Browserprint: An Analysis of the Impact of Browser Features on Fingerprintability and Web Privacy

Seyed Ali Akhavani[1], Jordan Jueckstock[2], Junhua Su[2], Alexandros Kapravelos[2], Engin Kirda[1], and Long Lu[1]

[1] Northeastern University, Boston, MA, USA
{sadatakhavani.s,e.kirda,l.lu}@northeastern.edu
[2] North Carolina State University, Raleigh, NC, USA
{jjuecks,jsu6,akaprav}@ncsu.edu

Abstract. Web browsers are indispensable applications in our daily lives. Millions of users use web browsers for a wide range of activities such as social media, online shopping, emails, or surfing the web. The evolution of increasingly more complicated web applications relies on browsers constantly adding and removing features. At the same time, some of these web services use browser fingerprinting to track and profile their users with clear disregard for their web privacy. In this paper, we perform an empirical analysis of browser features evolution and aim to evaluate browser fingerprintability. By analyzing 33 Google Chrome, 31 Mozilla Firefox, and 33 Opera major browser versions released through 2016 to 2020, we discover that all of these browsers have unique feature sets which makes them different from each other. By comparing these features to the fingerprinting APIs presented in literature that have appeared in this field, we conclude that all of these browser versions are uniquely fingerprintable. Our results show an alarming trend that browsers are becoming more fingerprintable over time because newer versions contain more fingerprintable APIs compared to older ones.

Keywords: Browser Security · Fingerprinting · Privacy · Web Security

1 Introduction

Web browsers have become indispensable in our daily lives. The majority of the online activity of many Internet users comprises of using a browser to access social media, online shopping, surfing the web, messaging, and accessing stored information in the cloud. Unfortunately, many companies are interested in collecting the private browser activities of end-users for marketing and sales purposes. To achieve their data collection objectives, some web services use "browser fingerprinting" to track and profile their users with clear disregard for their web privacy.

As browsers increasingly supplant traditional operating systems as the application publishing platforms of choice, many unique details of a user's browser

such as its hardware, operating system, browser configuration and preferences can be exposed through the browser. An attacker who collects and sums these outputs can create a unique "fingerprint" for tracking and identification purposes. In addition, browsers have also been increasing in complexity as more and more new features are being integrated into them, raising concerns that the attack surface offered by this software "bloating" (i.e., the increase in the number of components and code not needed by every user) is contributing to making browsers more difficult to secure against attacks.

Browser fingerprinting has been determined to be an important problem by previous research (e.g., [23,4,20,7]) as well as browser vendors themselves (e.g., [8,28,22]). To date, however, no studies have looked at popular browsers historically and have attempted to determine how their fingerprintability has evolved over the years. Past work has demonstrated that the ability to simply fingerprint a browser's precise version without relying on possibly spoofed `User-Agent` strings can be useful to attackers [26]. In the further light of web privacy research showing the potential and/or real-world exploitation of novel APIs for fingerprinting [24,6,15], we consider the raw volume of implemented APIs to be a rough but useful proxy estimate of a browser's potential fingerprintability.

In this paper, we perform an empirical analysis of a large number of browser features that have been integrated or phased out of the popular Mozilla Firefox, the Google Chrome, and the Opera browsers between the years 2016 and 2020. We consider browser features to be all functionality that is available to attackers directly through JavaScript, since these are the root problem of most web attacks. Our aim is to answer a number of research questions about the *fingerprintability* and security of these browsers over this time period. We propose a new metric for quantifying the fingerprintability of browser versions that rely on the number of browser features that are associated with fingerprinting. This metric is based on previous research and current fingerprinting techniques discovered in the wild (see Section 3.2 for more details). By analyzing 33 Google Chrome, 31 Mozilla Firefox, and 33 Opera major browser versions, our results suggest that these popular browsers have unique feature sets that make them significantly different from each other. Hence, by comparing these features to the fingerprinting APIs presented in literature, we conclude that all of these browser versions are uniquely fingerprintable. Our results suggest the alarming trend that browsers are becoming more fingerprintable over time as newer versions of popular browsers have more fingerprintable APIs embedded in them.
This paper makes the following key contributions:

- We show that all major Mozilla Firefox, Google Chrome, and Opera browser versions between 2016 until 2020 are uniquely fingerprintable based exclusively on the presence or absence of browser features.
- We analyze Mozilla Firefox, Google Chrome, and Opera and report major differences between feature introduction and removal trends. While Firefox tends to keep a steady number of features in the browser (i.e., introducing new features while removing older ones), Chrome, in contrast, is growing and more features are kept as the browser evolves. Opera, similar to Chrome,

seems to be adding lots of features and not interested in removing the older ones.
- We show that although Google Chrome and Opera are both based upon Chromium and share the same codebase, there are still differences in their feature introduction and removal patterns. But this shared codebase makes them very similar in our fingerprintability analysis.
- We provide all the source code and datasets that we have collected in our experiments to the community.[3]

2 Research Questions

In this paper, by performing an automated analysis, we attempt to answer the following research questions:

