Appendix F: browser engines

Introduction

1. As discussed in Chapter 5, the browser engine is a key part of the browser, and its role is to transform web page source code into web pages (or web apps) that people can see and engage with.

2. In this appendix, we provide additional details on key topics that are discussed in Chapter 5, namely:

   - web compatibility (which is primarily determined by the browser engine);
   - evidence comparing browser engines;
   - browser engine security;
   - in-app browsing; and
   - the history of browser engines.

Web compatibility

3. The type of content which can be created by online content providers and accessed by users depends on the capabilities which have been implemented by browser engines. If the browser engine in a user's browser does not have a particular capability, then a user will be unable to properly engage with the relevant web content. For example, if a user's browser engine lacks the ability to process a particular video format, the user will not be able to watch a video using that format which has been uploaded to a web page.

4. Online content providers try to ensure that their content is compatible with multiple browser engines so that it reaches as many consumers as possible. However, where browser engines' capabilities differ, online content providers may choose to produce content which is not supported by all browser engines. This is a particularly important issue for online content providers given that consumers appear to not regularly multi-home on mobile devices.[1]

5. There is an inherent tension between compatibility and competition by browser engines to provide new capabilities, as any novel capability provided

[1] Multihoming means that users use more than one browser on their mobile device. See for example, in the Australian context, the ACCC Consumer Survey - Roy Morgan Report, - 17 September 2021.pdf section 3.3.2.1

by a browser engine will not be present in other browser engines. A small browser engine which adds a novel capability cannot expect this to be used extensively by online content providers, given the small audience it commands. A capability implemented by a widely used browser engine, on the other hand, is more likely to be adopted. Once a critical mass of online content providers have implemented content using a novel capability, other browser engines may feel pressure to replicate it as their users increasingly encounter web pages which use the new capability (and as a result of the standards-setting work described below).

6. Compatibility is not 'all or nothing', and web developers have adopted a range of strategies to manage compatibility issues. A popular approach is 'progressive enhancement', where web content is structured to be available to as wide a variety of browser engines as possible, but enhanced by newer functionality where a user's browser permits.[2] If a capability is sufficiently important, online content providers can resort to telling users to use a specific browser to access their content.

7. At times online, content providers have found it difficult to ensure that their content is compatible with multiple browser engines (such as the browser engines in the early versions of Netscape and Internet Explorer). To avoid this, and to maximise compatibility, institutions have been created through which they can coordinate around novel browser engine capabilities through web standards.

8. Key standards development organisations include:

   - **The Internet Engineering Task Force (IETF)**:[3] a loosely self-organised group that contributes to the engineering and evolution of Internet technologies, via producing high quality, relevant technical and engineering documents (such as protocol standards and best current practices documents) that influence the way people design, use, and manage the Internet. It aims to support the evolution of the internet and maintaining the smooth running of the internet as a whole, via developing and maintaining the Request For Comment (RFC) documents that define the open standards by which the internet is managed. These open standards are developed via rough consensus.

   - **The World Wide Web Consortium (W3C)**:[4] which develops web standards via its international community of member organisations, a full-

[2] Hesketh (2003) Progressive Enhancement and the Future of Web Design. Accessed 04/05/2021.
[3] IETF website. Accessed 04/05/2021.
[4] W3C website. Accessed 04/05/2021.

time staff, and the public. W3C's primary activity is to develop protocols and guidelines that aim to ensure long-term growth for the web. The W3C adopts a process to get to a 'W3C Recommendation' or 'standard',[5] via workshop, activity proposal and working group, by which specifications and guidelines are reviewed and revised.

9. As a result of work by standards development organisations, there are a series of open standards that should address concerns of web compatibility. However, in practice, compatibility and standards-setting issues remain. In particular:

- Mozilla submitted that dominant players, and in particular Google, can distort markets due to not committing to final standard and/or not respecting the agreed timelines to deploy or deprecate relevant technologies. By way of example, Mozilla told us that in video conferencing services 'this is the implementation of WebRTC in browsers, where premature deployment of a non-standard interface in Chromium resulted in over half a decade of compatibility problems between websites and other browsers'.[6]

- Apple submitted that with respect to G Suite, which is a suite of web applications created by Google for businesses (now called Google Workspace), users in browsers based on WebKit are unable to access documents offline or voice typing. Apple submitted that it understands that rather than using web standard functionality, G Suite depends on a browser extension that ships with Chrome and is not available on other browsers.

- One browser vendor submitted that website developers do not always develop sites against the standards and instead develop sites using development tools (often provided by browser makers) and test those sites against the web browsers they want to make sure they support. Similarly, an internal document from Microsoft states that 'web developers are explicitly coding first and foremost to Chrome' and that this means that 'in most cases developers are not even consulting web standards or testing on other browsers.

[5] W3C website. Accessed 04/05/2021.
[6] Mozilla's Response to the Public Consultation on Google's Privacy Sandbox Commitments at Page 7, dated 8 July 2021.

- One market participant submitted that the strength of Chrome has given Google 'control' over standards bodies, allowing it to push its preferred specifications which must then be implemented by its competitors.

- Although online content providers largely told us that they ensure web compatibility with all major browsers, there are certain exceptions. Out of the 33 online content providers that commented on web compatibility, seven did not list the Firefox mobile browser as a browser for which they ensure compatibility.[7] Additionally, it is unclear which level of support the online content providers considered when answering this question.

10. A Open Web Advocacy told us that they estimated that supporting all browsers costs around 130-160% of the initial implementation cost of a webpage or web app if standards have been followed correctly, compared to a 300-500% cost (in addition to substantial ongoing maintenance) to support multiple platforms for web, Android, iOS and desktop.

11. The network effects caused by web compatibility create weak incentives that mean the development of new features is not a main way in which browsers and browser engines differentiate themselves. Instead, performance, security, privacy, and UI are the main parameters along which browsers and engines are differentiated.

Comparing browser engines

12. We engaged with various stakeholders on test suites that compare WebKit to other browser engines. Several stakeholders proposed measures that assess compatibility and feature support. For example, in response to our request for information, such measures were proposed by Google and several technical experts.[8] Mozilla submitted that while measures that assess feature support may be useful to assess interoperability issues, it listed other factors such as performance and standards compliance as also relevant. Apple submitted that a focus on compatibility or web standards compliance does not adequately reflect attributes of browser engine quality such as performance, stability, and privacy. Apple explained that, in the normal course of business, it tests web app responsiveness, JavaScript performance and graphics performance.

13. Based on the above submissions, we consider that while a variety of measures are likely to be relevant, security, compatibility, and feature support

[7] Of these seven, three did not list Firefox at all, while the remaining four only listed the Firefox desktop browser. One respondent submitted that the latest version of Firefox was partially supported.
[8] Submissions included a submission from Alex Russell (Progress Delayed Is Progress Denied - Infrequently Noted).

appear to be particularly important. We have therefore focused primarily on these measures.

