges existing second and third user clusters into one large cluster that make all three users U_3, U_4, U_5 to be considered to have the same colliding fingerprint. This means that the fingerprinting graph has to be adjusted in a dynamic fashion by the fingerprinter. For this, fingerprinters can rely on prior works such as [11] that proposed fully online graph algorithms for dynamic connectivity queries. The algorithm proposed in [11] has an amortized operation cost of $O(\log^2 n)$ for graph updates and $O(\log n/\log\log n)$ for connectivity queries where n is the number of vertices in the graph. Let us assume that a particular fingeprinter has u users fingerprinted with a particular vector where the number of iterations for each user is k (note that $u = 2093$ and $k = 30$ in our study). In the worst case, even if every fingerprint in every iteration for every user is distinct, the maximum number of nodes in vertices will be $(k+1)u$ as there will be u users and ku fingerprints. Thus, the graph update operation cost for a fingerprinter is only $O(\log^2 u)$ while the query operations cost even less. Thus, we can see that this approach scales well to even billions of users. Alternatively, one can also consider utilizing a disjoint-set data structure [25] for storing and performing audio fingerprint operations efficiently.

3.3 Stability Analysis

We have proposed a fingerprint collation approach in order to aggregate multiple fingerprints that were seen for all FFT-based vectors. However, the question of whether this approach results in *stable*

[2] The graphs for the remaining five FFT-based vectors are very similar and have been avoided due to space limitations.

fingerprints still remains. We attempt to answer this using two measurement approaches below.

Clustering Agreement Scores. For this, we first break down the fingerprint iterations in our dataset of size k (= 30) into multiple equal-sized subsets of size s. Then, for each vector and a particular value of s ($< k$), we can obtain a clustering of users using the proposed fingerprint collation algorithm. For example, consider the value of $s = 10$ which implies that we break down the elementary fingerprints obtained during the 30 iterations into 3 disparate subsets each of size 10. Using only the data from first subset, we obtain a different clustering of users for each audio fingerprinting vector v. We can then do the same for the other subsets resulting in a total of $\left\lfloor \frac{k}{s} \right\rfloor$ clusterings for each vector. We can then use a cluster agreement measuring algorithm to compare how much clusterings from each of the $\left\lfloor \frac{k}{s} \right\rfloor$ different subsets agree with one another. For measuring cluster agreement, we use the Adjusted Mutual Information (AMI) metric which is an information theoretic measure for clustering comparison [18]. We chose AMI as it was shown by researchers to be a suitable algorithm for comparing clusters of imbalanced sizes (with small-sized clusters) [23] which is typically the case with browser fingerprints [16]. The AMI scores vary between 0 and 1 with 1 indicating exact matching of two user clusterings.

We performed these measurements for different values of s and present the average cluster agreement scores across clusters in Fig. 5. Note that when s is not a factor of k (=30), we simply consider only the first $\left\lfloor \frac{k}{s} \right\rfloor s$ iterations which are part of the first s subsets and ignore the last few iterations. For $s = 4$, the minimum average value of the score is 0.986 (for FFT vector) whereas for $s = 15$, this value is 0.997 (for Merged Signals vector). The results clearly show that even for low values of s (as long as it is at least two), the audio fingerprints using our proposed graph-based collation algorithm result in user clusterings that are highly similar to one another for a given vector across repeated attempts.

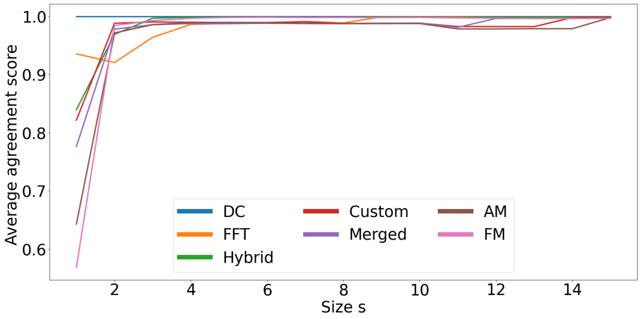


Figure 5: Average cluster agreement scores for different values of s ([1,15]) and different vectors.