1. *Are major versions of Firefox, Chrome, and Opera browsers fingerprintable?* Our results suggest that the feature set for each browser version is unique. There exist multiple APIs in every browser version that we have analyzed that can be used for fingerprinting. By extracting all the features supported by a browser and exposed via API calls, we can uniquely identify each browser version.
2. *Are Firefox, Chrome, and Opera becoming more fingerprintable over time?* One of the major conclusions of our study is that the number of APIs one can use in the newer versions of Chrome, Opera, and Firefox is larger than the older versions. Hence, newer browser versions are even more fingerprintable than previous versions, and our findings suggest that this trend is likely to continue. As a result, privacy might be an even more significant concern in the future for browser users.
3. *What "lifespan profiles" can we cluster browser features into? Are there any"permanently removed" features? If so, how does their life cycle look like?* Our results suggest that we can categorize browser features based on their lifespan into three main categories (i.e., persistent features, non persistent features, and recurring features). We observe that most of the features are added permanently, and are not removed over time – indicating that browsers are indeed becoming more "bloated" as they evolve.
4. *With respect to browser bloating, how does Firefox compare to Chrome and Opera?* In our study, we were able to map the number of unique features for major versions of Firefox, Chrome, and Opera. The results suggest that Chrome and Opera are introducing more features over time than Firefox, but that all of these browser vendors have shown a significant increase in the total number of features they support per version since 2016. Compared to Firefox, Chrome and Opera tend to introduce more new features and keep them around longer.
5. *Could the incognito mode in Chrome and the private window mode in Firefox and Opera reduce the possibility of being fingerprinted by websites?* Our

[3] `https://github.com/sa-akhavani/browserprint`

analysis suggests that the incognito and private window modes have negligible impact on reducing fingerprinting. That is, almost all fingerprinting APIs are accessible in these modes the same way that they are available in non-private mode.

6. *Although Opera and Chrome are both Chromium-based and share the same codebase, is there any noticeable difference between these two browsers in case of fingerprintability?* In our analysis, we found out that Opera and Chrome have very similar sets of fingerprintable APIs and there is not much difference between these two browsers in case of fingerprintability. But there exist differences in some browser-specific features between these two browsers. Additionally, Opera and Chrome follow almost the same pattern in feature adding and removal as a result of their shared codebase. These browsers tend to keep a majority of their features untouched.

3 Methodology

To be able to determine how fingerprintable a browser is, we need to determine the features it supports when a webpage is visited by a user. Similarly, we need to understand which features are supported by a specific version because attackers typically target such features in attacks (e.g., a bug in the video access functionality might be exploited). Hence, to answer the research questions we pose in this study, we need to be able to figure out exactly what features are supported by each browser version under analysis. In this section, we describe the methodology we followed in this work, and explain how we created the datasets we used in our analyses.

3.1 Feature Gathering

In order to collect *browser feature* sets from Firefox, Opera, and Chrome, we crafted a special JavaScript-instrumented webpage that analyzes the visiting browser. We use the term *feature* to describe JavaScript objects, methods, and property values built into the global namespace of the browser's JavaScript implementation (i.e., the `window` object). Clearly, this definition is JavaScript-centric. However, it is unambiguous and naturally scalable, as we can automate the collection of features from many different browser implementations using standard scripting and crawling techniques. When our instrumented page is loaded by the browser, our JavaScript is executed. This code probes and iterates through the features supported by the browser. This is done by using JavaScript to traverse the tree of non-cyclic JavaScript object references accessible from a pristine (i.e., unmodified by polyfills or other prototype-chain modifications) `window` object, and collecting the full feature names encountered during the traversal. Each feature name comprises the sequence of property names leading from the global object to a given built-in JavaScript value. The traversal code is careful to not modify this object (which doubles as the global variable namespace) in any way, to avoid contaminating the resulting set of feature names.

Captured feature sets are then stored in a database, tagged with identifying metadata such as the browser's User-Agent string.

We use the terms *browser features*, as defined in this section, and *JavaScript APIs* interchangeably in our work.

3.2 Browser Fingerprinting APIs

We conduct an in-depth analysis in order to determine which browser features are associated with fingerprinting. Our analysis generates a list of suspicious APIs that we use in our measurements in Section 4 to quantify *fingerprintability*: the ratio of browser features that are associated with fingerprinting techniques in a browser version. We describe in the following how we determine which browser features are related to browser fingerprinting.

Our list of suspicious browser fingerprinting APIs contains a total of 313 JavaScript APIs. These APIs are considered suspicious because the purpose of using these API depends on the intent of the programmer who writes the code. We call this list *suspicious fingerprinting APIs* in this paper. In Panopticlick's research [4], browser fingerprinting is achieved through a combination of APIs that seem innocent, such as `Navigator.plugins`, `Navigator.userAgent`, and `Screen.colorDepth`. These APIs provide functionality that matches their original objectives. However, they can be abused by creating a unique fingerprint of the client's browser due to exposing information that narrows down the diversity of visited users. We use two methods to assemble the list of fingerprinting APIs: literature review and experimental analysis.

