Exploring and Applying Browser Fingerprinting Methods and Their Countermeasures

Ruhi Siddiqui[1], Md Tabrez Nafis[2]
[1,2]Department of Computer Science & Engineering, Jamia Hamdard, New Delhi, India
(E-mail: ruhirsiddiqui@gmail.com, tabrez.nafis@gmail.com)

Abstract - Browser fingerprinting is a sophisticated device-identification method utilized to monitor users' activity across the web. While it offers advantages to website operators by providing insights into user demographics and interaction patterns, it simultaneously raises significant privacy concerns. This paper investigates prevalent t browser fingerprinting techniques, evaluates countermeasures adopted by modern browsers, and presents the design and implementation of a browser extension aimed at reducing user identifiability. The extension's effectiveness is assessed through analysis on real-world websites.

I. INTRODUCTION

Internet tracking has long been associated with privacy issues, many of which remain unnoticed by everyday users. As a result, it is often unclear to the average person how to effectively prevent such tracking. At its core, tracking involves gathering details about user identity and their interactions on a single website or across multiple platforms. This data collection is crucial for numerous websites, as it enables administrators to analyze visitor behaviour and engagement. Such insights support better decision-making related to website design, user experience, and business operations.

Services like Google Analytics assist website owners by providing metrics on how users discover and navigate their site, as well as demographic data such as user locations and visit duration. These analytics play a significant role in improving site performance, enhancing interface design, and enabling targeted advertisements or personalized content delivery.

Initially, websites relied on cookies— small text-based files stored in a user's browser—to retain information about user sessions and identities. While users can delete these cookies to limit tracking, more sophisticated tracking techniques have emerged. These methods often involve placing tracking data in obscure or less accessible storage locations, making it harder for users to detect or remove them. As traditional tracking methods became more transparent, efforts were initiated to limit or eliminate the use of Cookies —especially by third-party entities. Some browsers even introduced features to block or automatically delete cookies in response to rising privacy concerns.

As the effectiveness of cookies declined, browser fingerprinting emerged as a more advanced and persistent form of user identification. This technique functions independently of cookies, relying instead on data passively collected from a user's browser. While this data—such as screen resolution, graphics rendering capabilities, fonts, plugins, memory usage, and browser version—is typically used to enhance site performance, when aggregated, it can create a unique and identifiable profile of the user. This fingerprint enables websites to track users across single or multiple domains.

The practice of browser fingerprinting has become a major privacy issue, as it allows for continuous monitoring of user behavior, often without their knowledge or consent. Even when users adopt privacy-enhancing practices like clearing cookies, using incognito modes, or disabling trackers, fingerprinting can still persist as a method of identification.

In this project, we conducted a thorough investigation of how browser fingerprinting operates. We first explored the technical foundations by developing a sample website that implements several standard fingerprinting techniques. Following this,we studied the defensive measures integrated into modern web browsers to evaluate how well they prevent fingerprint-based tracking. To further our understanding, we also designed and developed a custom Google Chrome extension that applies fingerprint-masking strategies with the aim of minimizing user identifiability.

II. RELATEDWORKAND MOTIVATION

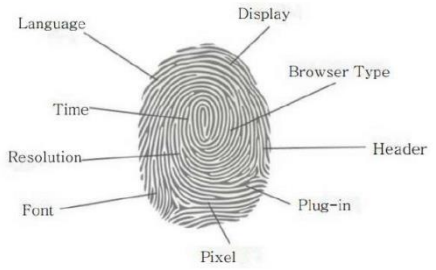

Fig. 1. Information from many data sources is collected to generate a user's digital fingerprint

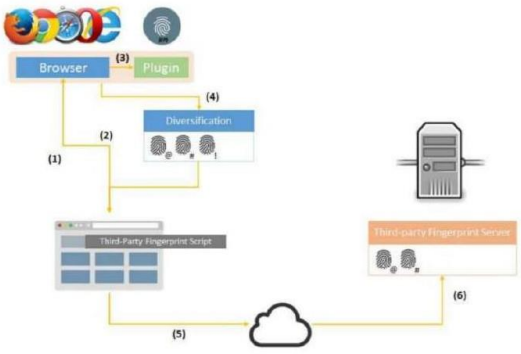

Fig. 2. Browser Finger Printing Diversification Flow

Device fingerprinting was initially introduced as a fraud-prevention mechanism, especially by financial institutions. These systems aimed to detect suspicious behaviour by comparing it to patterns from known devices, without relying on cookies or browser-stored data .Instead, they identified users by correlating multiple device-specific attributes.