Fingerprint Match Scores. It is also vital for a fingerprinter to pinpoint a given visitor to exactly the same user's connected component (or cluster) generated in a prior visit. This allows fingerprinting to be consistent across multiple visits of a given user. To measure this, we first divide the fingerprint iterations into subsets of size s (=3,10,15). For each value of s and each vector, we consider the first subset as a "training set" and use its fingerprints to build a **training graph** as in Fig. 4. We then consider the elementary fingerprints from each of the remaining subsets iteratively for each user and measure how many of the users can be mapped to the same cluster that they belong to as per the graph built from the first subset. Using this simple approach, we computed the fraction of remaining "user subsets" which we were able to positively point to the right cluster in the training graph. The results show that even for very small set size such as $s = 3$, the lowest fingerprint match score across all the vectors was only 0.9899. This score increased to 0.9978 for $s = 10$ thus showing that our method is able to accurately point the vast majority of users uniquely to their "original cluster" based on their current fingerprints. Table 6 in Appendix shows the full results.

4 DIVERSITY ANALYSIS

Entropy measures are commonly used to measure the diversity and there by, the "fingerprinting power" of browser fingerprints [10, 14, 16]. We followed the same approach for our study and computed the Shannon bit entropy as well as normalized entropy for all the web audio fingerprinting vectors that we studied. We describe the computation here for clarity. Assume that there exist n distinct fingerprints, with u_i (where $i \in [1,n]$) denoting number of users in the study that have the ith fingerprint and U denoting total number of users. We compute bit entropy $\boldsymbol{e}$ for a given fingerprinting vector as $e = -\sum_{i=1}^{i=n} \frac{u_i}{U} \log_2 \frac{u_i}{U}$

Then, the normalized entropy ($\boldsymbol{e_{norm}}$) is obtained by dividing the bit entropy by the maximum possible entropy i.e. $\frac{e}{\log_2 U}$ in order to bring it down to a range of 0 to 1. Note that 1 indicates maximum possible entropy and unique fingerprintability of every user. This normalized measure enables comparison between fingerprint fingerprinting measures of various studies even if the number of users in the study is different [10, 16].

The diversity of the 7 audio fingerprint vectors based on utilizing Web Audio APIs is presented in Table 2. It is to be noted that in order to compute these measurements, we utilized the graph-based approach as described previously. Thus, the "Distinct" column counts the number of clusters in the fingerprint graph while the "Unique" column counts the number of clusters associated with only a single user. In order to allow for comparison, in Table 3 we also present the entropy values of other fingerprinting vectors which were shown to be effective in prior works. Table 2 shows that FFT-based audio vectors are more effective at fingerprinting than pure Dynamics Compressor vector with a normalized Shannon entropy of more than 0.23. Most of these FFT-based vectors result in 80-85 distinct fingerprints for the 2093 users with about 40 of them being unique (i.e. only associated with one user in the dataset). The table shows that all the diversity values of the FFT-based vectors are very close to one another thus indicating that the discriminatory cause behind all these vectors is potentially the FFT operation alone.

The final row of Table 2 considers a combination of all the individual audio fingerprints. In order to compute the diversity of the combination of multiple fingerprinting vectors, the following simple logic is used. Assume, that a user U_i has multiple fingerprints associated with different vectors such as f_i, g_i, h_i etc. Then, in order to find the diversity of a combination vector of all these individual vectors, we simply compute the diversity of tuples: $(f_i, g_i, h_i, ...)$ across all values of i. By definition, the diversity of a combination vector will at least be as much as the diversity of the most diverse component vector. We can see in Table 2 that the entropy of combinations of all audio vectors is again close to that of the FFT-based vectors thus providing further proof for alignment of all FFT-based vectors. Moreover, we computed the cluster agreement scores between various audio fingerprint vector clusters and found that all FFT-based vectors have a very high agreement score (Fig. 9 in Appendix).

Comparing Tables 2 and 3 shows that the diversity of audio fingerprints is much less than that of other effective fingerprinting vectors such as Canvas, Fonts and `User-Agent` header based fingerprints. This difference can also be seen in terms of number of distinct and unique fingerprints.