Compatibility and feature support

14. In the below, we present the test suites that were recommended to us and further discuss stakeholders' views on them.

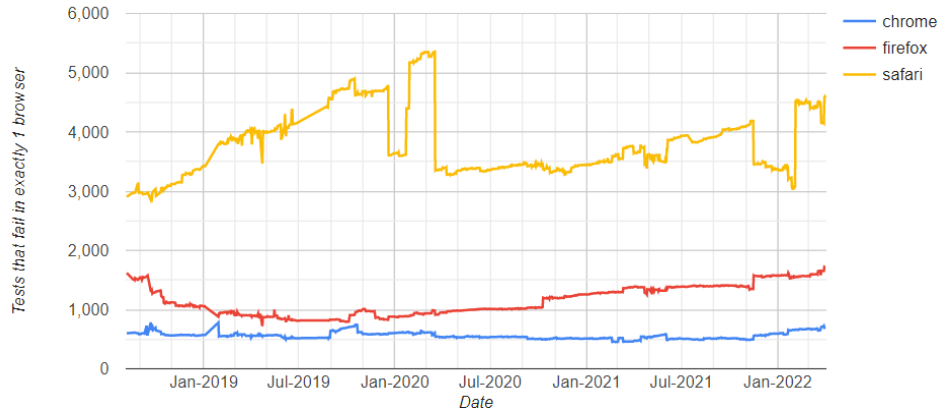
Recommended test suites

15. In response to our request for information, Google as well as several technical experts endorsed a test suite measuring compatibility and feature support called the Web Platform Test (WPT) Dashboard, also referred to as wpt.fyi.[9,10] This project runs tests for various browser technologies and, on the basis of the test results, provides assessments of compatibility and feature support of different browsers.

16. Figure F.1 below shows one of these assessments. For each browser, the line measures the number of tests that this browser fails but that all other browsers that were considered pass.[11] This can broadly be interpreted as instances where the browser is not compatible while the other browsers are. The yellow Safari line (which represents any browser built on WebKit) is substantially and persistently higher than the blue Chrome and red Firefox lines (representing browsers built on Blink and Gecko respectively). This indicates that WebKit has performed significantly worse in terms of compatibility than Blink and Gecko over this period.

[9] See web-platform-tests dashboard (wpt.fyi)
[10] Endorsements included an endorsement from Alex Russell (Progress Delayed Is Progress Denied - Infrequently Noted). Mozilla submitted that the Web Platform Test Project is useful to gauge interoperability issues, but that it looks at just one facet of how browser engines operate and implement certain web standards.
[11] For example, a number of 4,000 for Safari means that there are 4,000 tests which Safari fails but that all other browsers that were considered pass.

Figure F.1: Number of tests which fail in exactly one browser

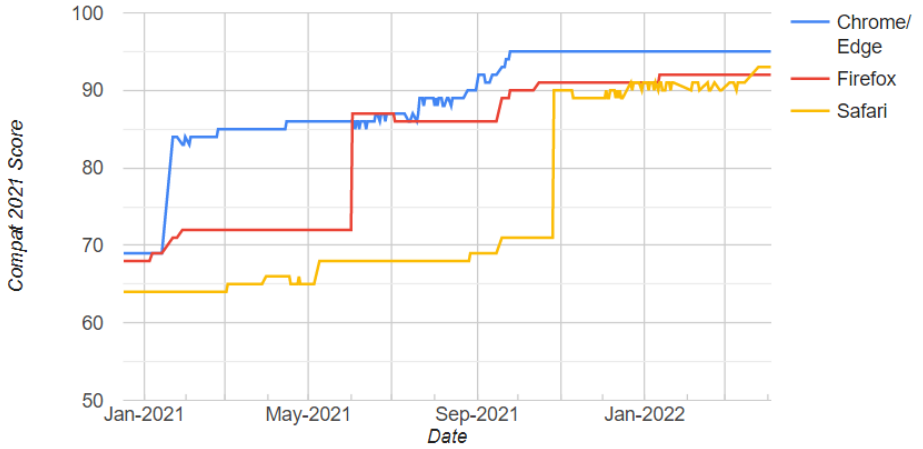


Source: web-platform-tests dashboard (wpt.fyi)
Note: Graph shows test results based on stable (rather than experimental) version. Graph retrieved 05 April 2022.

17. Another assessment provided by wpt.fyi is presented in Figure F.2. This assessment focuses on the 2021 Compat Focus Areas, which are five key areas that represent the most painful compatibility bugs (ie a small subset of the features considered in Figure F.1 above). The scores represent how well browser engines are doing on the 2021 Compat Focus Areas (a higher score being better).

18. The yellow Safari line (which represents any browsers built on WebKit) shows that Safari had a significantly worse compatibility score for most of 2021 than browsers built on Blink or Gecko. We note that the score for Safari has recently improved significantly, following from Apple releasing Safari 15 as part of its iOS 15 release (the compatibility score prior to the jump was based on versions of Safari 14).

Figure F.2: Compat 2021 score

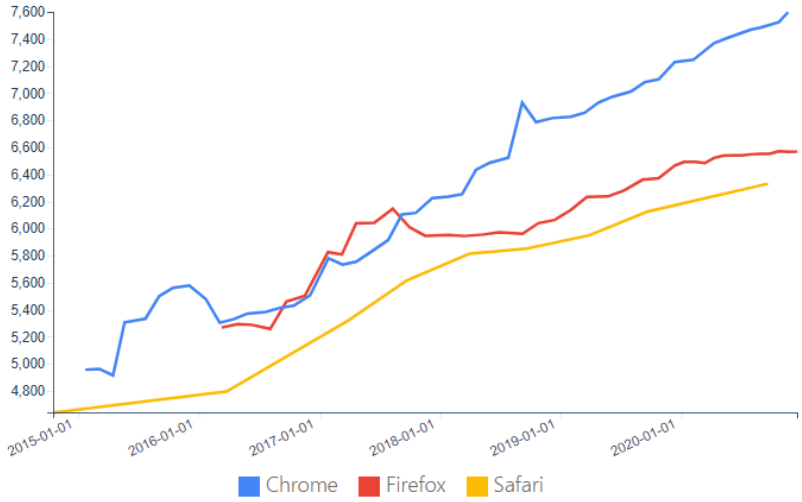


Source: web-platform-tests dashboard (wpt.fyi)
Note: Graph shows test results based on stable (rather than experimental) version. Graph retrieved 05 April 2022.

19. There are several other tests measuring compatibility and feature support to which certain stakeholders referred us, which are set out below.