A comprehensive study by Englehardt and Narayanan [3] revealed that among a sample of 100,000 devices, nearly 80% of desktop and mobile browsers exhibited unique fingerprints. Their research further highlighted the growing adoption of third-party tracking systems and HTTP-based fingerprinting across websites.

In particular, their findings showed that companies like Google could monitor user activity on nearly 80% of websites via third-party integrations and network requests [3]. Despite raising public awareness, these tracking methods remain prevalent. In response, major browsers have begun incorporating built-in protections aimed at limiting the data accessible to fingerprinting scripts.

III. APPROACH

This section outlines the methodology we followed to explore browser fingerprinting in depth, evaluate the features commonly exploited for identification, investigate possible countermeasures, and assess the effectiveness of our proposed solution.

A. Understanding Fingerprinting Mechanisms

To gain practical insights into how fingerprinting works, we built a demonstration website that collects a range of browser and device attributes.

This setup allowed us to simulate common fingerprinting methods and analyze which features are most frequently used to build a unique user profile.

To capture a range of data points from users' browsers, we designed a website capable of extracting multiple identifying features.

Before proceeding with the implementation, it was essential to first identify the specific attributes that websites typically use to create browser fingerprints. To do this, we examined several fingerprinting- focused platforms:

- CoverYourTracks (coveryourtracks.eff.org)
- BrowserLeaks (browserleaks.com)
- FingerprintJS(fingerprintjs.com)

While there was noticeable overlap in the types of fingerprinting data these tools collected, each service offered distinct insights. FingerprintJS, for instance, does not directly show the exact browser attributes it gathers, instead presenting users with a unique identifier. However, since it is open- source, we were able to examine its codebase to better understand the fingerprinting techniques it employs.

Among these, CoverYourTracksproved to be the most informative. It provides a comprehensive assessment of how resistant a browser is to fingerprinting and outlines the specific data points used to construct a fingerprint profile.

Using the information from these tools, we sought to replicate the fingerprinting methods on our own test site. To keep our project focused and manageable, we limited our implementation to only those features identified by CoverYourTracks. The status of each implemented feature is illustrated in Figure 3.

This implementation phase gave us practical experience with fingerprinting methods that are actively used on the web today. The insights gained from this phase laid the foundation for developing our anti-fingerprinting browser extension, as discussed in subsequent sections.

Coveryourtracks	Our website
User Agent	Yes
HTTP_Accept Headers	No
Browser Plugin Details	Yes
Time Zone Offset	No
Time Zone	Yes
Screen Size and Color Depth	Yes
System Fonts	Yes
Cookies Enabled	Yes
Limited Supercookie Test	No
Hash of Canvas Fingerprint	No
Hash of WebGL Fingerprint	No
WebGL Vendor and Renderer	Yes
Do-not-track Header Enabled	Yes
Language	Yes
Platform	Yes
Touch Support	Yes
Ad Blocker Used	No
Audiocontext Fingerprint	No
CPU Class	No
Hardware Concurrency	Yes
Device Memory	Yes
Network Details (not on Coveryourtracks)	Yes

Fig. 3. CoverYourTracks fingerprint details

B. Defences against Browser Fingerprinting

After examining various fingerprinting methods, we evaluated how some of the widely-used web browsers attempt to mitigate these techniques. Our analysis focused on data provided by CoverYourTracks, along with the available documentation and source code for Chrome, Firefox, Brave, and Tor Browser.

Based on our findings, we identified three primary strategies that browsers use to resist fingerprinting. These approaches are outlined below:

1) **Anonymization:** This technique modifies certain browser attributes to reflect the most statistically common values among users. For instance, since Windows is the most prevalent desktop operating system, a browser employing anonymization may set the user-agent string to report Windows, even if the actual system is macOS or Linux. This method is notably used by the Tor Browser.