Vectors	Distinct	Unique	Entropy	e_{norm}
DC	59	34	1.935	0.175
FFT	73	42	2.593	0.235
Hybrid	84	42	2.692	0.244
Custom Signal	72	41	2.582	0.234
Merged Signals	87	45	2.767	0.251
AM	82	45	2.69	0.244
FM	82	43	2.717	0.246
Combined	95	49	2.803	0.254

Table 2: Diversity of audio fingerprints (2093 users)

Vectors	Distinct	Unique	Entropy	e_{norm}
Canvas	352	224	6.109	0.554
Fonts	690	555	7.146	0.648
`User-Agent`	427	284	6.466	0.586

Table 3: Diversity of other vectors (2093 users)

Comparison with User-Agent fingerprints. The `User-Agent` (UA) header is an indicator of the web browser, its version number as well as the OS being used to visit a web server. Tables 2 and 3 indicate that the entropy of UA header-based fingerprinting is more than the entropy of audio fingerprints. On a related note, we noticed that the "Security and Privacy Considerations" section of the World Wide Web Consoritum's (W3C) organization's standards document states that Web Audio fingerprinting "merely allows deduction of information already readily available by easier means (User Agent string)" [30]. Given that ours is the first systematic study of web audio fingerprints, we wanted to utilize our data to verify this statement.

For this, we first considered the UA strings that were each associated with more than one user in our dataset. There were 143 such UA strings and were seen with 1950 users in total in our study. Of these 143 UAs, we noted that as many as 90 of them were spanning multiple FFT-based fingerprint clusters[3]. Together, these accounted for about 1610 of the 1950 users. Further, several of these UAs were associated with more than 2 fingerprint clusters. For example, 7 UAs were each associated with at least 5 different Merged Signal fingerprints with one particular Chrome/Windows UA being associated with as many as 10 different fingerprints. However, we did not notice any explicit differences between browser families in this behavior with both Firefox and Chrome UAs both getting frequently associated with more than one audio fingerprint. This clearly shows that *unlike what was mentioned in W3C's documentation, there are a significant number of cases where audio fingerprinting reveals more information about users than* `User-Agent` *fingerprinting alone.*

Additive Value of Audio Fingerprints. The above showed that web audio fingerprints have more fingerprinting value beyond simply recording the `User-Agent` header. It would be useful to quantify this additive value that audio fingerprinting can potentially add to existing powerful fingerprinting schemes. For this, we first consider Canvas fingerprinting as it was shown to be one of the most discriminative fingerprinting techniques previously [16]. We measured the entropy of a "pure" Canvas API-based fingerprinting technique as well as "Canvas + Audio" fingerprint where Audio fingerprint includes an aggregation of all 7 web audio fingerprinting techniques as described previously and shown in the final row of Table 2. Our measurements show that "Canvas + Audio" fingerprint has an entropy of 6.699 in comparison to an entropy of 6.109 for the pure "Canvas" vector thus showing that *audio fingerprinting helps cause a 9.6% increase in the normalized entropy of Canvas fingerprinting techniques.* Similarly, we also repeated this analysis for "UA + Audio" and saw that it resulted in a *a 9.7% increase from using just UA as a fingerprint thus reaffirming the additive value of audio fingerprinting to UA fingerprinting.*

Mitigations. It is to be noted that Audio fingerprinting (like Canvas fingerprinting) is more difficult to defend against unlike other techniques such as Font and `User-Agent` fingerprinting. The latter can be tackled by simply changing fonts/`User-Agent` headers (using a browser extension such as [26]) periodically in a browser. However, combating Canvas and Web Audio fingerprinting techniques requires more intricate measures such as the fingerprint randomization measures taken up by the Brave Browser recently [3, 13] which can have considerable computational as well as compatibility side-effects [4, 5]. Our measurement results in this paper can help browser developers to weigh the privacy risks their user might face if audio fingerprinting is left undefended against the compatibility risks of possible defenses and act accordingly.

5 DISCUSSION

Participant Pool Size. Due to financial limitations, we had to restrict the size of our study to 2093 users who were mainly recruited and paid via Amazon's MTurk platform. However, it is important to note that the normalized Shannon entropy measures that we obtained for some well known fingerprinting vectors such as Canvas and `User-Agent` are in line with the figures from prior studies that employed even more number of users. For example, the normalized entropy for `User-Agent` headers in [16] which employed 118,934 users is 0.580 while it is 0.586 in our study. Furthermore, we also performed additional analysis to see how our dataset sizes can affect the relative rankings we present. For this, we divided our set of users into 4 disparate equal sized subsets and repeated the entropy analysis for each subset. We noticed that the relative rankings (by e_{norm}) of the 9 fingerprinting vectors we covered in Tables 2 and 3 remained *exactly the same across all the small subsets as well as our main dataset.* This further confirms that the analysis we present in our paper remains the same irrespective of the size of the user set that is considered.

