

Browser Extension Standards: How Google Monopolized and Exploited the Web Browser Industry

Kento Nishi
Harvard University

December 8, 2022

Introduction

Over the past several decades, the web has become an irreplaceable part of our lives. In particular, web standards have made it possible for full-fledged applications and services such as YouTube and Amazon to run entirely in the browser. Interestingly, modern web standards are democratized: for example, while the JavaScript programming language was created under Netscape in 1995, it was standardized as ECMAScript by ECMA International in 1997.¹ Similarly, HTML and CSS were handed off to the World Wide Web Consortium (W3C) soon after they became popular,² and the US Department of Defense's TCP/IP protocol was standardized by the Internet Engineering Task Force (IETF) in 1981 after gaining widespread adoption in the late 1970s.³ Most existing historical literature regarding web standards focus on the standardization of programming languages and networking protocols.

1. *JavaScript History* [in en-US], accessed December 7, 2022, https://www.w3schools.com/js/js_history.asp.

2. *A brief history of CSS until 2016*, accessed December 7, 2022, <https://www.w3.org/Style/CSS20/history.html>; *The history of the Web - W3C Wiki*, accessed December 7, 2022, https://www.w3.org/wiki/The_history_of_the_Web.

3. Amy Slaton and Janet Abbate, "The Hidden Lives of Standards: Technical Prescriptions and the Transformation of Work in America" [in eng], in *Technologies of power: essays in honor of Thomas Parke Hughes and Agatha Chipley Hughes* (Cambridge, Massachusetts ; London, England: The MIT Press, 2001), 125–126, ISBN: 978-0-262-26703-8; *TCP/IP 25th Anniversary* [in en-US], accessed December 7, 2022, <https://www.ietfjournal.org/tcpip-25th-anniversary/>.

However, there is another side to the web that is less highlighted yet equally as important: browser extensions. Browser extensions are add-on applications which have the unique ability to modify the functionality of websites and the browser itself. Typically, applications (web applications, mobile apps, browsers, etc.) are self-contained in that they do not interfere with other software—meanwhile, browser extensions are built with the express purpose of modifying other developers’ software.⁴ For example, I personally develop an extension called “HyperChat” which adds optimization and customization options to YouTube’s interface.⁵ On desktop browsers such as Chrome, Firefox, and Edge, HyperChat directly modifies the contents of YouTube.com—however, HyperChat cannot work the same way on mobile devices. This is because Android and iOS do not provide a mechanism for apps to extend or modify other apps (such as the YouTube mobile app). Browser extensions are an outlier in the way that the software industry generally operates because end users and extension developers can manipulate a piece of software without obtaining approval from its corresponding developer(s).

While extensions have significantly more privileges than self-contained applications, they are not unlimited in power. Over the years, browser vendors such as Google, Mozilla, Microsoft, and Apple have set limits on extensions’ capabilities by establishing browser-specific extension standards. However, unlike web standards such as HTML, browser extension standards themselves are not standardized—each vendor is free to establish their own standard. This freedom initially led to healthy competition between standards in the late 1990s through the early 2010s, but since then, all major browser vendors have converged on standards which closely mimic Google’s “Manifest” standard. In other words, Google’s Manifest standard has become the “standard” standard.

Although standardization can be beneficial in reducing barriers to entry for extension developers, Google’s monopoly over browser extension standards has become controver-

4. *Extensions and Apps in the Chrome Web Store - Google Chrome*, accessed December 7, 2022, https://web.archive.org/web/20140802091824/https://developer.chrome.com/webstore/apps_vs_extensions.

5. *HyperChat* [in en], accessed December 7, 2022, <https://chrome.google.com/webstore/detail/hyperchat-by-livetl/naipgebhooiiccifflecbffmjbabdbh>; *HyperChat* [in en-US], accessed December 7, 2022, <https://addons.mozilla.org/en-US/firefox/addon/hyperchat/>.

sial in recent years. Most notably, Google’s Manifest Version 3 (Manifest V3) announced in 2020 has faced significant backlash because it scales back on many core capabilities of a browser extension. For instance, extensions now have limited ability to intercept network requests, crippling most adblockers and privacy-enhancing extensions.⁶ Google claims these changes were implemented in consideration of end users’ privacy and security;⁷ however, Manifest V3’s technical details do not reflect this claim, and they instead point to Google’s capitalistic self-interests of diminishing adblockers and tracking prevention extensions. The undesirable state of current browser extension standards thus raises an important question: how did we get to this point, and what could have been done to prevent this degeneration?

