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

Anonymous Author(s)

ABSTRACT

Web browsers have become important tools in our daily lives. Millions of users use web browsers for different purposes such as social media, online shopping, or surfing the web. Some of these services use browser fingerprinting to track and profile their users which can be in contrast with their web privacy. At the same time, browsers are also being targeted by attackers because they are attractive platforms to compromise.

In this paper, we perform an empirical analysis of a large number of browser features intending to evaluate fingerprinting possibility, and also vulnerability issues that are based on different browser features. By analyzing 33 Google Chrome and 31 Mozilla Firefox 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 the 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 have more fingerprintable APIs in them. Another key discovery of our analysis is that unlike the popular belief about software bloating causing an increased attack surface, the number of vulnerabilities in browsers are not directly related to the number of new features being introduced by those browsers. In fact, our measurements suggest that browsers are becoming more secure over time although they are supporting more features than in the past.

ACM Reference Format:
Anonymous Author(s). 2020. Browserprint: An Analysis of the Impact of Browser Features on Fingerprintability and Browser Security. In *Proceedings of ACM Conference (Conference'20).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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, as browsers have become critical and important infrastructure components in our lives, they have also become attractive targets for attackers who are looking into compromising them. At the same time, 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. Clearly, browser fingerprinting can be in stark contrast with the web privacy of browser users.

As browsers are becoming increasingly entwined with operating systems, 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., [3, 5, 19, 22]) as well as browser vendors themselves (e.g., [6, 15, 25]). To date, however, no studies have looked at popular browsers historically and have attempted to determine how their fingerprintability has evolved over the years. Analogously, we are also not aware of studies that have looked into how, and if, the increase in the number of new features integrated into the browsers every year have contributed to an increase in the number of reported vulnerabilities.

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 and the Google Chrome 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 based on previous research and current fingerprinting techniques discovered in the wild (see Section 3.2 for more details). By analyzing 33 Google Chrome and 31 Mozilla Firefox major browser versions, our results suggest that these popular browsers have unique feature sets that make them significantly different from each other. Hence,

Conference'20, December 2020, Austin, TX, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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. Another key finding of our study is that unlike the popular folk wisdom that software bloating always leads to more security vulnerabilities [18], the number of vulnerabilities in browsers are not directly correlated to the number of new features being supported by those browsers. In fact, our measurements suggest that browsers are becoming more secure over time even though they are supporting more features and are becoming more "bloated".

This paper makes the following key contributions:

- We show that all major Mozilla Firefox and Google Chrome browser versions between 2016 until today are uniquely fingerprintable.
- We analyze both Mozilla Firefox and Google Chrome 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.
- As Firefox and Chrome are becoming more "bloated" over time and as their sizes increase, there is no direct correlation between the total number of features in the browsers and the number of vulnerabilities that are reported.
- We provide all the datasets that we have collected in our experiments to the community at [redacted for submission].

The rest of the paper is organized as follows: The next section lists the research questions we aimed to answer in this study. Section 3 describes our methodology and data gathering techniques. Section 4 presents a different analysis based on the feature reports and their relation to browser vulnerability. In Section 5, we present the related work and then briefly conclude the paper in Section 6.

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 and Chrome browsers fingerprintable?* Our results suggest that the feature set for each Firefox and Chrome 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 using API calls, we can indeed uniquely identify each browser version.

(2) *Are Firefox and Chrome browsers 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 and Firefox is larger than in 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 did 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?* In our study, we were able to map the number of unique features for major versions of Firefox and Chrome. The results suggest that Chrome is introducing more features over time than Firefox, but that both browser vendors have shown a significant increase in the total number of features they support per version since 2016. Compared to Firefox, Chrome tends to introduce more new features and keep them around longer.

(5) *Is there a correlation between the number of features available in a browser (i.e., how "bloated" the browser is) and the number of vulnerabilities that exist on that browser?* Our data suggest that, unlike the widely held belief, there is no direct correlation between the number of features that a browser version supports and the number of vulnerabilities in that version. Although browsers are indeed becoming more "bloated" over time, at the same time, their codebase seems to be becoming more secure.