Disclosure and Followup. As discussed in Section 4, some of our results regarding the diversity of audio fingerprints clearly contradict

[3] This number of 90 is about same for all the 6 FFT-based vectors.

the web API standards documentation. We disclosed our results to the Web Audio Working Group in the form of a bug report in order to request the documentation's "Security and Privacy Considerations" subsection [30] be updated to accurately delineate the potency of web audio fingerprinting attacks. After our report was shared with the major browser developers, some browser developers (Firefox, Tor and Brave) expressed interest in the results of our project and provided feedback on our work. Upon request, we shared our project's fingerprinting code with them to have it integrated into their test suites.

Causal Factors. During disclosure, some developers hinted that the fingerprintability of Math JS [24] is potentially the sole causal factor behind Web Audio fingerprinting. Web Audio processing involves complex mathematical operations whose implementation difference can result in an exploitable fingerprinting surface. In order to investigate this further, we performed a small follow-up study by employing 528 users. Our study found that while FFT vectors had an entropy of 2.3 bits and DC vectors had an entropy of 1.3 bits, MathJS vectors only had an entropy of about 0.4 bits thus showing that audio fingerprinting goes beyond MathJS fingerprinting. We can also see this difference in terms of total number of discinct fingerprints. While there were only 7 Math JS fingerprints among 528 users, there were 16 distinct DC Web Audio fingerprints. These results are shown in Table 4.

Vectors	Distinct	Unique	Entropy	e_{norm}
DC	16	4	1.301	0.144
FFT	24	7	2.288	0.253
Hybrid	25	9	2.240	0.248
Math JS	7	2	0.416	0.046

Table 4: Comparison with Math JS fingerprinting to investigate causal factors (528 users).

Surprised by this result, we dug deeper into the results to see if this difference is uniform across different platforms. The results of this analysis in Table 5 show that for the most part, there is a one-to-one correspondence between DC and Math JS fingerprints. For example, users on Windows/Chrome and Windows/Edge platforms who account for 80% of all users in this experiment have only a single DC and Math JS fingerprint. However, both macOS and Android operating systems clearly saw more diversity in Web Audio fingerprints than Math JS thus hinting that there are likely other causes behind Web Audio fingerprinting beyond Math JS implementation differences on these platforms. Further investigation of the role of these browser/OS platform differences as well as other potential factors such as hardware differences, varying CPU load will be pursued as future work.

6 RELATED WORK

Multiple works have devised browser fingerprinting techniques [6, 17, 20] and studied fingerprint defenses [8, 12, 13, 15, 19, 27]. Many have also focused on measuring and comparing the effectiveness as well as evolution of various browser fingerprints [10, 16, 29]. However, audio fingerprinting measurements have remained a notable absence in this. Only [9] who first discovered audio fingerprinting in the wild and [13] have briefly touched upon diversity aspects of audio fingerprinting with a Dynamics-Compressor (DC) vector by conducting user studies. However, none of these works have focused on studying the stability of audio fingerprints by using repeated fingerprinting attempts across the same user. Further, as these works were not full-fledged studies on Web Audio fingerprinting, they have not explored alternative audio fingerprint mechanisms (other than DC and FFT) as we have done here with this paper.

Platform	# Users	DC	Math JS
Windows/Chrome	393	1	1
macOS/Chrome	30	5	1
Windows/Edge	27	1	1
Windows/Firefox	25	1	3
Android/Chrome	21	5	1

Table 5: Comparing number of distinct DC and Math JS fingerprints across different platforms (528 users).

Nevertheless, both [9] (in 2016) and [13] (in 2017) have cited the entropy measures obtained with their web audio fingerprinting user studies. This gave us an opportunity to estimate how web audio fingerprinting surface has evolved in relation to prior related works. We computed the normalized entropy for their user studies and present these values in obtained a value of 0.38 for (DC+FFT vectors) and 0.24 for [13] (DC vector). Our normalized entropy measure of 0.244 (DC+FFT vector) and 0.175 (DC vector) shows that the fingerprinting surface has reduced slightly in the past few years likely due to privacy protection measures pursued by browser developers such as standardizing Math JS computations [2].