20. In response to our request for information, Google referred us to a count of APIs available on JavaScript, which is based on data from the Web API Confluence Dashboard and was aggregated by Alex Russell.[12] It shows that the count of APIs available from JavaScript on Safari is lower than on Chrome and Firefox.[13]

Figure F.3: Count of APIs available from JavaScript

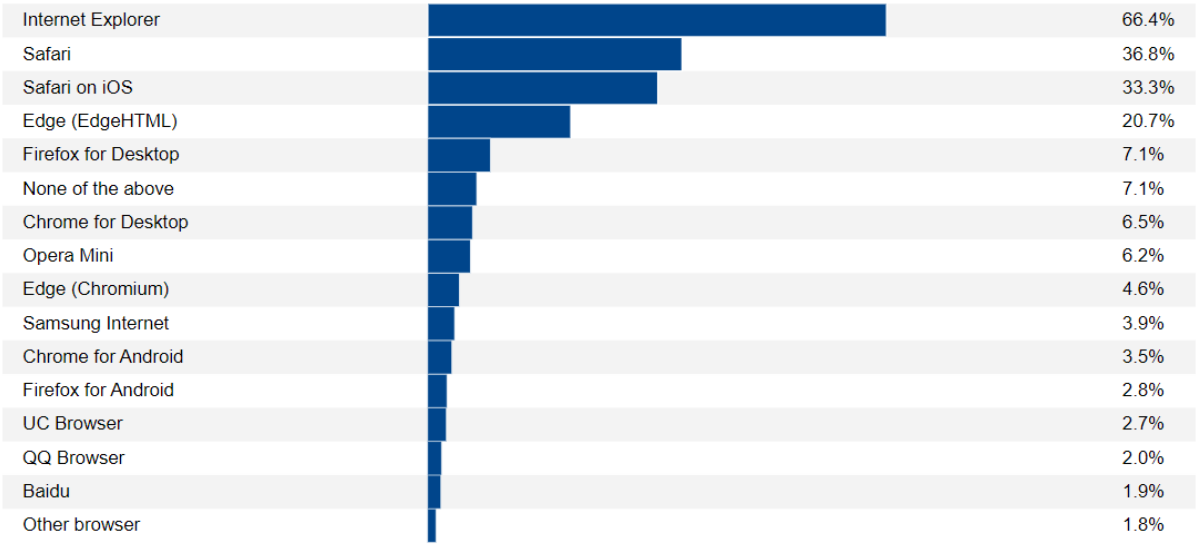


Source: Progress Delayed Is Progress Denied - Infrequently Noted

21. Open Web Advocacy recommended the Mozilla Developer Network (MDN) Web Developer Needs Assessment. Figure F.4 below shows developers' responses to being asked to rank which browsers cause the most issues (they were allowed to select up to three). With the exception of Internet Explorer (which Microsoft is no longer developing), Safari ranks worse than all other browsers.

[12] For disaggregated data, see Web API Confluence Dashboard (web-confluence.appspot.com).
[13] We understand that, again, Safari should be interpreted as any browser built on WebKit while Chrome and Firefox should be interpreted as any browser build on Blink or Gecko, respectively.

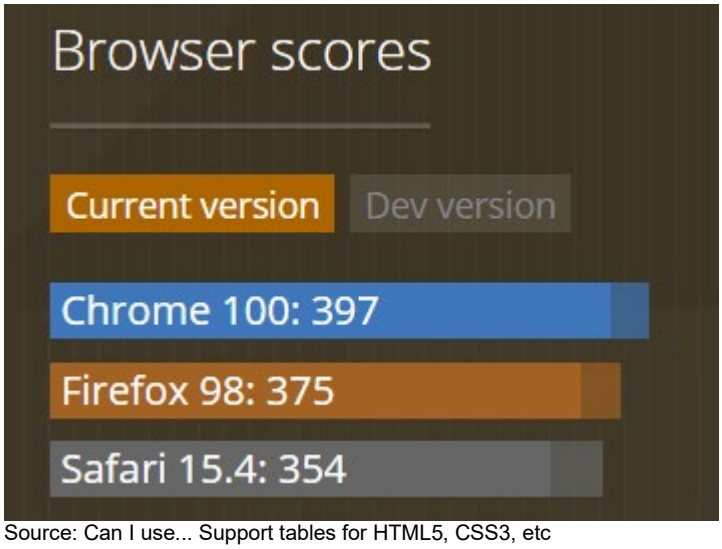
Figure F.4: MDN Web Developer Needs Assessment



Source: MDN Web Developer Needs Assessment 2020 - MDN (mozilla.org)

22. Mozilla referred us to a website called 'Can I Use', which lists which browsers support which features. Open Web Advocacy told us that while this data does not to capture the detail that wpt.fyi includes, it is typically the 'first stop' for web developers in assessing the ability to attempt to use a feature. Figure F.5 shows a summary of the browser scores based on all features tracked on 'caniuse', indicating that Safari supports fewer features than Chrome and Firefox. These scores represent tallies of features that canisue track.

Figure F.5: Can I use browser scores

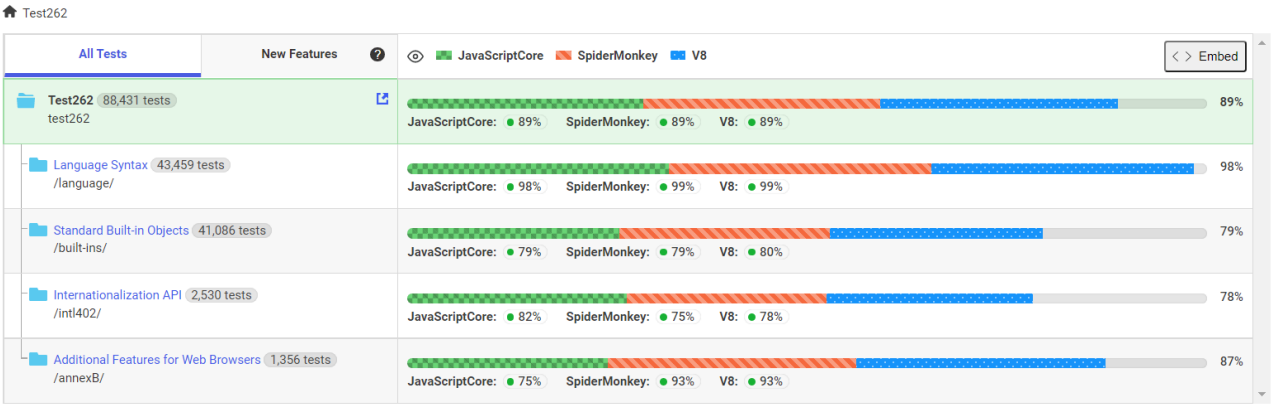


Source: Can I use... Support tables for HTML5, CSS3, etc
Note: Graph shows results for current version. Graph retrieved 05 April 2022.