Literature Review The foundation of the API list is composed of four core fingerprinting papers, Panopticlick [4], AmIUnique [1], Hiding in the Crowd [18], and FPDetective [7]. This analysis results in approximately 10% of the list of suspicious fingerprinting APIs. Some of the APIs are directly mentioned in these papers and the others are chosen to match standard APIs[4] with the same functionality. The concepts of Canvas, WebGL, and Font fingerprinting are introduced along with these APIs. These concepts lead to the next turn of investigation of papers which are Cookieless Monster [23] and Pixel Perfect [20]. This investigation does not bring more APIs but a direction to experimental analysis.

Experimental Analysis The experimental analysis consists of two stages, collecting APIs by crawling websites and extracting suspicious APIs from the crawling data. In terms of data collection, the workflow is the same as the one in VisibleV8 [19]. A customized crawler was driven to visit all websites in the Easylist [2] domain file that contains 13,241 domains. Then, the raw logs generated by VisibleV8 were gathered and the VisibleV8 post-processor was applied to process the raw data. After removing duplicate and non-standard APIs, the API usage of 8,682 domains with 56,828 origins was collected. Non-standard

[4] `https://developer.mozilla.org/en-US/docs/Web/API`

APIs indicate ones that are not listed in the WebIDL [5] data package. In other words, VisibleV8 and its post-processor were adopted to aggregate and summarize standard JS API usage of the target domains.

While collecting APIs from the wild, the API suspicious list was extended through crawling on `panopticlick.eff.org`, `amiunique.org`, and `browserleaks.com` websites. These websites are explicitly marked as browser fingerprinting websites. Therefore, augmenting suspicious fingerprinting APIs among these websites is more efficient than a random walk on the enormous JS API pool.

The next step is to perform a manual analysis to check every API utilized by these three websites. First, we search for information and usage of an API on Mozilla's MDN Web Docs [21]. Then, we determine whether an API fingerprints users based on the information the API conveys. That is to say, an API is classified as a suspicious fingerprinting API if it can provide the information to filter certain users out. For example, there are two users with distinct user agents. By calling `Navigator.userAgent`, the programmer should be able to distinguish between these two users. `Navigator.userAgent` can be recognized as a fingerprinting API in this case. The majority of suspicious fingerprinting APIs come from manual analysis and the idea of categorizing fingerprinting APIs is incited by the `browserleaks.com` website.

The last step is to manually search for more fingerprinting APIs with the keyword. Namely, in Canvas fingerprinting, most APIs include the "Canvas" or "CanvasRendering". A program was created to filtrate APIs that contain "Canvas" or "CanvasRendering" among APIs of 8k crawled domains. The same pattern also applies to BatteryManager, WebGLRenderingContext, and SpeechSynthesis. Meanwhile, the fingerprint2.js [16] was reviewed to supplement the suspicious fingerprinting API list.

There are limitations to the methods we used for constructing a suspicious fingerprinting API list. First and foremost, this list only provides a partial view of full fingerprinting APIs. To the best of our knowledge, there is no complete table of fingerprinting APIs and more research is needed in this direction. The second limitation is during the manual analysis. There could be misconceptions between the API usage provided by the Mozilla API page and the way programmers exploit them. Lastly, part of JS APIs is filtered out by the VisibleV8 post-processor. This can be improved by using a larger set of WebIDL data or precisely use the aggregated raw APIs.

As a service to the community, we have made our list of fingerprinting APIs publicly available.

3.3 Browser Testing Platform

In this work, we target Google Chrome, Mozilla Firefox, and Opera browsers as they are well-known, popular browsers that have millions of users. Firefox possesses a distinct codebase unlike Chrome and Opera which are both based on Chromium. We gathered a copy of every major Firefox, Chrome, and Opera

version that was released during the March 2016 to April 2020 timeframe, i.e., Chrome versions 49–81, Firefox versions 45–75, and Opera versions 36–68.

To individually connect each browser version to our instrumented feature gathering web application, we mainly used the BrowserStack web service [10]. BrowserStack is a cloud-based web and mobile testing platform that enables developers to test their websites and mobile applications across on a wide range of browsers, operating systems, and real mobile devices. If a specific browser version or configuration was not available on BrowserStack, we developed and used automation scripts to instrument and run the browser instances on a desktop computer running Windows 10.

4 Analysis

In this section, we describe the analysis we performed on the datasets that we collected, and the insights that we distilled from the analysis. We leverage the browser features dataset and the suspicious fingerprinting APIs dataset in our analysis.

4.1 Analysis of the Browser Features

The first analysis we performed on the dataset we collected was to understand how browser features have evolved over time. As we describe in Section 3, we consider *browser features* all functionality exposed to JavaScript as objects, methods, and property values. This definition of browser features reflects on 1) how attackers craft web attacks (i.e., creating a unique fingerprint using such features, or exploiting vulnerabilities) and 2) a measurable metric across browser versions. Understanding and gaining insights into how browsers are dealing with new as well as older features is important to be able to distill conclusions about how secure and fingerprintable browsers are becoming as they evolve. Hence, our analysis looked at specific browser features that were introduced, what the typical lifespan of features looks like.