3 METHODOLOGY

To be able to determine how fingerprintable a browser is, we need to determine the features it supports when a web page is visited by the user. Similarly, we need to understand what 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, and we also need access to vulnerability data for each browser. 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 "feature" sets from Firefox and Chrome, we redirect the browser under analysis to a JavaScript-instrumented web page. 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. That is, we can automate the collection of features from many different browser implementations using standard scripting and crawling techniques. When the instrumented page is loaded by the browser, our JavaScript is executed that then 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) `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. We use this list of suspicious APIs in our measurements in Section 4 to quantify *fingerprintability*: the ratio of browser features in a browser version that are associated with fingerprinting techniques. We describe in the following how we conducted the list of suspicious browser features that are related to browser fingerprinting.

Our list of suspicious browser fingerprinting API contains a total of 313 JavaScript APIs. These APIs are considered suspicious because the purpose of using these API depends on the programmer who writes the source code. In Panopticlick's research [3], browser fingerprinting is achieved through a combination of APIs that seem innocent, such as `Navigator.plugins`, `Navigator.userAgent`, and `Screen.colorDepth`. These APIs are designed with their original objectives. However, they are chosen to fingerprint browsers due to their alternative functionality in collecting information to narrow down the scope of visited users. Based on our approach taken to collect APIs, there is no way to determine whether the source code is doing browser fingerprinting without the acknowledgment of the writer of the code.

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 [3], AmIUnique [1], Hiding in the Crowd [16], and FPDetective [5]. This analysis resulted in approximately 10% of the list of suspicious fingerprinting APIs. Some of the APIs are directly mentioned in these papers and the others are modified to match standard APIs[1] with the same functionality. The concepts of Canvas, WebGL, Font fingerprinting are introduced among these APIs. These concepts lead to the next turn of investigation of papers which are Cookieless Monster [22] and Pixel Perfect [19]. This investigation does not bring more APIs but a direction to experiment analysis.

The experiment analysis consists of two stages, collecting APIs by crawling websites and extracting suspicious APIs from the crawling data. In terms of data collecting, the workflow is the same as the one in VisibleV8 [17]. 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 VV8 were gathered and the VV8 post processor was applied to process the raw data. After removing duplicate and non-standard APIs, the APIs usage of 8,682 domains with 56,828 origins was collected. Non-standard APIs indicate ones that are not listed in the WebIDL [4] data package. In other words, VV8 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 implement a manual analysis to check every API utilized by these three websites. First, we search for information and usage of an API on https://developer.mozilla.org/en-US/docs/Web/API. Then, 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 comes from the manual analysis and the idea of categorizing fingerprinting APIs is incited by 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 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 there could be research in this direction. The second limitation is during the manual

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

analysis. There could be misconceptions between the API usage provided by Mozilla web APIs page and the way programmers exploit them. Lastly, part of JS APIs is filtered out by the VV8 post processor. This can be improved by using a larger set of WebIDL data or precisely use the aggregated raw APIs.

We plan to make our list of fingerprinting APIs publicly available upon publication.

3.3 Browser Testing Platform

In this work, we decided to target Google Chrome and Mozilla Firefox browsers as they are well-known, popular browsers that have millions of users. Also, these browsers possess distinct codebases (i.e., unlike Microsoft Edge that is based on Google Chrome). We gathered a copy of every major Firefox and Chrome version that was released during the March 2016 to April 2020 timeframe (i.e., Chrome versions 49 to 81, and Firefox versions 45 to 75).

To individually connect each browser version to our instrumented feature gathering web application, we mainly used the BrowserStack web service [8]. 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.

3.4 Vulnerability Information Gathering

One major source of information for security vulnerabilities is the CVE (Common Vulnerabilities and Exposures) dataset that is hosted by MITRE. A CVE entry contains several fields with standardized text/definitions that represent a publicly disclosed cybersecurity vulnerabilities and exposures. Each CVE entry has a unique CVE identifier, a general description, and several references to one or more external information sources of vulnerability.

For our study, we used the CVE data from the National Vulnerability Database (NVD) that is provided by the National Institute of Standards and Technology (NIST). For each CVE entry in this dataset, we extracted the description, the affected product its specific version number, and the severity of the vulnerability. Then, we parsed this data and generated a CVE entry list for each browser version in our dataset. This new dataset was the basis of the vulnerability and feature analysis in this paper.

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.

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 Chrome and Firefox 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 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 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 9482 persistent, 711 non-persistent, and 3,397 recurring features that it supports. In contrast, Firefox supports 6,237 persistent, 809 non-persistent, and 152 recurring features. Note that Firefox, overall, supports a less number of features than Chrome. Also, our analysis suggests that Firefox, compared to Chrome, is keeping fewer features (i.e., they are removing more) over time. Figures 1 and 2 illustrate the feature categories for each browser vendor.