23. Mozilla submitted that it recommends Test262, which tests standards compliance of different Java Script engines as an example of a way to test browser engines that goes beyond the Web Platform project (Java Script engines, as discussed in Chapter 5, form part of the browser engine). Open Web Advocacy told us that this test suite is more limited in use for web

developers than wpt.fyi or Can I Use because it focuses on only one of the web's four major languages. Figure F.6 below shows that Safari's Java Script engine (called JavaScriptCore) performs similar to those of Blink (V8) and Gecko (SpiderMonkey) overall, but significantly worse with respect to additional features for web browsers.

Figure F.6: Test 262



Source: Test262
Note: Graph retrieved 05 April 2022

Stakeholders' views on the test suites

24. With respect to tests in general, Mozilla submitted that it is generally hard to judge the real-world performance of different browser engines from generalized tests, and that the comparisons such tests lead to are 'often subjective, blunt and lack nuance that is meaningful to end users' both when it comes to performance but also web compatibility and interoperability.

25. With respect to the test suites discussed above, and in particular the wpt.fyi assessment on the number of tests which fail in exactly one browser (Figure F.1), Apple submitted the following main criticisms:[14]

- First, Apple submitted that several of the tests discussed above focus on metrics based on the total number of tests run, rather than the importance of those tests. Apple submitted that it therefore does not believe that these metrics are reflective of a browser engine's quality or impact the vast majority of web developers.

- Second, Apple submitted that many tests run by wpt.fyi turn out to be mistaken, or to support single-browser non-standard features, which Apple does not believe is helpful. Apple also noted that, since tests are

[14] Apple also submitted that wpt.fyi is a third-party group of test suites that are contributed to by various people, including browser vendors.

contributed by third parties, the test quality is inconsistent, and vulnerable to gaming by browser vendors.

- Third, Apple submitted that wpt.fyi is configured in a way that distorts the actual performance of Safari. In this regard, Apple submitted that (i) wpt.fyi compares an old beta version of Safari with newer versions of other browsers,[15] (ii) wpt.fyi runs tests in private browsing mode (which leads to many of the compatibility failures that are being reported) and (iii) a substantial portion of the reported test failures can be attributed to Safari's lack of support for the web specification 'SharedWorkers', for which Apple removed support due to its low adoption.

- Finally, Apple suggested the use of Interop2022[16] as a test suite. On this measure Safari outperforms Chrome and Firefox. This measure is not exactly a comparable measure to the wpt.fyi tests. The measure groups tests into categories to calculate the percentage of tests that pass in each. The score is then computed as the average across categories. Therefore, one clear limitation of this metric is that it weights categories equally and not by importance or number of tests.

26. With respect to Apple's first point, Google told us that it considers measures on overall API support (such as the graph on the number of tests which fail in exactly one browser in presented in Figure F.1 and the graph on the count of APIs available from JavaScript presented in Figure F.3) are simple and objective measures. Google also submitted that it considers these tests to be quite comprehensive and cover a wide range of APIs.

27. We also note that one of the assessments (Figure F.2) focuses on the five key areas that represent the most painful compatibility bugs. Even on the basis of this narrower set of features, Safari was performing significantly worse than Blink and Gecko based browsers until very recently.

28. With respect to Apple's second and third point, Google told us that the tests are developed with and shared with the community. Google also told us that it considers the comparison to be fair and conducted on equivalent terms. In particular, it told us that it does not think that the test results are reliant on enabling a special mode and that it would expect that the graphs would, independent of which mode is selected, show a similar pattern.

---

[15] In particular, Apple submitted that wpt.fyi uses Safari Technical Preview ("STP"), a months-old beta version of potential future iterations of Safari for a technical audience.
[16] See: interop2022

29. We also note that, as set out above, the wpt.fyi assessments were endorsed by several stakeholders, which calls into question the quality concerns raised by Apple.

30. Overall, we acknowledge that there are certain limitations to the test suites discussed above. With respect to the wpt.fyi assessment on the number of tests which fail in exactly one browser, we consider that it is a meaningful indicator of the relative feature support of WebKit compared to other browser engines. We also note that a number of other test suites show similar patterns with respect to WebKit's feature support.

Quantitative comparisons of security between Safari on iOS and Chrome on Android

31. We engaged with various stakeholders on the best ways to compare security. While there are limitations to any quantitative security comparison, we believe that measuring the time to patch security vulnerabilities and the regularity of updates is a useful measure.

32. Table F.1 shows analysis from Google's Project Zero team of vulnerabilities found by the team between January 2019 and December 2021. The table shows that while there were more vulnerabilities spotted by the team on Chrome, it is WebKit vulnerabilities that were slowest to be patched following spotted vulnerabilities. This analysis is limited as it only included vulnerabilities reported by the Google Project Zero team,[17] however it is consistent with concerns we have heard.

Table F.1: ProjectZero Metrics 2019-2021

Fix time analysis for open-source browsers, by bug volume

Browser	Bugs	Avg days from bug report to public patch	Avg days from public patch to release	Avg days from bug report to release
Chrome	40	5.3	24.6	29.9
WebKit	27	11.6	61.1	72.7
Firefox	8	16.6	21.1	37.8
Total	75	8.8	37.3	46.1

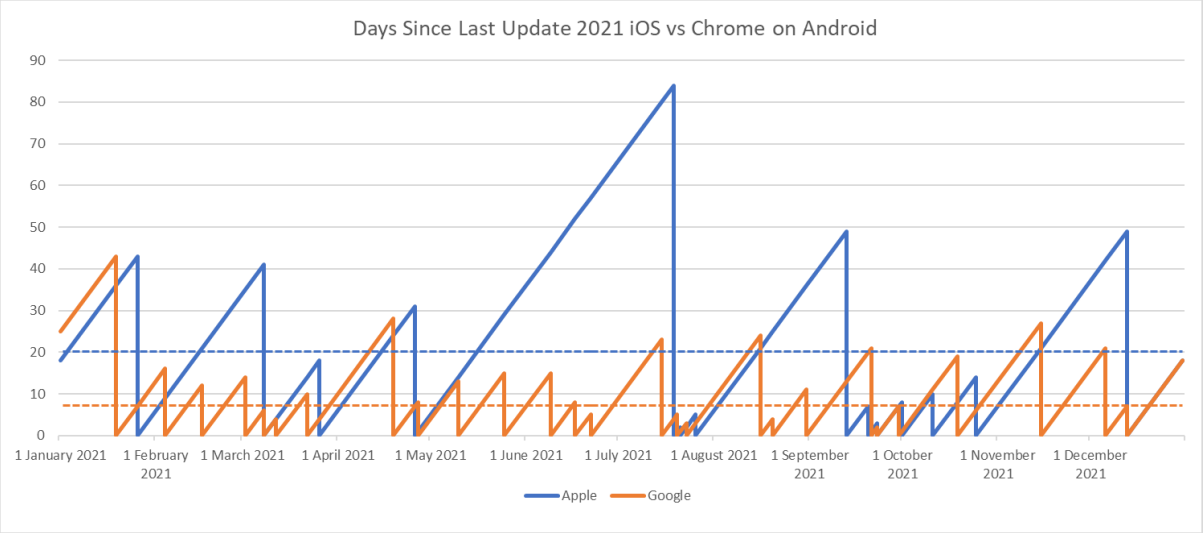
Source: Project Zero: A walk through Project Zero metrics (googleprojectzero.blogspot.com)