Our closest related work is [21] by Queiroz et al. as it is the only other dedicated study on Web Audio fingerprinting. In this work, the authors manually studied the stability of audio fingerprinting by using DC and FFT schematics similar to the ones we used in our paper with the help of four personal devices. Unfortunately, based on the apparent "fickleness" in the fingerprints exhibited by the FFT vectors in their preliminary study, the authors decided to exclude FFT-based fingerprints fro and rather only use pure DC fingerprinting vectors for further evaluation (with 122 devices and 4 separate `OscillatorNode` signals). However, as we demonstrated with our proposed graph-based approach, FFT vectors can in deed be used as stable fingerprinting vectors. Furthermore, we have shown that these have superior diversity results compared to a pure DC vector yielding an entropy of 2.593 bits versus 1.935 bits for a DC vector in our user study.

[21] and other works also do not measure the additive value of Web Audio fingerprinting in the context of other previously known fingerprinting vectors. This is vital as it serves as a measurement to gauge how benefical Web Audio fingerprinting is to an attacker.

Acknowledgements. The authors would like to thank Peter Snyder, Simon Mainey, Vassil Roussev and the anonymous reviewers for providing helpful feedback. This work was partly supported by the National Science Foundation (NSF) under grant CNS-2126655.

7 CONCLUSION

In this paper, we conducted the first systematic study of effectiveness of Web Audio-based browser fingerprinting vectors. Our results show that audio fingerprints are much less diverse than other vectors with only 95 distinct fingerprints among 2093 users. At the same time, further analysis shows that web audio fingerprinting can potentially bring considerable additive value (in terms of entropy) to existing fingerprinting mechanisms.

REFERENCES

[1] [n. d.]. FingerprintJS. ([n. d.]). https://github.com/fingerprintjs/fingerprintjs

[2] [n. d.]. Floating point differences between platforms. https://bugzilla.mozilla.org/show_bug.cgi?id=531915. ([n. d.]).

[3] Brave. 2020. Fingerprinting 2.0: Web Audio · Issue #9187. (Apr 2020). https://github.com/brave/brave-browser/issues/9187

[4] Brave. 2021. Html5 Canvas Web Font Alignment is off · Issue #15326 · brave/brave-browser. (Apr 2021). https://github.com/brave/brave-browser/issues/15326

[5] Brave. 2021. Rendering issue on Google Sheets · Issue #13448 · brave/brave-browser. (Jan 2021). https://github.com/brave/brave-browser/issues/13448

[6] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society. https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/cross-browser-fingerprinting-os-and-hardware-level-features/

[7] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 1515–1532. https://doi.org/10.1145/3243734.3243860

[8] Amit Datta, Jianan Lu, and Michael Carl Tschantz. 2019. Evaluating Anti-Fingerprinting Privacy Enhancing Technologies. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. 351–362. https://doi.org/10.1145/3308558.3313703

[9] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 1388–1401. https://doi.org/10.1145/2976749.2978313

[10] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 309–318. https://doi.org/10.1145/3178876.3186097

[11] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. 2001. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM* 48, 4 (2001), 723–760. https://doi.org/10.1145/502090.502095

[12] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2020. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. *CoRR* abs/2008.04480 (2020). arXiv:2008.04480 https://arxiv.org/abs/2008.04480

[13] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. 2017. FPRandom: Randomizing Core Browser Objects to Break Advanced Device Fingerprinting Techniques. In *Engineering Secure Software and Systems - 9th International Symposium, ESSoS 2017, Bonn, Germany, July 3-5, 2017, Proceedings (Lecture Notes in Computer Science)*, Eric Bodden, Mathias Payer, and Elias Athanasopoulos (Eds.), Vol. 10379. Springer, 97–114. https://doi.org/10.1007/978-3-319-62105-0_7

[14] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser Fingerprinting: A Survey. *ACM Trans. Web* 14, 2 (2020), 8:1–8:33. https://doi.org/10.1145/3386040

[15] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2015. Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*. 98–108. https://doi.org/10.1109/SEAMS.2015.18

[16] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 878–894. https://doi.org/10.1109/SP.2016.57

[17] Keaton Mowery and Hovav Shacham. 2012. Pixel Perfect: Fingerprinting Canvas in HTML5. In *Proceedings of W2SP 2012*, Matt Fredrikson (Ed.). IEEE Computer Society.