   The advantage of anonymization is that it prevents websites from collecting enough distinctive data to uniquely identify individual users. However, its effectiveness depends on accurate knowledge of the distribution of user traits, which often requires gathering data from the browser's user base. Furthermore, if only certain properties are altered while others remain unique, the resulting fingerprint could paradoxically stand out more. For example, a browser might claim to run on Windows, but simultaneously reveal the presence of fonts that are exclusive to another operating system.

For instance, if a browser identifies itself as running on Windows but reports fonts specific to macOS, the inconsistency may increase its uniqueness, making the user more trackable.

2) **Randomization:** In this method, browser-exposed features are assigned random—but plausible— values, which may not reflect the user's actual configuration. For example, the browser may report a random value for available device memory. The Brave browser adopts this technique in certain areas, such as randomizing the number of logical processors (hardware concurrency).
One of the key advantages of randomization is that it does not rely on statistical profiling of the user base. Instead, it disrupts long-term tracking by altering fingerprintable features between sessions. While a user may still appear unique during a specific visit, their fingerprint changes with subsequent visits, preventing persistent identification. However, improper implementation of randomization can be counterproductive. The use of implausible or inconsistent values may itself become a distinguishing trait. Additionally, random values can impact functionality—such as when a browser misreports its platform, leading tolayout or rendering issues on websites designed for different device types.
3) **Disabling Feature Access:** Another mitigation strategy is to block access to browser APIs that expose fingerprintable attributes. By disabling these interfaces, the browser prevents websites from retrieving sensitive information altogether. Firefox, for example, restricts access to certain APIs such as those related to browser plugins.

Although effective in limiting fingerprinting vectors, this approach has its trade-offs. Some information, like the user-agent string or screen resolution, is often necessary for optimizing user experience across diverse devices. In our testing, we observed that omitting the user-agent string led certain websites to assume a default configuration, such as serving content formatted for Android browsers on desktop systems.
In the browser extension developed as part of this work, we incorporated the principles behind all three strategies discussed above. However, for this project, we chose to focus on implementing the randomization technique due to its balance of privacy protection and adaptability.

C. Anti-Fingerprinting Chrome Extension

This section details the design and implementation of a custom Chrome browser extension created to introduce controlled randomness into fingerprintable attributes. The goal was to minimize consistent fingerprinting while preserving usability during typical browsing sessions.

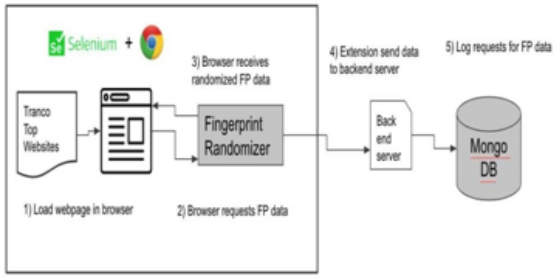

Fig. 4. Flow of Data Collection

Object.defineProperty(window, "screen", {
value:{
width: randInt, height: randInt

D. Techniques Used in Extension Development

1) **JavaScript Property Modification:** A majority of fingerprinting techniques rely on JavaScript to extract information from the browser environment. For instance, a website may use the following script to access the dimensions of the user's screen:
javascript CopyEdit
var width = window.screen.width; var height = window.screen.height;

To counter this, we override certain JavaScript properties with randomized values before the site scripts can execute. JavaScript's Object.defineProperty method enables such manipulation by redefining built-in object properties. The following code snippet demonstrates how window.screen can be assigned randomized width and height values:
javascript CopyEdit
}
});

A critical challenge is ensuring that these modifications take effect before the website's own scripts are executed. Fortunately, Chrome extensions allow for early script injection using the "run_at":"document_start" configuration [4], which executes the script before the page begins loading.

2) **Canvas Fingerprinting Defence:** Canvas fingerprinting is a common tracking technique that involves rendering text or shapes on an off- screen HTML5 <canvas> element. JavaScript is used to draw complex content on this canvas, and the rendered image is converted into a pixel-based hash. Due to hardware and driver variations, the same content will yield different hash values across different systems [5].