33. Additionally, Figure F.7 reports for 2021 the time since last update on iOS vs Chrome on Android. The figure shows Google more regularly updated its

[17] This is also a general issue with any vulnerability list as they only include vulnerabilities that are publicly disclosed. It can also be common for vendors to group several vulnerabilities into one CVE ID (Common Vulnerabilities and Exposures ID) which makes such analysis difficult when using other available vulnerability lists. Apple does not publish an equivalent publicly available vulnerability list.

mobile browser in 2021. On average there was just under 10 days between Chrome updates on Android in 2021 compared to around 20 days for iOS.[18]

Figure F.7: Days since last update (dotted lines indicate average time to update across the period)



Source: Public information on updates

34. There are limitations to quantitative comparisons of security across software or devices. For example:

- There is no way to effectively measure how many vulnerabilities software contains.
- A higher number of vulnerabilities may reflect more active efforts to find and fix security issues.
- Measures of attacks reveal more about attacker preferences than security.
- A higher number of updates may not reflect better security. For example, this could reflect imperfect fixes for old issues or the fact that a system has more features.

35. As a result of these limitations, we do not place significant weight on the above evidence. It is however broadly consistent with concerns submitted to us that Apple's browser engine is not demonstrably more secure than other major browsers engines.

[18] Immediately prior to the publication of our report, Apple announced that it is changing its security practices to allow it to rollout security updates more quickly (Apple, iOS 16 Preview).

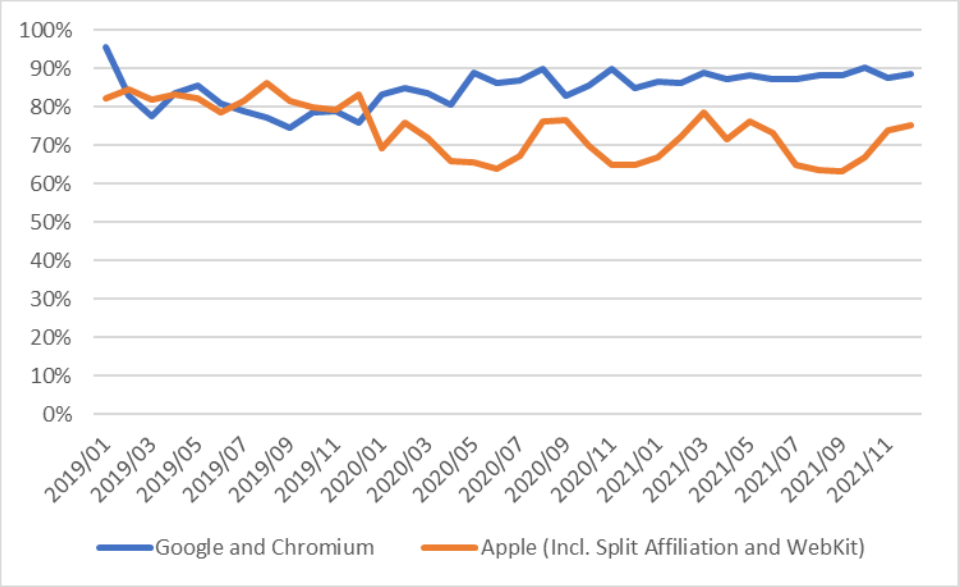
Evidence related to investment in browser engines

36. We are interested in patterns of investment in browser engines as a measure of the competitive effort browser engine stewards exert. We examined contributions to the codebase of the browser engines as an imperfect proxy for the level of development activity of each browser engine.

37. The number of contributions to the engine's codebase and the relative number of these made by the developer serve as a proxy for investment into the engine. Each time a developer submits code to the codebase a 'commit' record is generated. Attribution of these 'commits' is not always clear and this analysis is limited by the fact that not all 'commits' are equal in length or importance. We have however received similar analysis from respondents to the interim report.[19] Therefore, while we acknowledge the limitations to this work, we believe it provides a further source of information to understand the relative investment of developers into their browser engines.

38. In figures F.8 and F.9 we present 'commit' data scraped from the open-source codebases of Blink and WebKit. We have also tried to compile similar data for Gecko however due to the governance of Gecko and difficulties in attributing commits we have not been able to provide comparable figures.

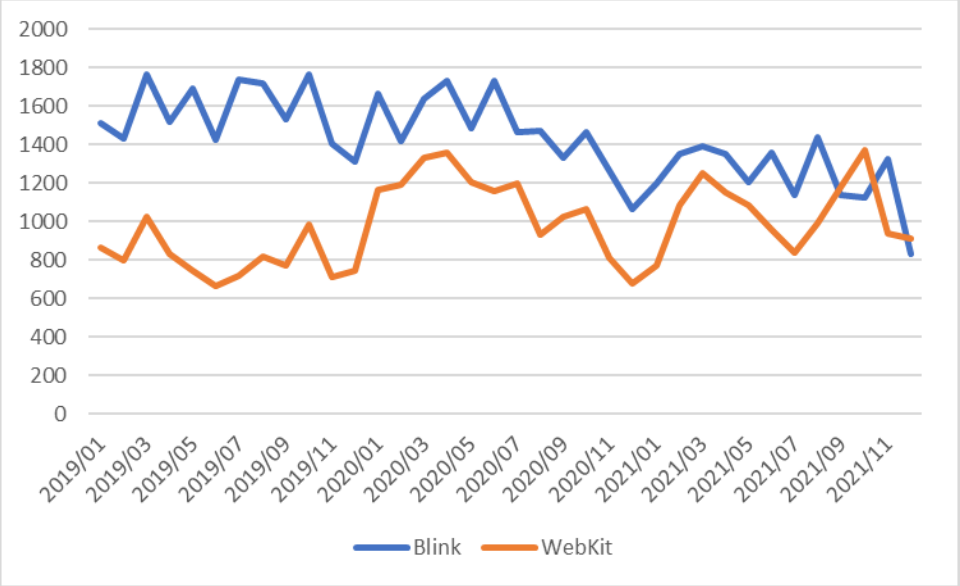
Figure F.8: First Party Commits relative to Total Commits



Source: CMA analysis of WebKit and Blink commit data

[19] Open Web Advocacy (2022) Bringing Competition to Walled Gardens. Page 72.