[18] Xuan Vinh Nguyen, Julien Epps, and James Bailey. 2009. Information theoretic measures for clusterings comparison: is a correction for chance necessary?. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009 (ACM International Conference Proceeding Series)*, Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman (Eds.), Vol. 382. ACM, 1073–1080. https://doi.org/10.1145/1553374.1553511

[19] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. 2015. PriVaricator: Deceiving Fingerprinters with Little White Lies. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*. 820–830. https://doi.org/10.1145/2736277.2741090

[20] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2013. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*. 541–555. https://doi.org/10.1109/SP.2013.43

[21] Jordan S Queiroz and Eduardo L Feitosa. 2019. A Web Browser Fingerprinting Method Based on the Web Audio API. *Comput. J.* 62, 8 (01 2019), 1106–1120. https://doi.org/10.1093/comjnl/bxy146 arXiv:https://academic.oup.com/comjnl/article-pdf/62/8/1106/29162322/bxy146.pdf

[22] Chris Rogers. [n. d.]. Web Audio API is now available in Chrome. https://lists.w3.org/Archives/Public/public-xg-audio/2011Feb/0000.html. ([n. d.]).

[23] Simone Romano, Xuan Vinh Nguyen, James Bailey, and Karin Verspoor. 2016. Adjusting for Chance Clustering Comparison Measures. *J. Mach. Learn. Res.* 17 (2016), 134:1–134:32. http://jmlr.org/papers/v17/15-627.html

[24] Takamichi Saito, Takafumi Noda, Ryohei Hosoya, Kazuhisa Tanabe, and Yuta Saito. 2018. On estimating platforms of web user with JavaScript math object. In *International Conference on Network-Based Information Systems*. Springer, 407–418.

[25] Raimund Seidel and Micha Sharir. 2005. Top-Down Analysis of Path Compression. *SIAM J. Comput.* 34, 3 (2005), 515–525. https://doi.org/10.1137/S0097539703439088

[26] Chrome Web Store. [n. d.]. User-Agent Switcher for Chrome. ([n. d.]). https://chrome.google.com/webstore/detail/user-agent-switcher-for-c/djflhoibgkdhkhhcedjiklpkjnoahfmg?hl=en-US

[27] Christof Ferreira Torres, Hugo L. Jonker, and Sjouke Mauw. 2015. FP-Block: Usable Web Privacy by Controlling Browser Fingerprinting. In *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part II*, Vol. 9327. 3–19. https://doi.org/10.1007/978-3-319-24177-7_1

[28] Princeton CITP's Web Transparency and Accountability Project. [n. d.]. Audio-Context Fingerprint Test Page. ([n. d.]). https://audiofingerprint.openwpm.com/

[29] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 728–741. https://doi.org/10.1109/SP.2018.00008

[30] WWWC. 2021. (May 2021). https://web.archive.org/web/20210517012714/https://www.w3.org/TR/webaudio/#priv-sec

A ETHICS

Our study primarily relied on browser fingerprint information collected from real world users. Besides the browser fingerprint data, no other sensitive user information is collected from the participants. We also also presented a clear disclosure message to the participants informing them about browser fingerprint extraction prior to the beginning of the study. For any visitor to our web page, only after seeking the consent of the user study participants in the form of a click, we start collecting browser fingerprint data from the user. We disclosed our experimental procedure to the IRB board at our university and sought an exemption from them for our study.

B NEW FINGERPRINTING VECTORS

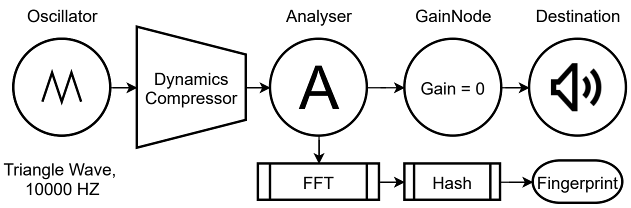


Figure 6: Hybrid (DC + FFT) Method

Along with studying the effectiveness of known audio fingerprinting vectors, we also wanted to see if it is possible to improve these vectors in order to increase their "fingerprintability" of audio software/hardware stack. For this, we created 4 new vectors by extending the hybrid (DC + FFT) vector. In all the vectors, we attempted to create more complicated signals so as to increase diversity in fingerprints. We describe these below.

Merged Signals. Our first idea in extending the earlier hybrid vector is to simply use multiple signals instead of the single `triangle` signal. This is depicted in Figure 7. Our main idea was to check if using other shapes of the waves could potentially increase the diversity of fingerprints. For this, we used all the four shapes of waves supported by `OscillatorNode` (generated in different frequencies). We then merged them together using `ChannelMergerNode` which is usually used to combine mono audio inputs (such as L,R,C etc) into a single output channel. The rest of the fingerprinting mechansim is the same as that of the hybrid method.