To mitigate this, one effective strategy is to subtly alter the canvas content before it is processed—introducing controlled noise to modify the resulting hash. By doing so, we ensure that the fingerprint value varies between sessions or page loads, thereby disrupting consistent tracking.

3) Canvas Fingerprinting Countermeasures:

One method to disrupt canvas fingerprinting is by introducing random visual noise—such as altering individual pixels—during the rendering process. This creates variation in the resulting canvas hash across sessions or devices. The primary benefit of this technique is that it allows the existing hash function to remain mostly untouched. However, its drawback is that the visual noise may be noticeable to users, potentially affecting the appearance of web content.

To avoid visual disruption, we adopted an alternative solution that involves directly modifying the function responsible for generating canvas data. Typically, canvas fingerprinting relies on the to Data URL method, which belongs to the HTML Canvas Element Prototype. Since all canvas elements inherit this method, overriding it at the prototype level ensures the change is applied universally.

By redefining to Data URL, we can B introduce controlled randomness or distortions into the canvas output before it is converted into a hash, effectively preventing consistent tracking across sessions.

4) Real-Time User Alerts on Fingerprinting Access:

To enhance transparency, our extension also includes a mechanism to notify users whenever a website attempts to access finger printable properties that have been randomized. This is implemented by intercepting the access through getter functions, which trigger alerts while returning randomized values.

For example, we override the window.Screen object so that any attempt to access screen.width or screen.height prompts an alert.The approach is illustrated in the code below:

```
javascript CopyEdit
Object.defineProperty(window, "screen", {
value:{
get width() { alert("Attempt toaccess
screen.width"); returnrandInt;
},
get height() { alert("Attempt toaccess
screen.height"); returnrandInt;
}
}
});
```

This functionality allows users to be immediately informed whenever a site attempts to gather information that could contribute to a unique browser fingerprint.

For functions, integrating user notifications is relatively straightforward, as alert messages can be directly inserted into the function body. However, handling property accesses requires a more advanced approach since properties are typically treated as simple values. To address this, we utilize JavaScript's getter pattern [6], which allows property access to behave normally while internally invoking a function that can execute custom code—such as alerting the user—during access.

E. Data Logging for Analysis

Another key component of our extension is its ability to log fingerprinting-related activity for post- analysis. This feature is implemented using three components:

1. An AJAX request embedded within the Chrome extension
2. cA backend server to receive incoming data
3. A MongoDB database for persistent data storage

F. AJAX Integration in the Chrome Extension

AJAX, which stands for Asynchronous JavaScript and XML, is employed to send data to the server without interrupting other browser processes [7]. Within the overwritten functions, we include a fetch call that transmits relevant information to the backend. Below is an example of the code that performs this asynchronous request:

```
javascript CopyEdit
fetch('http://127.0.0.1:3000/analysis', { method: "POST",
headers:{
'Content-Type':'application/json'
},
body:JSON.stringify({

"site":window.location.origin, "fnCall": fnName
})
})
.then(data=>console.log(data));
```

This request sends the origin of the site and the name of the function being accessed to a local server running on port 3000.

G. Backend Server Setup and CORS Configuration

CORS (Cross-Origin Resource Sharing) is an HTTP-based protocol that enables a server to permit resources to be requested from domains other than its own [8]. Since the extension collects data from multiple websites, it is essential that the backend server supports requests from all origins. To achieve this, the server must explicitly set appropriate CORS headers to avoid the browser blocking the request due to cross-origin policies.

Once data is successfully received from the browser extension, the backend server processes and stores it in a MongoDB database. This storage enables further analysis of finger printing attempts and the effectiveness of the extension's interventions.

H. Automated Data Collection

With the necessary setup complete, we proceeded to gather statistical data. To automate the process of loading websites, we utilized a Python-based Selenium Chrome driver. Selenium is a widely-used web automation tool designed for testing web applications .It offers a comprehensive API that can mimic user interactions with a browser.

In our case, we used Selenium solely to automate the task of opening a designated webpage and activating the browser extension we developed .Once the extension is loaded, it monitors specific JavaScript function calls. If any of the targeted fingerprinting functions are triggered, the extension sends the relevant data to the backend server for logging. The overall architecture of our data collection system is illustrated in Figure 4.