After extracting feature information for all of the browsers under analysis, we automatically parsed the generated reports and analyzed them to see if the features in these browsers fall into specific categories. Our analysis suggested that the features in Firefox, Opera, and Chrome can be categorized into three main categories:

- **Persistent Features**: These are features that are added to a specific version, and that continue to exist in every version that is released after the feature was introduced. We consider a feature to be "persistent" if it appears in at least two distinct browser versions.
- **Non-Persistent Features**: These are features that existed in older versions of the browser, but were removed, and never appeared in newer versions of the browser again. We consider a feature to be "non-persistent" if it is absent in at least two distinct versions of the browser versions under analysis.

– **Recurring Features**: These are features that are added and removed from the browser from time to time. That is, they are introduced, they are removed, and they might appear again at some point. Such features are typically being tested by the vendors, and it is not clear if they will become persistent, or non-persistent.

Our analysis suggests that Chrome possesses 9,718 persistent, 711 non-persistent, and 3,161 recurring features that it supports. Similarly, Opera contains 9,674 persistent, 711 permanently removed, and 3,219 recurring features. On the other hand, Firefox supports 6,274 persistent, 809 non-persistent, and 115 recurring features. Note that Firefox, overall, supports significantly fewer features than Chrome and Opera. Also, our analysis suggests that Firefox, compared to Chrome and Opera, is keeping fewer features (i.e., they are removing more) over time. Figure 1 illustrates the feature categories for each browser vendor. It can be seen that Opera and Chrome are having similar patterns since lots of their features are related to Chromium which is their shared codebase. Besides, Chrome and Opera have a greater portion of recurring features compared to Firefox. This means that Chrome and Opera tend to do more experiments on adding and removing specific features through time.

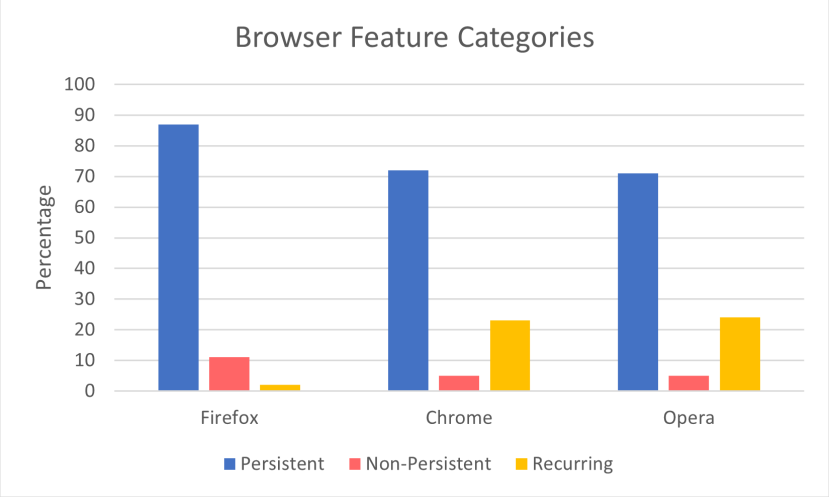

Fig. 1. Feature category distribution for browsers.

In this work, we also performed an analysis of the common features between Firefox, Chrome, and Opera. Since 2016, the total number of features introduced by these browsers is 15,945. Among all these features, there exist only 4,843 common features among Firefox and Chrome – which is approximately 30% of the total number of features that these vendors support. This number is the same between Firefox and Opera too, with 4,843 common features between them.

On the other hand, Chrome and Opera have a bigger set of common features. There exists 13,558 common features between Opera and Chrome – which is approximately 85% of the total number of features that these vendors support. The impact of this huge common features set on fingerprintability between two browsers are analyzed in the next section.

We can conclude that Firefox does not have a high feature overlap with Chrome and Opera. Note that although these browsers often offer very similar functionality, unsurprisingly, their codebase might be very different from each other. We are aware that Firefox's codebase is very different from Chrome's and Opera's. Hence the API names through which these features are available are also often significantly different. To the contrary, Chrome and Opera share the same codebase. This leads to having a bigger set of common features between these two browsers.

Figures 2 and 3 show the feature addition and removal trends for Firefox and Chrome. The data shows that Chrome is adding and removing many more features than Firefox in each version that is released if one looks at the overall numbers of features. However, Firefox seems to be more constant with respect to the number of new features added, and older features removed. Hence, Firefox seems to be more aggressive with respect to removing older features from the browser, "debloating" this way the browser. Chrome and Opera share the same trend, so we omit a separate figure for Opera and leave Figure 3 as a representative visualization of feature introduction and removal for Chromium-based browsers.