Figure F.9: Total Number of Commits



Source: CMA analysis of WebKit and Blink commit data

39. Figures F.8 and F.9 show that for the period Jan 2019- Dec 2021:

- The number of commits and the share of these by the main developer are relatively stable.
- There are around 45% more commits to Blink than WebKit.
- There was a large difference in the total number of commits between Blink and WebKit however in recent months this difference has declined.
- Close to 84% of Blink commits were by Google.[20]
- Around 73% of Webkit commits were by Apple.[21]

40. Overall, given the limitations of this exercise we do not place considerable weight on this evidence. However, the analysis provides evidence consistent with concerns we have heard from stakeholders that Apple underinvests in WebKit.

Other measures

41. Apple submitted that it uses test providers to test web app responsiveness, JavaScript performance and graphics performance. Apple further submitted

[20] This includes commits by those with chromium.org emails.
[21] This includes commits by those with split affiliations and identified as WebKit but does not include any commits made by other organisations on behalf of Apple.

that, on these tests, WebKit-based Safari on iOS outperforms competing browsers based on competing engines on Android devices.

42. Apple additionally submitted that it believes that power efficiency (particularly in mobile applications), page load speed and memory use are good metrics to compare browser engine performance. While Apple submitted that there are no third-party test suites that measure these metrics across browsers, it submitted internal tests showing that WebKit-based browsers perform better on page loading and power efficiency than competing browsers based on other browser engines.

Browser engine security

43. As browsers connect users to websites running unknown code, they are a target of cyber-attacks by malicious actors. The vast majority of mobile consumers are rarely targeted by browser-based exploits. Instead, most malware enters through malicious (or otherwise predatory) apps downloaded on app stores.

44. For users using the most up-to-date browser, successful attacks are extremely rare. Browsers are complex and protected by advanced layers of defences often in the form of sandboxes and hardware-level security mitigations. This means attacks are costly to develop and are unlikely to be used on average consumers.

45. Attacks occurring through the browser are more likely to target users on outdated browsers. For these users, attackers can utilise older exploits yet to be patched. As a result, a key line of defence against cyber-attacks is regularly updated software (see figure F.7 for an analysis of Chrome and Safari updates).

46. Apple has submitted that several of the policies which we have identified as anti-competitive are necessary to protect users' security.[22] For example, Apple has used security arguments to justify:

    - the WebKit restriction; and
    - reserving features or APIs for Safari.

47. We consider Apple's arguments below. To understand the merits of these arguments we have spoken to a wide range of stakeholders. In addition to engaging with browser vendors and web developers, we consulted experts

[22] Apple, response to Interim Report, 7 February 2022.

including RET2 (a computer security firm from which we commissioned advice).

Box F.1: Key Security Findings

We heard from RET2 that all three existing main browser engines are very secure for the average user:

- Browser engines are rarely the source of consumer harm on mobile devices. Exploits of browser engines are rare and the most common way an average user is likely to fall victim to malware is from installing a malicious app.
- Browser engines are 'hard' targets which are difficult to exploit directly. Exploits targeting up to date browsers are valued in the millions of dollars which makes them too costly for use by average criminals.
- Consumers are most vulnerable to browser engine security exploits if using an outdated browser engine.

Apple's dedicated Safari sandbox and hardware integration are not the only ways to secure a browser engine on Apple devices:

- A common way browser engines protect against attacks is by adding layers of defense in the form of sandboxes.
- Even without hardware-level integration a browser engine such as Blink can provide comparable levels of security to engines with hardware-level integrations such as WebKit on iOS.

The WebKit restriction and security

48. Apple submitted that one of the main justifications for the WebKit restriction is security.[23] We recognise that browser engines are a potential source of security vulnerabilities. However, we have heard from experts that in practice the main browser engines (Blink, WebKit, Gecko) are very secure. For the average user who regularly updates their device the security risks they face through the browser are low.

49. Apple has made arguments that WebKit's tight integration with hardware and the operating system is key in allowing it to deliver substantial security protections. We have heard from experts that hardware integrations are just one way to strengthen security and even absent hardware integration it is likely that a browser engine such as Blink can provide comparable protections to engines with hardware integrations such as WebKit on iOS.[24] Furthermore, other browser engines may be able to implement some of Apple's integrated

[23] Apple, response to Interim Report, 7 February 2022.
[24] For example, due to the utilisation of Site Isolation by Blink.

security measures on iOS, if given the opportunity which would further enable them to improve security.

50. As outlined above the risk from direct exploitation on an up-to-date browser engine is low. However, if users are on an outdated version of the browser engine,[25] they are vulnerable to attacks that directly exploit unpatched vulnerabilities in the browser engine. As such most browsers actively push updates on consumers, and this is vital for consumers' protection. Apple's WebKit integration is contrary to this in two ways:

- WebKit updates are shipped with operating system update. This makes updates less frequent (see figure F.8) and larger in size which can delay consumers from installing them.[26]

- WebKit is the only browser engine on iOS. Therefore, in the time between a vulnerability being exposed and WebKit being patched consumers are unable to temporarily switch to a secure browser.[27]

51. Regarding the importance of updates, Apple noted that having all native apps use WebKit for in-app browsing is important as allows updates to be shipped to many apps at once.[28] We agree that this is an important concern however do not agree that this justifies not allowing alternative browser engines for a select group of dedicated browser apps.

52. Finally, we also note that Apple does not require a similar browser engine restriction on macOS.[29]

Reserved features and security

53. Apple submitted that it reserves certain features for Safari due to privacy, performance, and security concerns. This includes hardware features that can be leveraged to improve security or performance such as JIT access. It also relates to browser features such as push notifications and web extensions.[30] Apple submitted that:

---

[25] This includes the case where a user is on the most updated version of a browser, but the browser vendor has not updated the underlying browser engine.
[26] This concern was highlighted by several web developers who cited the recent example of a bug with IndexDB on WebKit which allowed malicious sites to harvest user data, and which remained unpatched for almost two months.
[27] Concerns regarding Apple's single point of failure" approach were raised by an antivirus company.
[28] Apple, response to Interim Report, paragraph 88, 7 February 2022.
[29] This point was raised by Open Web Advocacy and a large games developer.
[30] We have heard from Apple that they are actively developing the push notifications feature in WebKit on Apple devices.

- The add web page to home screen feature could allow web apps to be added and run without user consent and knowledge. These may contain new security attacks, and create fraud and privacy threats.