IV. EXPERIMENTAL FINDINGS

A. Website Fingerprinting Techniques

This section outlines the fingerprinting methods we implemented for various browser features, along with a brief evaluation of their effectiveness:

1) User-Agent String:
The browser's user-agent can be retrieved through the navigator object using navigator.userAgent. This method is straight forward and consistently reliable for identifying the browser and operating system.

2) Browser Plugins:
Plugin details can also be accessed through the navigator object via Navigator.plugins. While effective, it should be noted that some modern browsers limitor obfuscate this data for privacy reasons.

3) Time Zone Detection:
The user's time zone is obtained using JavaScript's Intl API. Specifically, Intl.DateTimeFormat().resolvedOptions ().timeZone provides an accurate and consistent result across major browsers.

4) Screen Dimensions and Color Depth:
The screen object provides information about the user's screen resolution and color depth. Properties such as screen.width, screen.height, and screen.colorDepth allow us to collect these metrics effectively.

5) System Font Enumeration:
To detect available system fonts, we dynamically create a element and populate it with sample text. We then loop through a predefined list of fonts—sourced from Windows 10 and macOS font libraries [9], [10]—and apply each one to the element. If the dimensions of the text change from a baseline value, the font is assumed to be installed on the system. This approach allowed us to successfully identify most fonts also detected by CoverYourTracks.

6) Cookie Availability:
Whether cookies are enabled in the browser can be determined using the navigator.cookieEnabled property, making this a simple yet informative fingerprinting attribute.

7) WebGL Vendor and Renderer:
To obtain details about the GPU vendor and renderer, we first insert a canvas element into the DOM using JavaScript. Then, by invoking the getContext("webgl") method, we access the WebGLRenderingContext, from which vendor and renderer strings can be extracted using specific extensions.
The "webgl" context, when obtained using the getContext("webgl") method, allows access to GPU-specific information. This is achieved through functions like getParameter() and getExtension(), which return values such as the GPU vendor and renderer.

8) Do-Not-Track Header:
The browser's support for user tracking preferences can be determined using navigator.doNotTrack, which reveals whether the "Do Not Track" header is enabled.

9) Language Settings:
User language preferences are accessible through navigator.languages, which returns an array of supported languages in order of preference.

10) Platform Detection:
Although navigator.appVersion does not explicitly state the platform, parsing the returned string can often reveal the underlying operating system.

11) Touch Capability:
To detect whether a device supports touch input, multiple indicators within the window, navigator, and window.navigator objects can be queried.

12) Processor Thread Count (Hardware Concurrency):
The number of logical processor threads available to the browser can be identified using navigator.hardwareConcurrency.

13) Device Memory:
Approximate available device memory can be accessed via navigator.deviceMemory.

14) Network Information (Not Reported by CoverYourTracks):
Network-related metrics, such as connection type and estimated bandwidth, are accessible through the navigator.connection interface.
Most of these attributes can be obtained using straightforward JavaScript queries, making them relatively easy to exploit for fingerprinting purposes.

B. Built-In Browser Protections

Table I summarizes the defensive mechanisms implemented by various browsers to counteract fingerprinting. Our evaluation revealed substantial variation in the scope of protections provided by different browsers. Notably, Google Chrome, which has the largest market share, exposes nearly all the fingerprinting-related attributes by default—including fonts, language preferences, WebGL details, and audio configurations.
In contrast, Brave and Tor Browser demonstrate comprehensive protections across the majority of fingerprintable vectors. This suggests that browsers designed with privacy in mind offer significantly better resistance to tracking techniques. Ultimately, the browser choice can have profound implications on user privacy.

C. Evaluation of the Anti- Fingerprinting Extension

We deployed our Chrome extension on several high-traffic websites, as ranked by Tranco [11], to evaluate its practical usability and identify commonly accessed fingerprinting APIs. Additionally, we tested the extension on the CoverYourTracks platform. The results indicated that the extension successfully randomized the browser fingerprint—an outcome typically not observed with the default settings of Chrome.
Interestingly, the fingerprint obfuscation achieved through our extension matched the behavior of privacy-centric browsers like Brave, confirming that JavaScript-level intervention can meaningfully reduce trackability evenon mainstream browsers.