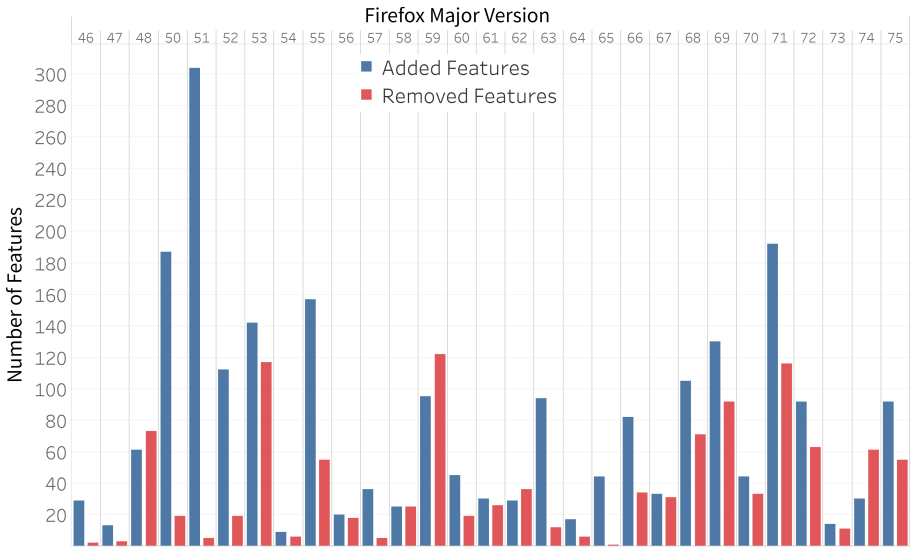

Fig. 2. Feature introduction and removal in Firefox.

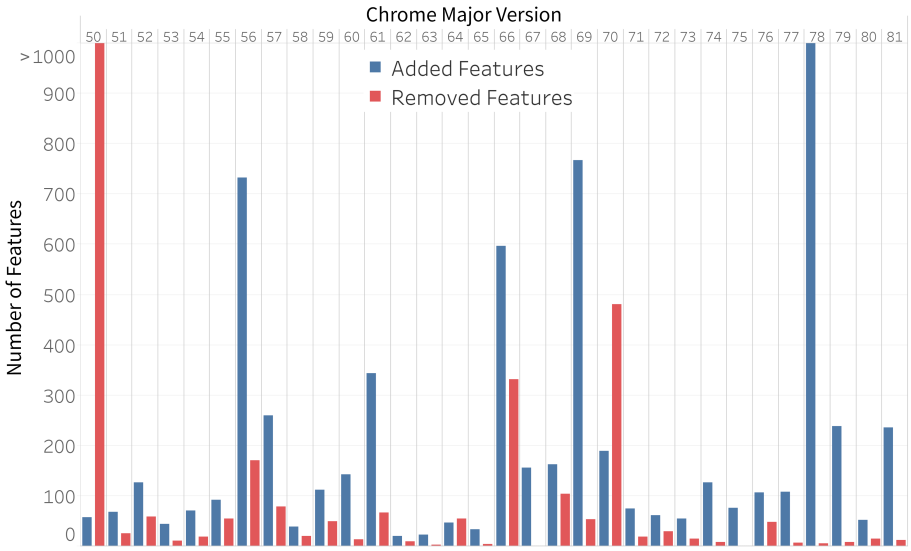

Fig. 3. Feature introduction and removal in Chrome.

By using the feature datasets we extracted from the Firefox, Opera, and Chrome versions, we compared feature trends for these browsers. The trends are depicted in Figure 4. The graph shows that the number of features supported by Firefox seems to be quite steady (i.e., if new features are added, some older ones are typically removed) while the number of features supported by Chrome and Opera is growing over time. Hence, the data suggests that Chrome and Opera are following differing browser feature development philosophies compared to Firefox.

4.2 Browser Fingerprintability

Analyzing fingerprinting API presence in Chrome, Firefox, and Opera Recall that one of the key research questions we asked at the beginning of this paper was if popular browsers such as Firefox, Chrome, and Opera are generally becoming more fingerprintable over time. In particular, we were also interested in answering if every browser version is unique in a fingerprintability sense.

Using the fingerprinting APIs that we collected (and described in Section 3), we aimed to determine how many of these APIs are available and active in specific browser versions. That is, we iterated through all the major Firefox and Chrome browser versions between 2016 and 2020, and tested their fingerprintability.

In Chrome 49 (i.e., the oldest Chrome version in our analysis), there exist 139 APIs from the suspicious fingerprinting APIs list. Which means they could be used for fingerprinting. In Chrome 81 (the newest Chrome version in our analysis), there exist 274 APIs from the suspicious fingerprinting APIs list. In short, the number of APIs that could be used for fingerprinting Chrome versions

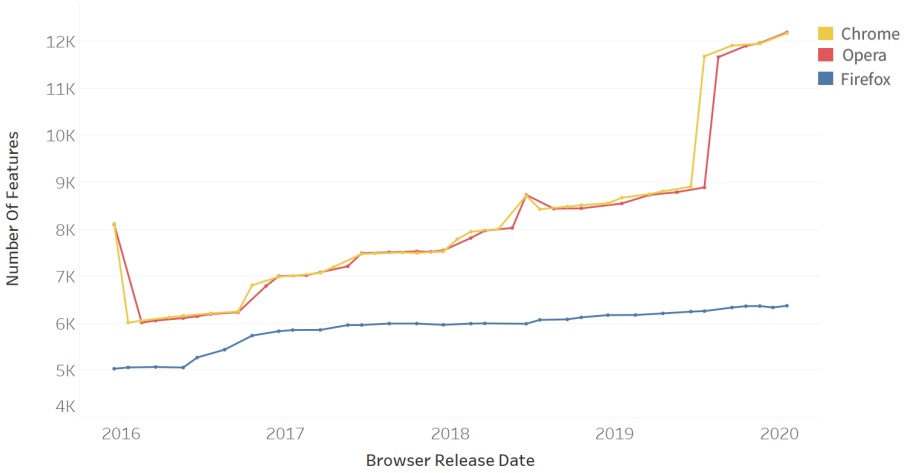

Fig. 4. Feature trends in Firefox, Opera, and Chrome when compared to each other.

are increasing over time. That is, the data suggest that Chrome is becoming easier to fingerprint as it evolves over time.