In this work, we also performed an analysis of the common features between Firefox and Chrome. Since 2016, the total number of features introduced by both Firefox and Chrome is 15,945. However, there exist only 4,843 common features among them – which is approximately %30 of the total number of features that these vendors support. Hence, we can conclude that Firefox and Chrome do not have a high overlap of the features that they support. Note that although these browsers often offer very similar functionality, unsurprisingly, their codebases are very different and the APIs though which

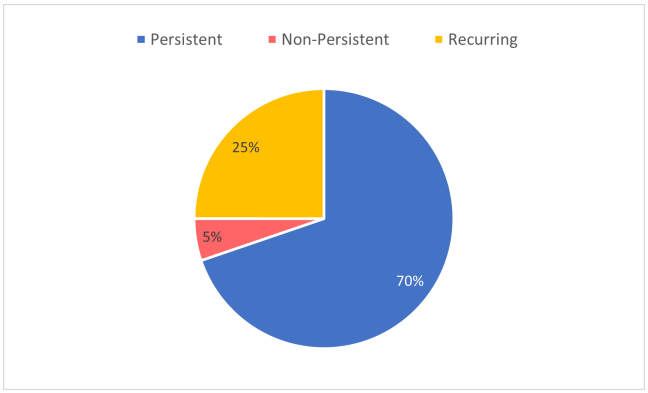

Figure 1: Feature category distribution for Chrome.

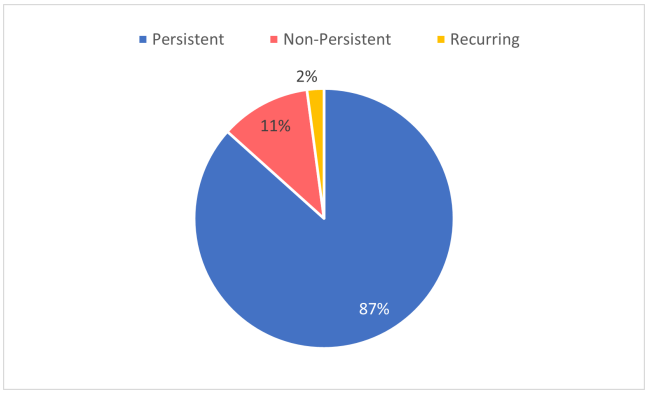

Figure 2: Feature category distribution for Firefox

these features are available are also often significantly different. As a result, it is clear that vulnerabilities in these browsers will be very specific to the version and vendor.

Figures 3 and 4 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, and hence, "debloating" the software.

By using the feature datasets we extracted from the Firefox and Chrome versions, we compared feature trends for both browsers. The trends are depicted in Figure 5. 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 is growing over time. Hence, the data suggests that Chrome and Firefox are following differing browser feature development philosophies.

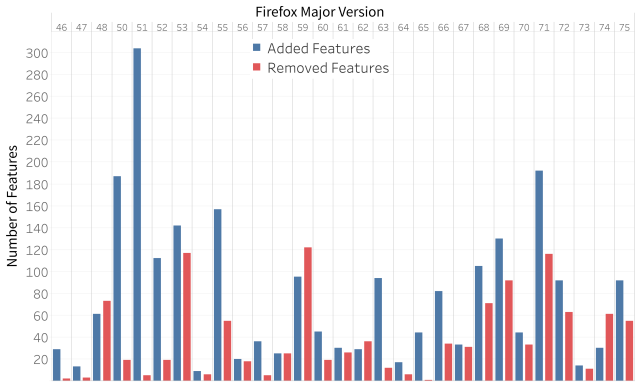

Figure 3: Feature introduction and removal in Firefox.

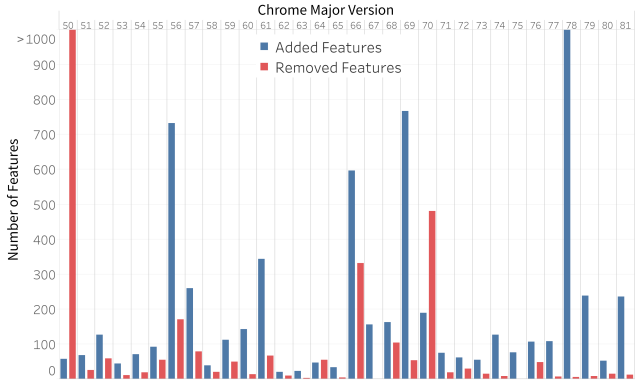

Figure 4: Feature introduction and removal in Chrome.