D. Usability Assessment

By leveraging our database to store data collected through the Selenium- driven automation setup, we analyzed the impact of our browser extension on browsing stability. Specifically, we compared the frequency of errors encountered while navigating websites with and without the extension enabled.
Our observations revealed that, in over 50% of the cases, the extension had no noticeable effect on the number of errors

generated. Interestingly, around 9% of websites exhibited a reduction in errors when the extension was active. However, we attribute this to natural fluctuations during page loading or the possibility that an early error may have suppressed subsequent ones.

Although tracking error types was feasible, correlating specific errors with the extension's randomized data proved challenging. This is primarily because the observed errors typically originated in parts of the code base unrelated to the fingerprinting modifications.

E. Case Study: Impact on Video Conferencing Platforms

While our general usability study provided insight into error occurrence rates, it did not reflect the severity or functional implications of those errors. To investigate this further, we used the extension during regular, day-to-day browsing activities.

For most sites, even when minor errors were logged, the overall functionality remained intact. However, significant issues arose with platforms that support video communication—such as Google Meet and Blue Jeans. When the extension was active, video conferencing features often failed to operate as expected. Although we attempted to trace the root causes, the connection between the randomized fingerprinting properties and the video failures remained inconclusive.

F. Case Study: Audio Buffer API Usage

We also examined the prevalence of various fingerprinting-related APIs among top-ranked websites. One such API, Audio Buffer. get Channel Data, appeared infrequently but is known to play a central role in audio fingerprinting. This method enables access to raw audio data prior to playback, allowing subtle hardware- driven differences in audio rendering to be measured.

Despite its fingerprinting potential, this API serves legitimate use cases— particularly in multimedia applications. Its presence on video-heavy websites suggests that its usage may be aligned with its intended audio processing functionality, rather than explicitly for tracking purposes. As with many browser APIs, distinguishing between privacy-invasive and benign use remains context-dependent.

V. DISCUSSION

One of the most striking insights from our research was the relative simplicity involved in implementing browser fingerprinting techniques

TABLE I
COMPARISON OF BROWSER FINGERPRINTING DEFENSES

Data Type	Chrome	Firefox	Brave	Tor
Fonts	Reveal	Only Default Font (Disable)	Default Font (Disable)	Default Font (Disable)
Language	Reveal	Reveal	Reveal	Reveal
WebGL	Reveal	Reveal	Anonymous	Anonymous
Audio	Reveal	Random	Random	Permission
Hardware Info	Reveal	Reveal	Random	Anonymous
Browser Plugin	Reveal	Anonymous	Random	Anonymous

TABLE II
NUMBER OF NEW ERRORS CAUSED WHEN BROWSING TOP WEBSITES WITH OUR EXTENSION ENABLED

# JS Errors Introduced	# Websites (N=228)
Decreased Errors	20 (9%)
No New Errors	128 (56%)
Between 1 and 10	34 (15%)
Between 11 and 100	20 (9%)
Greater than 100	26 (11%)

TABLE III
TOP WEBSITES THAT USE THE AUDIOBUFFER-GETCHANNELDATA API, AS WELL AS THE WEBSITE CATEGORY.

Website	Categories
www.yy.com	Streaming Media & Downloads
www.bloomberg.com	News
www.sciencedirect.com	Education
www.pornhub.com	Pornography/Sexually Explicit
www.bilibili.com	Streaming Media & Downloads
www.taboola.com	Business
www.zhihu.com	Forums & Newsgroups
www.binance.com	Computers & Technology
www.prnewswire.com	News

Through our experimentation, we discovered that JavaScript-based fingerprinting scripts are readily accessible and easy to integrate into a webpage. Tools such as **FingerprintJS** provide streamlined APIs that enable developers to effortlessly generate and manage unique user identifiers. While the extent of fingerprinting across the web remains uncertain, it is evident that most websites face few technical barriers in collecting such data— particularly those already familiar with using cookies for user tracking.

During our evaluation, we also realized that privacy-enhancing tools can negatively affect user experience. However, measuring the precise impact of these usability issues remains a complex challenge due to their subjective and context-dependent nature.