Compared to Chrome, Firefox 45 (i.e., the oldest version in our study) has 147 APIs from the suspicious fingerprinting APIs list. In contrast, Firefox 75 (which is the latest Firefox version in our study) has 271 fingerprinting APIs from the suspicious fingerprinting APIs list. Interestingly, though, Firefox 71 has 276 APIs from the suspicious fingerprinting APIs list. Our data analysis suggests that Firefox has become more fingerprintable over time, but that lately, although more features are added to it, its fingerprintability might have started to decline. In fact, Firefox has indeed started to take the fingerprinting problem seriously and has been increasingly taking steps to prevent it (e.g., [22]).

In addition, Opera 36 (i.e., the oldest version in our study) contains 139 suspicious fingerprinting APIs. On the other hand, Opera 68 (the latest Opera version in our measurement) is consist of 274 suspicious fingerprinting APIs. The trend is very similar to Google Chrome but there are minor differences at some points which could be seen in Figure 5.

Figure 5 depicts, in detail, the presence of fingerprinting APIs in Chrome, Firefox, and Opera that we measured. Note that in January 2017, there is a significant increase in the number of fingerprinting APIs that each browser supports. More than 100 fingerprinting APIs were added to both browsers. To determine what caused this spike, we investigated and analyzed the release notes of both Firefox 51 [17], Chrome 56 [13], and Opera 43 which is based on Chromium 56 [3].

The release notes indicate that HTML5 was enabled for all users by default in Chrome 56. As of this version, Adobe Flash Player was disabled and only allowed to run with specific user permissions. Chrome also enabled the WebGL 2.0 API that provides a new rendering context, and supports objects for the HTML5

Canvas elements. This context allows rendering using an API that conforms closely to the OpenGL ES 3.0 API[5]. Similarly, in Firefox 51, we observed that the browser had also added WebGL2 support during that time. The same happened to Opera 43 since Chromium 56 added WebGL2 support to its codebase.

When we analyzed our fingerprinting API list, we saw that the 107 new fingerprinting APIs that became possible as of this date were actually related to `WebGL2RenderingContext` which was added to Firefox 51, Chrome 56, and Opera 43. The straight-forward lesson to distill from our observation is that browser vendors need to be extra careful when they implement and release new features if they are interested in making their browsers more difficult to fingerprint.

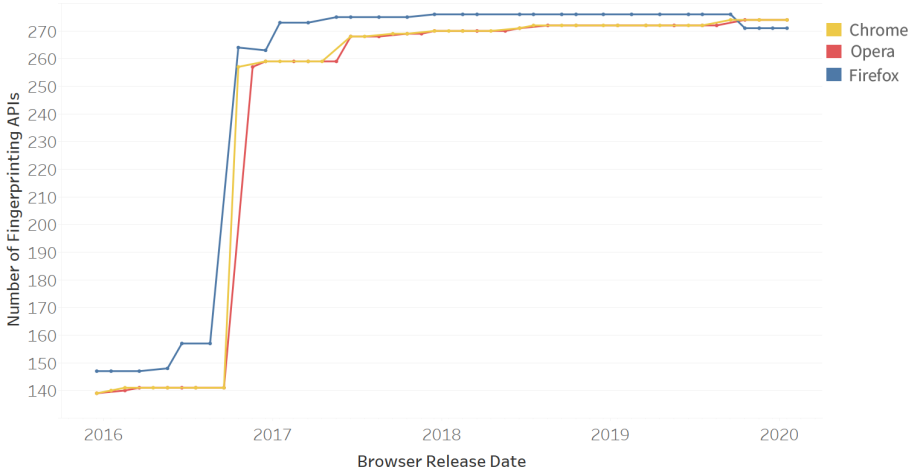

Fig. 5. Presence of Fingerprinting APIs in Chrome, Firefox, and Opera.

As part of our experiments, we also collected the feature sets for Firefox's Private Window, Google Chrome's Incognito, and Opera's Private Window. We measured the fingerprintability of the browsers in these modes. For Chrome, our results show that there is a small difference between the total number of features in regular mode versus the total number of features in incognito mode. For instance, Chrome 80's regular mode has 11,946 features while it has 11,936 features available in Incognito mode. The results were similar for Firefox's regular mode versus its Private Window Mode. For example, Firefox 75's regular mode has 6,370 total features while its Private Window Mode has 6,358 features available. Besides, Opera's private window had the same fingerprinting APIs compared to the regular mode and had zero impact on reducing fingerprintability.