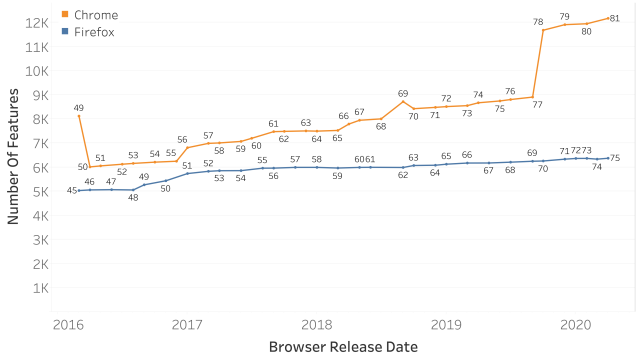

Figure 5: Feature trends in Firefox and Chrome when compared to each other.

4.2 Browser Fingerprintability

4.2.1 Analyzing fingerprinting API presence in Chrome and Firefox. Recall that one of the key research questions we asked at the beginning of this paper was if popular browsers such as Firefox and Chrome 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 earliest Chrome version in our analysis), there existed 139 APIs that could be used for fingerprinting. Chrome 81 (the oldest Chrome version in our analysis), in contrast, there existed 274 APIs that could be used for fingerprinting. In short, the number of APIs that could be used for fingerprinting Chrome versions were increasing over time. That is, the data suggest that Chrome is becoming easier to fingerprint.

Compared to Chrome, Firefox 45 (i.e., the oldest version in our study) has 147 APIs that can be used for fingerprinting it. In constrast, Firefox 75 (which is the latest Firefox version in our study) has 271 fingerprinting APIs. Interestingly, though, Firefox 71 has 276 APIs that can be used for fingerprinting. 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., [20]).

Figure 6 depicts, in detail, the presence of fingerprinting APIs in Chrome and Firefox 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 [14] and Chrome 56 [11].

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 [2]. Similarly, in Firefox 51, we observed that the browser had also added WebGL2 support during that time.

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 and Chrome 56. The straightforward 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.

4.2.2 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 and Firefox 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 both Firefox and Chrome are converging towards homogeneity of browser features.

4.3 Firefox and Chrome Vulnerability Analysis

After looking at feature trends and fingerprintability of major browsers, one interesting question to attempt to answer is this: If browser fingerprinting is becoming easier over time because Firefox and Chrome are, in general, becoming more "bloated" with features, does this mean that these browsers are also becoming less secure over time? After all, the common folk wisdom is that if a software is more bloated, it offers more vulnerabilities and a larger attack surface to launch attacks against (e.g., [18]). In this section, we attempt to answer if there is a direct link between the number of features a specific browser offers, and the number of vulnerabilities that have been reported for that browser version.

To answer this question, we extracted the CVE reports for Chrome and Firefox from 2016 to 2020 as discussed in the methodology section. We parsed these reports automatically, and generated CVEs that are related to Google Chrome and Mozilla Firefox for the versions we were interested in. For Chrome, most CVEs were affecting Google Chrome 49.0.2623 which had 919 reported CVEs. In contrast, Chrome 81.0.4044.129 had the least number of CVEs – hence, making it the most secure Google Chrome version. In contrast, for Firefox, 589 CVEs were reported for Firefox 45 which made it the most vulnerable Firefox browser in our study. Compared to Firefox 45, Firefox 75 was the most secure Firefox version in our study with only 11 CVEs. Detailed information about the number of CVEs reported per browser version is available in Figures 8 and 7.

If we compare CVE date per browser version with the feature trends we reported in Figure 5, it is evident that the increase in the number of reported vulnerabilities does not correlate with the increase in the number of supported browser features. At first, this might seem counter-intuitive.