- Web extensions could inject code into web pages the user visits to extract user data and activity, and execute new, potentially unsafe code without the user's knowledge. Additionally, third-party developers would gain access to browsing activity and history, as well as user data, without the knowledge of the user, which creates an increased risk of new security attacks.

- If a third-party browser was able to offer background upload and download, the app could take up all computing resources from the device while running in the background, which could cause devices to become unstable or un-usable.

54. We understand that in general adding more features may increase the possibility of attacks and agree that some features pose more serious risks to consumers than others. However, it is not the case that Apple bans these features entirely from all browsers on iOS (eg, as it does with WebBluetooth), in fact Safari users have access to these features.

55. We have not received compelling arguments that allowing dedicated rival browsers access to such features would worsen security. It is unclear why third-party browsers would provide worse security for these features than Apple's current implementation. For example, on Android where such features are accessible to all browsers, we have not seen evidence that this has led to a material worsening of browser security. Rival browsers may be able to innovate on the security of these features if allowed to implement them.

56. Additionally, Apple submitted that to the extent that it can identify a safe and secure way to implement new features without compromising device security or user privacy, it will consider introducing such features within as short a timeframe as is reasonably possible. We welcome this, however have heard concerns that Apple may unreasonably delay adding the ability for other browsers to implement such features.[31] This is concerning if it leads to Safari being the only browser on iOS with access to a specific feature for a prolonged period.

[31] Open Web Advocacy (2022) Bringing Competition to Walled Gardens.

In-app browsing

57. In this section we describe how in-app browsers work on iOS and Android devices. Many native apps interact with the web and as such it is common while using a native app for users to access links to external webpages or to wish to browse the web. Maximising the time users spend within the native app is important for developers. Developers can provide access to the web while keeping users within the native app through in-app browsers.

58. In-app browsers play an important role in allowing users to access the web. Data submitted to us suggests that the amount of traffic through in-app browsing is a substantial proportion of total browsing.[32]

59. For developers to implement in-app browsing they face a number of choices. The choice of implementation has important implications for competition. For example, often developer implementation means that the in-app browser used is not the same as the user's default browser. This undermines competition at the browser level for consumers as choices are made by developers and not users who may have different incentives. More generally given the growing scale of in-app browsing we may be concerned that the way in-app browsers are implemented may reinforce network effects at the browser level. Some of these concerns were highlighted in responses to the interim report.

60. On both iOS and Android, it is the app developer who chooses which option they will use to implement the in-app browser, not the operating system. However, operating systems can facilitate or limit the scope for choice and competition through the implementation options they provide to app developers.

61. A native app developer has three choices, when providing users with the ability to view a webpage:

    - direct the user out of the app to a dedicated browser (eg, to the user's default browser);
    - open an in-app version of an installed browser; and
    - open an in-app browser which has been customised by the app developer.

62. Most apps either direct the user out of the app or open an in-app version of an installed browser, but some larger developers modify their own browser. This

[32] Data submitted by Google indicated that in-app browsing is significant, although Google told us that its estimate was potentially inaccurate. Additionally, data from a large app developer suggests close to 50% of users use in-app browsing when available.

can allow an app to customise the browsing experience for their users. Modifying a browser can also be done to create a dedicated browser app. DuckDuckGo uses the standard WebView and adds additional features and functionality, in particular privacy features.

63. On iOS, the only in-app version of an existing default browser which can be used is Safari (via SafariViewController). Similarly, all customised browsers which can be opened on iOS are based on WebKit (via WkWebView). Therefore, there is little scope for user choice and all in-app browsers use WebKit, reinforcing its position in the market and the network effects it enjoys. App developers cannot respect a consumer's choice of default browser when providing in-app browsing on iOS; they can only redirect the user out of the app entirely.

64. On iOS we have also heard concerns that Apple may encourage or mandate the use of SafariViewController through their proposed App Bounds Domain (ABD) policy. If mandated this could prevent some app developers from providing a customised in-app browser further reducing the scope for differentiation between in-app browsers on iOS.

65. On Android the app developer can either redirect the user out of the app or they can use a in-app browser. If they wish to customise the browser, they can use an in-app browser such as WebView or GeckoView. In these cases, the choice of in-app browser is disconnected from the choice of dedicated browser app that a user selects as default.

66. On Android if a developer wishes to pick an installed browser to appear then they can use a system such as Custom Tabs. This allows the developer to open links in any installed browser app that supports that system. The developer must choose the browser or can show a list for the user to pick from. There is therefore greater scope for developer and user choice than on iOS.

67. We are concerned that the ways in which in-app browsers are currently implemented on iOS and Android reinforce the positions of WebKit and Blink, contributing to the advantages they have as the most popular browser engines.

History of browser engines

68. Every web browser requires a browser engine for layout and rendering. Below, we set out the dates at which new browser engines were created and discontinued, and when browsers switched to and away from those browser engines.

Early proprietary browser engines

69. Browser engines surfaced when the first two commercial-like browsers started competing with each other for popularity. Netscape Navigator and Internet Explorer (IE) both featured proprietary browser engines designed for desktop computers. The intense competition between Netscape and Microsoft in the development of their browser engines led to rapid but incompatible implementations of new technologies, forcing developers to duplicate their efforts and develop content targeting each browser separately.[33]

70. Netscape Navigator was the most popular web browser prior to IE's rise, peaking in 1995 at 83% market share.[34] However, from 1995 onwards, Microsoft invested heavily in IE, spending over $100 million each year on development, distributing it for free and bundling it with its Windows operating system.[35] By 2002, Microsoft had achieved over 90% share of supply in browsers.[36]

71. Netscape did not label its original browser engine separately from the browsers in which it was embedded. In 2000, it replaced its original browser engine with a new browser engine called Gecko.

72. Microsoft initially licenced the Spyglass engine for use in IE,[37] but debuted its own browser engine Trident with the release of IE 4 in 1997. Since then, Microsoft has used Trident in all versions of IE for Windows. Trident is a proprietary browser engine. However, Microsoft permits the development of so-called Trident shells, which are essentially expansions of IE with added functionality.[38,39]

Modern browser engines

73. Most modern browser engines are open source. This means that their code is available at no cost to be reviewed, copied, modified and used. A key advantage of open-source development is that many contributors can submit proposed changes, including features and bug fixes. However, a 'steward' is ultimately in control of which changes are accepted.

---

[33] Web Design in a Nutshell, Niederst Robbins, 2001, page 4.
[34] See: History of Web Browser Engines 1990-2020.
[35] U.S. V. Microsoft: Court's Findings Of Fact (justice.gov), findings of fact 135 and 136.
[36] TheCounter.com: The Full-Featured Web Counter with Graphic Reports and Detailed Information (archive.org).
[37] See: Memoirs From the Browser Wars (ericsink.com).
[38] See: Hosting and Reuse (Internet Explorer).
[39] Examples of Trident shells include AOL Explorer, which is now discontinued, and MSN Explorer.