Hence, we conclude that the incognito and private window modes do not help users against browser fingerprinting since every fingerprinting API that exists in

[5] `https://www.khronos.org/registry/webgl/specs/latest/2.0/`

a version's normal mode also appears in the same browser version's Incognito (or Private Window) mode.

Unique Feature Set In our analyses, we automatically deduced a "feature set" for each browser version that we analyzed. A feature set is a set of (i.e., the list of) browser features that exist in that specific browser version under analysis. When we compared the features sets for each browser version to each other (e.g., Firefox 54 versus 55), we observed that each feature set was unique for all the browser versions that we tested. That is, there exist no two browsers that possess the same feature set. Hence, from this observation, we can deduce that all the browser versions that we analyzed are uniquely fingerprintable.

The reason why the feature sets are unique among different browser versions is that each browser, as we described before, have recurring as well as non-persistent features. As a result, the fact that vendors continuously add, remove, and sometimes re-add features into their browsers also make them more fingerprintable.

One interesting trend is that the differences between the feature sets of Chrome, Firefox, and Opera in their newer versions is becoming smaller. That is, we observed much more intersections with each other than in older versions. Our data suggest that the feature sets for all Firefox, Chrome, and Opera are converging towards homogeneity of browser features.

5 Related Work

Our work focuses on the intersection of browser evolution and browser fingerprinting.

Browser evolution The first web browser, WorldWideWeb [9], was developed in 1990 by Tim Berners-Lee. That browser did not have JavaScript, did not support cookies and users could not adapt their browser with extensions. All these features and thousands more were introduced in browsers over time, matching the needs of the ever-evolving web.

Synder et al. [27] use a similar method to us to collect browser features by using the web API and extracting different kinds of JavaScript functions. They measure browser feature usage among Alexa's popular websites and also how many security vulnerabilities have been associated with related browser features. However, they do not aim to measure fingerprintability of different browsers which is one of the main goals of our paper. In another work by Snyder [29], a cost-benefit approach to improving browser security was conducted. Our work focuses on how browsers have become more fingerprintable over time based on the features they introduce, taking a new perspective on the privacy and security costs that the browser evolution brings.

Recent work has focused on methods to automatically reduce the functionality of the browser at the binary level. Chenxiong et al. [12] propose a debloating framework for the browser that removes unused features. Our work is complementary to debloating efforts of the browser, as we focus on which browser

features affect the users' privacy the most. Also, our work suggests that the debloating of browsers might not really be necessary as there does not seem to exist a correlation between the number of features added to the browsers over time, and how insecure they become.

Browser fingerprinting There have been a number of studies on browser fingerprinting and browser bloating. The first large-scale study on browser fingerprinting was conducted by Eckersley [14]. Eckersley showed that a wide range of properties in a user's browser and the installed plugins can be combined to form a unique fingerprint. His study made us eager to see what is happening in the world of browser features, and to try to analyze the impact of different browser features on creating unique user fingerprints.

Browser fingerprinting can be done by using different methods. Cao et al. [11] created user fingerprints by using OS-level features from screen resolution to the number of CPU cores. They also measure the uniqueness of different browser types by analyzing its OS-level features.

Olejnik et al. [25] show that one way of fingerprinting a browser is using web history. In this method, there is no need for a client-side state. However, note that this method is no longer possible because browser vendors have fixed this issue and (i.e., extracting user history is not possible as before).

Nikiforakis et al. [23] showed how tracking has moved from using cookies (stateful) to browser fingerprinting (stateless) on the web. Mowery et al. [20] demonstrated how the `canvas` HTML5 feature can be abused for browser fingerprinting based on the differences in rendering images on different GPUs. Starov et al. [30] measured how bloated browser extensions are in terms of the artifacts that they inject in visited pages, and can be used to identify the presence of the users' installed extensions. Trickel et al. [31] proposed a defense mechanism against identifying installed browser extensions in users' browsers based on artifacts that reveal their presence on the visited pages.

In light of the prior research on browser fingerprinting, our aim was to collect data and analyze the trends, and to see whether we are becoming better at managing browser fingerprinting (or if this privacy issue is becoming worse as new features are being introduced in new browser versions).

6 Conclusion

The evolution of the web relies on browsers adding new features that drive innovation in web applications. Yet, this innovation comes at a significant cost to the end users' privacy, since browser fingerprinting techniques abuse certain browser features. In this paper, we analyzed the impact of browser features on browser fingerprinting. We investigated more than 30 major browser versions for Google Chrome, Mozilla Firefox, and Opera between 2016 and 2020.

First, we extracted every browser feature that existed in these browser versions using the browser APIs. Then, we analyzed the feature sets for these browsers and compared them. One key observation was that the feature numbers

are overall increasing in modern browsers, and they are indeed becoming more "bloated" in general.

Next, we compared the feature reports for these browsers to the already listed fingerprinting APIs in browsers that are presented in the literature. Our findings suggested that each browser version between 2016 and 2020 was uniquely fingerprintable, and that the fingerprintablity of the browsers has been increasing over the years.