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

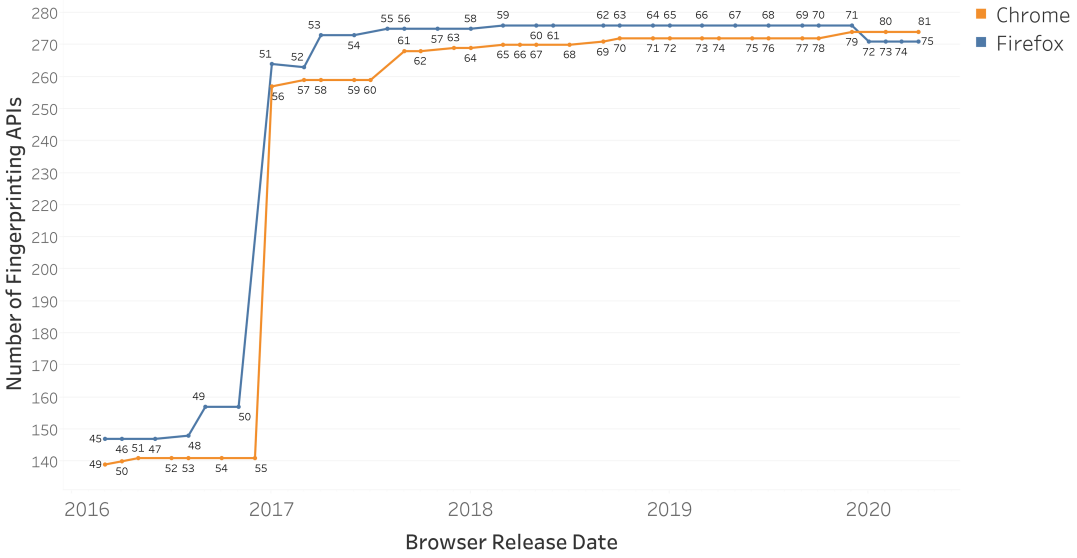

Figure 6: Presence of Fingerprinting APIs in Chrome and Firefox.

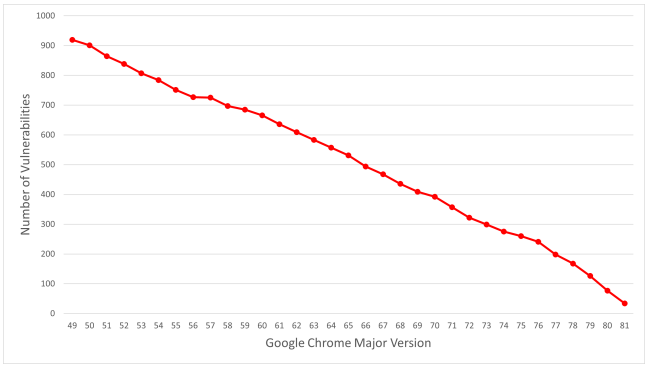

Figure 7: Google Chrome vulnerabilities per version.

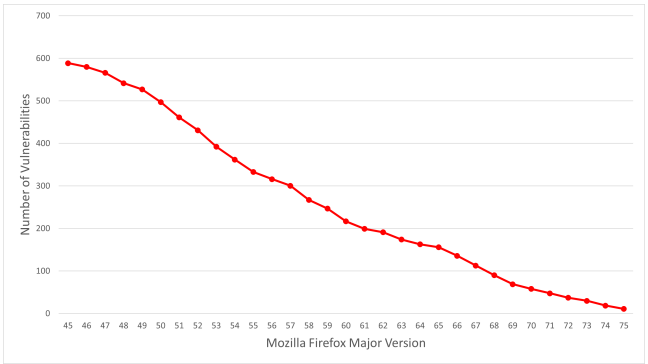

Figure 8: Mozilla Firefox vulnerabilities per version.

However, it is clear that browser vendors are investing heavily into making their browser codebase more robust and secure (e.g., [12, 21]). Hence, while the general intuition that code "bloating" leads to more security problems may be correct, at least in the case of browsers, this assumption does not seem to hold true.

5 RELATED WORK

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

Browser evolution The first web browser, WorldWideWeb [7], 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. [24] 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 [26], 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. [10] 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 [13]. 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. [9] 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. [23] 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. [22] showed how tracking has moved from using cookies (stateful) to browser fingerprinting (stateless) on the web. Mowery et al. [19] 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. [27] 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. [28] 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

Web browsers have become important tools in our daily lives. Millions of users use web browsers for a wide range of purposes such as social media, online shopping, or surfing the web. Some of these services use browser fingerprinting to track and profile their users which can be in contrast with their web privacy. At the same time, browsers are also being targeted by attackers because they are attractive platforms to compromise. In this paper, we analyzed the impact of browser features on browser fingerprinting and security. We investigated more than 30 major browser versions for both Google Chrome and Mozilla Firefox 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 have been increasing over the years.

We also analyzed if the increase in the numbers of supported features and the fingerprintability lead to more reported security vulnerabilities. We extracted the CVEs for every browser version, and compared the number of CVEs to the number of features supported in the browser. Our data suggest that there is no direct relationship between the number of vulnerabilities in a browser, and the number of total features that it supports. Unlike the common belief, the "bloating" of browsers does not seem to be leading to more insecurity.