74. We discuss below the three main modern browser engines that are under active ongoing development – Gecko, WebKit, and Blink – in the order of when they were established.

Gecko

75. Netscape announced Gecko as an open-source browser engine project in 1998.[40] However, Netscape Navigator versions 6.0 which was released in 2000 using Gecko failed to outcompete Microsoft's IE, and in 2003 AOL (which had acquired Netscape) created the non-profit Mozilla Foundation and made it the steward of Gecko.[41]

76. Gecko benefited from open-source contributions from many organisations, including Google, and by 2009, the Gecko-based Firefox browser had over 25% share of browser usage across all devices.[42] However, Gecko's share of browser usage has declined since, and Gecko-based browsers have never been popular on mobile devices; they had a share of browser usage of less than 1% on mobile devices in 2020.[43]

77. In 2016, Mozilla announced Quantum as an effort to build the next-generation browser engine for Firefox users based on Gecko. This involved merging stable portions of Mozilla's experimental browser engine Servo to Gecko to improve Firefox stability and performance.[44] However, the improvement of Gecko using Servo code did not reverse the decline in Gecko's market share.

WebKit

78. Apple created the WebKit browser engine in 2001 by forking the existing KHTML and KJS libraries from the Unix based K Desktop Environment (KDE), an early open-source project.[45] In 2005, Apple released WebKit as open-source software.[46] Apple told us that while it 'has provided guidance, input, and leadership, WebKit is the result of a massive, industry-wide effort. Since its inception, WebKit has had approximately 2,000 individuals who have contributed code, including employees of organisations such as Google, Adobe, Igalia, Samsung, Intel, and Sony'.

---

[40] Levitt, Jason (April 20, 1998). 'Netscape releases the source'. *InformationWeek* (678). pp. 85–90.
[41] Viseur, Robert. (2013) Identifying Success Factors for the Mozilla Project. IFIP AICT 404.
[42] See shares of supply in Chapter 5.
[43] See shares of supply in Chapter 5.
[44] See for example: Quantum - MozillaWiki.
[45] Don Melton on Twitter.
[46] Molkentin, Daniel (June 7, 2005). "Apple Opens WebKit CVS and Bug Database". *KDE News*. Archived from the original on July 15, 2009.

79. WebKit was initially adopted by other browsers, including Google Chrome at Chrome's release in 2008. Apple requires that web browsers use WebKit on iOS.[47]

Blink

80. Google created Blink in 2013 by forking WebKit. Its stated reason for doing so was that its Chromium browser project (which is the basis for Google Chrome) 'uses a different multi-process architecture than other WebKit-based browsers, and supporting multiple architectures over the years has led to increasing complexity for both the WebKit and Chromium projects. This has slowed down the collective pace of innovation'.[48] One commentator at the time noted that Google had been participating more actively than Apple in developing WebKit, by some measures.[49]

81. Like Gecko and WebKit, Blink today benefits from open-source contributions from many organisations. Major contributors include Microsoft, Opera, Facebook, Adobe, Intel, IBM and Samsung.[50] Moreover, Blink is popular with browser vendors. Since Chrome's switch to Blink, many other browsers have followed suit or adopted the engine upon entry to the browser market, including Opera, Yandex, Samsung Internet, Brave, Vivaldi and Edge.[51]

EdgeHTML

82. When it launched its new Edge browser in 2015 – a replacement for IE – Microsoft forked its proprietary Trident browser engine to create EdgeHTML.[52] One of Microsoft's main motivations for forking Trident was to make it easier to ensure web compatibility.[53] However, Microsoft ultimately decided that replacing EdgeHTML with Blink would best ensure web compatibility, and in 2020 it switched to using Blink as its browser engine for Edge.[54]

---

[47] We assess the impacts of this restriction in Chapter 5 of our interim report.
[48] See: Google Open Source Blog (googleblog.com).
[49] See: Hypercritical: Code Hard or Go Home.
[50] See: Git at Google (googlesource.com).
[51] For Yandex and Opera see: thenextweb.com; For Samsung see: Smashing Magazine; For Vivaldi see: Vivaldi blog; For Brave see: Brave.
[52] See: A break from the past: the birth of Microsoft's new web rendering engine.
[53] See: Building a more interoperable Web with Microsoft Edge.
[54] See: Windows Experience Blog.

Presto

83. Opera originally used its own proprietary browser engine, Presto, which debuted in 2003.[55] However, in 2013 Opera switched to using Blink.[56]

Others

84. As discussed above, most browsers are now powered by Google's browser engine Blink and the supply of browser engines is highly concentrated (as discussed in Chapter 5 in section on shares of supply).

85. There have been a limited number of entrants over the past decade, including:

- The open-source browser engine Goanna, a fork of Gecko which is stewarded by Moonchild Productions. Officially launched in 2015, Goanna powers both of Moonchild Productions' open-source web browsers, PaleMoon and Basilisk.[57] The browser engine has also been adopted by K-Meleon and Mypal.[58,59]

- The proprietary browser engine Flow, launched by UK-based developer Ekioh in its browser (also called Flow) in late 2020.[60] Unlike most browser engines, Flow is not a fork of a previous browser engine codebase. Ekioh is aiming to differentiate Flow from Blink by building a browser for uses where a new browser engine would have clear benefits, such as better performance.[61,62] Flow is however not currently available on mobile devices.[63]

86. To date, these browser engines have attracted very limited usage.

87. As discussed above, in-app browsers can use a customised engine. Embedding WebView into an app allows developers to innovate on the performance and features for their in-app browser. We have heard that large app developers have experimented creating a custom browser engine in order to build innovative performance improvements. Given the growth of in-app

---

[55] See: Dev.Opera — Opera Mini server upgrade.
[56] See: A First Peek at Opera 15 for Computers.
[57] See: Pale Moon adopts new Goanna browser engine, fine-tunes interface (betanews.com).
[58] K-Meleon (kmeleonbrowser.org).
[59] Mypal - Official Website (mypal-browser.org).
[60] Unlike many other modern browser engines, Flow is not a fork of an existing codebase.
[61] Flow Browser | The parallel, multithreaded HTML browser (ekioh.com).
[62] Flow: A lightweight browser with a new rendering engine (fastcompany.com).
[63] Flow RFI received Feb 10, 2022, Q10a.

browsing it is likely that these customised browser engines may become more important in the future.

88. Figure F.10 sets out the timeline of modern browser engine development, illustrating that WebKit, Blink and Gecko are the only three major browser engines that continue to be under active development.

Figure F.10: Timeline of modern browser engine development