We envision our research to affect how browser vendors introduce new features and take into consideration the effects that these have on browser fingerprintability. Our goal is to highlight the concerning trend of "bloating" in the browser and encourage browser vendors to remove abused features in order to improve privacy on the web.

Acknowledgment

We thank the anonymous reviewers for their helpful feedback. This work was supported by the National Science Foundation under grants CNS-1703454, CNS-1703375 and CNS-2047260, by the Office of Naval Research under grant N00014-17-1-2541 and partially supported by Secure Business Austria.

References

1. AmIUnique, `https://amiunique.org`, [Online; accessed 20. June. 2021]
2. EasyList, `https://easylist.to/`, [Online; accessed 20. June. 2021]
3. Opera version history. `https://help.opera.com/en/opera-version-history/`, [Online; accessed 30. Jun. 2021]
4. Panopticlick, `https://panopticlick.eff.org`, [Online; accessed 10. Jan. 2021]
5. WebIDL Level 1, `https://www.w3.org/TR/WebIDL-1/`, [Online; accessed 20. Jul. 2021]
6. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Diaz, C.: The web never forgets: Persistent tracking mechanisms in the wild. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (2014)
7. Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., Preneel, B.: FPDetective: Dusting the Web for Fingerprinters. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. p. 11291140. CCS '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516674, `https://doi.org/10.1145/2508859.2516674`
8. Apple: Safari Privacy Overview. `https://www.apple.com/safari/docs/.pdf` (2019)
9. Berners-Lee, T.: The WorldWideWeb browser. `https://www.w3.org/People/Berners-Lee/WorldWideWeb.html` (1990)
10. BrowserStack: App & Browser Testing Made Easy. `https://www.browserstack.com/` (2021)
11. Cao, Y., Li, S., Wijmans, E.: (Cross-)Browser Fingerprinting via OS and Hardware Level Features (01 2017). https://doi.org/10.14722/ndss.2017.23152

12. Chenxiong, Q., Koo, H., Oh, C., Kim, T., Lee, W.: Slimium: Debloating the Chromium Browser with Feature Subsetting. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2020)
13. Chrome, G.: New In Chrome 56 | Web. https://developers.google.com/web/updates/2017/01/nic56 (2017), [Online; accessed 20. Jun. 2021]
14. Eckersley, P.: "How Unique Is Your Web Browser?". In: Atallah, M.J., Hopper, N.J. (eds.) Privacy Enhancing Technologies. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
15. Englehardt, S., Narayanan, A.: 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 (2016)
16. fingerprintjs: fingerprintjs, https://github.com/fingerprintjs/fingerprintjs, [Online; accessed 15. Jul. 2021]
17. Firefox, M.: Firefox 51.0, See All New Features, Updates and Fixes. https://www.mozilla.org/en-US/firefox/51.0/releasenotes/ (2017), [Online; accessed 20. Jun. 2021]
18. Gómez-Boix, A., Laperdrix, P., Baudry, B.: Hiding in the Crowd: An Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In: Proceedings of the 2018 World Wide Web Conference. p. 309318. WWW '18, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE (2018). https://doi.org/10.1145/3178876.3186097, https://doi.org/10.1145/3178876.3186097
19. Jueckstock, J., Kapravelos, A.: VisibleV8: In-browser Monitoring of JavaScript in the Wild. In: Proceedings of the ACM Internet Measurement Conference (IMC) (Oct 2019)
20. Mowery, K., Shacham, H.: Pixel perfect: Fingerprinting canvas in HTML5. Proceedings of W2SP (2012)
21. Mozilla: MDN Web Docs - Web APIs. https://developer.mozilla.org/en-US/docs/Web/API
22. Mozilla: How to block fingerprinting with Firefox . https://blog.mozilla.org/firefox/how-to-block-fingerprinting-with-firefox/ (2020)
23. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting. In: Proceedings of the IEEE Symposium on Security and Privacy (2013)
24. Olejnik, L., Englehardt, S., Narayanan, A.: Battery Status Not Included: Assessing Privacy in Web Standards. In: Proceedings of the International Workshop on Privacy Engineering (IWPE) (2017)
25. Olejnik, ., Castelluccia, C., Janc, A.: Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns (07 2012)
26. Schwarz, M., Lackner, F., Gruss, D.: JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In: NDSS (2019)
27. Snyder, P., Ansari, L., Taylor, C., Kanich, C.: Browser Feature Usage on the Modern Web. In: Proceedings of the Internet Measurement Conference (IMC) (2016)
28. Snyder, P., Livshits, B.: Brave, Fingerprinting, and Privacy Budgets. https://brave.com/brave-fingerprinting-and-privacy-budgets/ (2019)
29. Snyder, P., Taylor, C., Kanich, C.: Most websites don't need to vibrate: A cost-benefit approach to improving browser security. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (2017)
30. Starov, O., Laperdrix, P., Kapravelos, A., Nikiforakis, N.: Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In: Proceedings of the World Wide Web Conference (WWW) (2019)

31. Trickel, E., Starov, O., Kapravelos, A., Nikiforakis, N., Doupe, A.: Everyone is Different: Client-side Diversification for Defending Against Extension Fingerprinting. In: Proceedings of the USENIX Security Symposium (2019)