REFERENCES

[1] 2020. AmIUnique. https://amiunique.org [Online; accessed 16. Oct. 2020].

[2] 2020. EasyList. https://easylist.to/ [Online; accessed 19. Oct. 2020].

[3] 2020. Panopticlick. https://panopticlick.eff.org [Online; accessed 16. Oct. 2020].

[4] 2020. WebIDL Level 1. https://www.w3.org/TR/WebIDL-1/ [Online; accessed 16. Oct. 2020].

[5] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective: Dusting the Web for Fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 1129–1140. https://doi.org/10.1145/2508859.2516674

[6] Apple. 2019. Safari Privacy Overview. https://www.apple.com/safari/docs/Safari_White_Paper_Nov_2019.pdf.

[7] Tim Berners-Lee. 1990. The WorldWideWeb browser. https://www.w3.org/People/Berners-Lee/WorldWideWeb.html.

[8] BrowserStack. 2020. App & Browser Testing Made Easy. https://www.browserstack.com/.

[9] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. https://doi.org/10.14722/ndss.2017.23152

[10] Qian Chenxiong, Hyungjoon Koo, Changseok Oh, Taesoo Kim, and Wenke Lee. 2020. Slimium: Debloating the Chromium Browser with Feature Subsetting. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.

[11] Google Chrome. 2017. New In Chrome 56 | Web. https://developers.google.com/web/updates/2017/01/nic56. [Online; accessed 16. Oct. 2020].

[12] Cloudwards. 2020. Most Secure Web Browser of 2020: Staying Safe Online. https://www.cloudwards.net/most-secure-web-browser/.

[13] Peter Eckersley. 2010. How Unique Is Your Web Browser?. In *Privacy Enhancing Technologies*, Mikhail J. Atallah and Nicholas J. Hopper (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[14] Mozilla Firefox. 2017. Firefox 51.0, See All New Features, Updates and Fixes. https://www.mozilla.org/en-US/firefox/51.0/releasenotes/. [Online; accessed 16. Oct. 2020].

[15] Mozilla Firefox. 2020. How to block fingerprinting with Firefox. https://blog.mozilla.org/firefox/how-to-block-fingerprinting-with-firefox/.

[16] 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* (Lyon, France) *(WWW '18)*.

International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 309–318. https://doi.org/10.1145/3178876.3186097

[17] Jordan Jueckstock and Alexandros Kapravelos. 2019. VisibleV8: In-browser Monitoring of JavaScript in the Wild. /projects/vv8/. In *Proceedings of the ACM Internet Measurement Conference (IMC)*.

[18] Michael Kassner. 2016. Bloatware as a security risk: Researchers' innovative ways to combat the scourge. https://www.techrepublic.com/article/bloatware-as-a-security-risk-researchers-innovative-ways-to-combat-the-scourge/.

[19] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP* (2012).

[20] Mozilla. 2020. How to block fingerprinting with Firefox . https://blog.mozilla.org/firefox/how-to-block-fingerprinting-with-firefox/.

[21] Mozilla. 2020. Progress in Our Quest to Defeat Browser Crashes. https://blog.mozilla.org/firefox/defeat-browser-crashes/.

[22] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Chris Kruegel, Frank Piessens, and Giovanni Vigna. 2013. Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[23] Łukasz Olejnik, Claude Castelluccia, and Artur Janc. 2012. Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns. (07 2012).

[24] Peter Snyder, Lara Ansari, Cynthia Taylor, and Chris Kanich. 2016. Browser Feature Usage on the Modern Web. In *Proceedings of the Internet Measurement Conference (IMC)*.

[25] Peter Snyder and Ben Livshits. 2019. Brave, Fingerprinting, and Privacy Budgets. https://brave.com/brave-fingerprinting-and-privacy-budgets/.

[26] Peter Snyder, Cynthia Taylor, and Chris Kanich. 2017. 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*.

[27] Oleksii Starov, Pierre Laperdrix, Alexandros Kapravelos, and Nick Nikiforakis. 2019. Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In *Proceedings of the World Wide Web Conference (WWW)*.

[28] Erik Trickel, Oleksii Starov, Alexandros Kapravelos, Nick Nikiforakis, and Adam Doupe. 2019. Everyone is Different: Client-side Diversification for Defending Against Extension Fingerprinting. In *Proceedings of the USENIX Security Symposium*.