Custom Signal. For our second vector, we used the '`custom`' wave shape type supported by `OscillatorNode` which allowed us to define our own wave shape. We used an array of 12 real and imaginary values to define this periodic signal with real values randomly selected between 0 and 1 and imaginary values alternating between 0 and $\pi/2$. It is to be noted that a '`custom`' wave type was also used as an input to a DC fingerprinting vector in [21] previously. More detailed comparison with [21] is presented in Section 6.

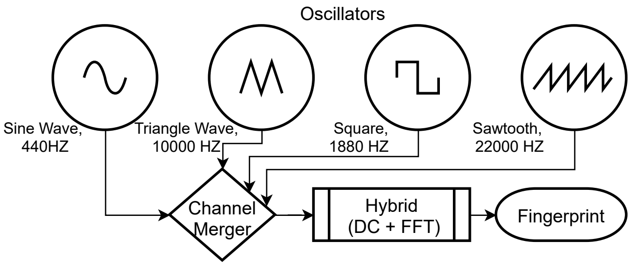


Figure 7: Merged Signals Method

Amplitude Modulation (AM). We also wanted to create an Amplitude Modulated (AM) wave signal in order to see if the process of modulation increases the fingerprint diversity. For this, as depicted in Figure 8, we generate two waves (triangle and square) and modulate them with the help of another generated sine wave as a carrier wave.

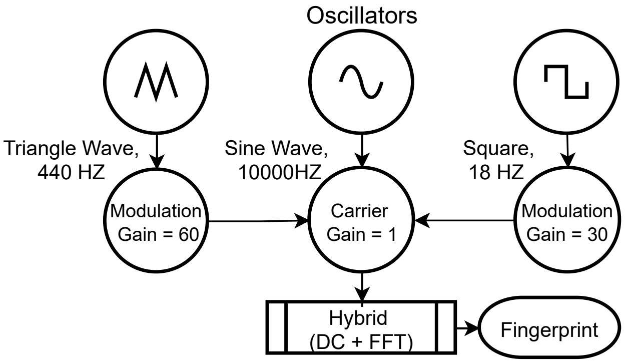
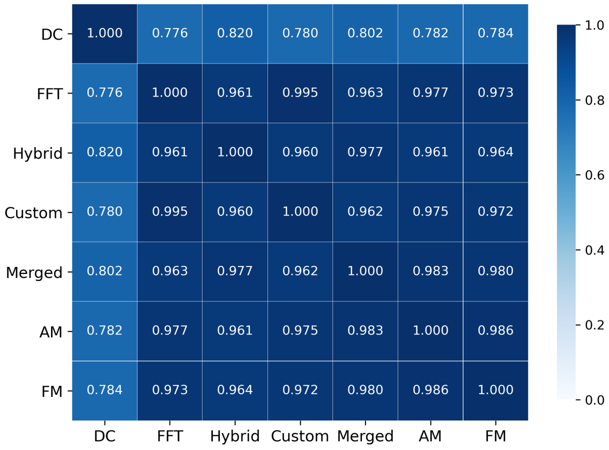


Figure 8: Amplitude Modulation Method

Frequency Modulation (FM). This final method is the same as previous AM method except that we used Frequency Modulation (FM) instead.

C ADDITIONAL RESULTS



	DC	FFT	Hybrid	Custom	Merged	AM	FM
DC	1.000	0.776	0.820	0.780	0.802	0.782	0.784
FFT	0.776	1.000	0.961	0.995	0.963	0.977	0.973
Hybrid	0.820	0.961	1.000	0.960	0.977	0.961	0.964
Custom	0.780	0.995	0.960	1.000	0.962	0.975	0.972
Merged	0.802	0.963	0.977	0.962	1.000	0.983	0.980
AM	0.782	0.977	0.961	0.975	0.983	1.000	0.986
FM	0.784	0.973	0.964	0.972	0.980	0.986	1.000

Figure 9: Cluster agreement scores between the different audio fingerprinting vectors.

Fingerprinting Vectors	$s = 15$	$s = 10$	$s = 3$
DC	1.0	1.0	1.0
FFT	1.0	1.0	0.9942
Hybrid (DC + FFT)	1.0	1.0	0.9952
Custom Signal	0.999	0.9988	0.9969
Merged Signals	1.0	0.9998	0.9953
AM	0.999	0.9983	0.991
FM	0.9981	0.9978	0.9899

Table 6: Fingerprint match scores.