In addition, even if current fingerprinting techniques are successfully mitigated, the rapid evolution of tracking methods raises concerns about the longevity of such protections. For instance, a novel tracking method using **favicons**— which leverages the browser's rarely cleared favicon cache—was recently introduced [12], demonstrating how seemingly innocuous browser features can be exploited for persistent tracking.

This highlights the adversarial and ever-evolving nature of digital privacy, where each Defence is often met with a new offensive technique, forming an ongoing cycle between trackers and privacy advocates.

a dynamic cycle of innovation between privacy advocates and trackers, with both sides continuously developing new methods to outpace the other.

Emerging advancements in advertising technologies may offer a more privacy- conscious future. For instance, the **Brave browser** has introduced its own advertising model that leverages on- device machine learning to serve ads while minimizing the amount of personal data shared with advertisers [13]. Similarly, Google Chrome is advancing a concept known as the **Privacy Sandbox** [14], which aims to reduce data leakage by retaining more user information locally on the user's device. Additionally, Chrome is exploring the **Federated Learning of Cohorts (FLoC)** framework [15], which enables advertisers to target groups with shared characteristics rather than individuals, thereby preserving user anonymity.

VI. FUTURE WORK

To enhance the reliability and practical usability of our browser extension, further exploration is needed to understand the root causes of the errors encountered during testing. However, identifying the exact source of such errors is challenging, as there is often a significant gap between the point where an error occurs and where it is reported in the code.

A more refined testing approach could involve toggling individual randomization settings during multiple browsing sessions. This would help isolate which randomized API calls are responsible for specific errors, providing a clearer picture of which fingerprinting Defences most affect functionality.

Moreover, improving user interaction with the extension could significantly enhance its adoption. By enabling the extension to intelligently infer whether a particular piece of data is likely being used for fingerprinting, users could be alerted in real-time. This would allow them to approve or deny access to certain features—similar to permission prompts on mobile platforms.

Incorporating a decision model that assesses both the rarity of the API being called and the context in which it is used (e.g., the source of the JavaScript file) may enable the extension to make smarter, context- aware decisions regarding fingerprinting behaviour.

VII. CONCLUSION

In this study, we explored the current landscape of browser fingerprinting techniques and actively deployed in real-world scenarios. To better understand them, we developed a custom website capable of collecting fingerprinting data and demonstrated that the simplicity of these methods makes them easily implementable across a wide range of platforms. We then explored the defensive mechanisms modern browsers employ to safeguard users against fingerprinting.

Building upon these insights, we created a Chrome browser extension designed to randomize fingerprintable attributes. Our evaluation showed that the extension effectively disrupted fingerprinting attempts, as platforms like CoverYourTracks were unable to generate a consistent identifier for our test environment.

Lastly, we reflected on potential areas for enhancement in our extension and offered insights into the future direction of privacy-preserving technologies in the context of evolving advertising models.

REFERENCES

[1]. B. Clifton, Advanced Web Metrics with Google Analytics. John Wiley Sons, 2012.

[2]. G. Zhang, Gzhang315/cs6262, https://github.gatech.edu/gzhang315/cs6262.

[3]. S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Oct. 2016. doi:10.1145/2976749.2978313. [Online]. Available: https://www.cs.princeton.edu/~arvindn/publications/OpenWPM_1_million_site_tracking_measurement.pdf.

[4]. Google, Content scripts. [Online]. Available: https://developer.chrome.com/docs/extensions/mv2/content_scripts/.

[5]. Wikipedia, Canvas fingerprinting. [Online]. Available: https://en.wikipedia.org/wiki/Canvas_fingerprinting.

[6]. M. W. Docs, Getter. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/get.

[7]. Wikipedia, Ajax (programming). [Online]. Available: https : / / en . wikipedia . org / wiki / Ajax_(programming).

[8]. M. W. Docs, Cross-origin resource sharing (cors). [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[9]. Apple, System fonts. [Online]. Available: https://developer.apple.com/fonts/system-fonts.

[10]. Microsoft, Windows 10 font list. [Online]. Available: https://docs.microsoft.com/enus/typography/fonts/windows_10_font_list.