In this paper, I lay out a framework similar to Microsoft’s infamous “Embrace, Extend, Extinguish” philosophy⁸ to explain Google’s strategy for acquiring near-totalitarian control over browser extension standards. A thorough analysis of the history of browser extensions reveals that Google strategically placed itself in a monopolistic position by executing the following five steps: (1) *Observe*, (2) *Propose*, (3) *Iterate*, (4) *Await*, and (5) *Exploit*. Frameworks such as “Embrace, Extend, Extinguish” have been useful in identifying and protesting monopolistic corporate behavior in the past⁹—my hope is that my analysis of extension standards under a similar lens can help prevent future standard takeovers with emerging technologies. For instance, it seems highly likely that Virtual Reality (VR) and Augmented Reality (AR) will spawn a new age of extension-like software—armed

6. Daly Barnett, *Chrome Users Beware: Manifest V3 is Deceitful and Threatening* [in en], December 2021, accessed December 7, 2022, <https://www.eff.org/deeplinks/2021/12/chrome-users-beware-manifest-v3-deceitful-and-threatening>; Alexei Miagkov and Bennett Cyphers, *Google’s Manifest V3 Still Hurts Privacy, Security, and Innovation* [in en], December 2021, accessed December 7, 2022, <https://www.eff.org/deeplinks/2021/12/googles-manifest-v3-still-hurts-privacy-security-innovation>.

7. *Web Request and Declarative Net Request: Explaining the impact on Extensions in Manifest V3* [in en], accessed December 7, 2022, <https://blog.chromium.org/2019/06/web-request-and-declarative-net-request.html>.

8. US Department of Justice, *After Netscape refused Microsoft’s offer to divide the browser market, Microsoft embarked on a predatory campaign to eliminate the browser threat* | Department of Justice, accessed December 7, 2022, <https://www.justice.gov/atr/file/705216>.

9. *Embrace, extend, extinguish: How Google crushed and abandoned the RSS industry* [in en], accessed December 7, 2022, <https://www.zdnet.com/article/embrace-extend-extinguish-how-google-crushed-and-abandoned-the-rss-industry/>; Matt Rickard, *Embrace. Extend. Extinguish.*, March 2022, accessed December 7, 2022, <https://matt-rickard.com>.

with a framework for understanding extension standards, we may be able to prevent the VR/AR-equivalent of Google from stifling the extendibility and customizability of these technologies. The following is a historical and technical overview of each of the five aforementioned phases; through these sections, we will come to understand modern browser extension standards, how they came to be, and what can be done to protect new technologies against similar devolutions going forward.

Step 1: *Observe* the Competition

The Early Days

Many might say the 90s were a simpler time—in terms of web technologies, however, the mid-to-late-90s were in fact a very chaotic time. 1993 saw the introduction of Mosaic (the first graphical web browser), with the Netscape Browser and Microsoft’s Internet Explorer quickly following up in 1994 and 1995, respectively.¹⁰ Because the W3C was only formed in 1994, web standards were still being developed;¹¹ meanwhile, multiple paradigms such as Netscape’s JavaScript and Microsoft’s JScript were competing for developer adoption.¹² At the time, it was unclear which technologies would become most dominant.

Browsers of this era were not easily extendable like they are today. During this period, most browsers were extended via plug-ins: for example, plug-ins such as Adobe Flash and Java could run dynamic content in the browser by leveraging browser-provided application programming interfaces (APIs) such as ActiveX and NPAPI.¹³ While these plug-ins did indeed extend the browser, they were still far from extensions as we know them today; most notably, plug-ins gave customization power to website developers, while

10. *Browser History: Epic power struggles that brought us modern browsers* [in en], accessed December 7, 2022, <https://www.mozilla.org/en-US/firefox/browsers/browser-history/>.

11. *The history of the Web - W3C Wiki*.

12. *What’s the difference between JavaScript and JScript?* [In en-us], Section: JavaScript, May 2020, accessed December 7, 2022, <https://www.geeksforgeeks.org/whats-the-difference-between-javascript-and-jscript/>.

13. *Microsoft Announces ActiveX Technologies* [in en-US], March 1996, accessed December 7, 2022, <https://news.microsoft.com/1996/03/12/microsoft-announces-activex-technologies/>; *Mozilla Plugins project*, accessed December 7, 2022, <https://www-archive.mozilla.org/projects/plugins/>.

modern extensions give customization power to end users. Customization options for end users were very primitive in the 90s—the most one could do to modify a website was to apply User Style Sheets to change basic properties such as fonts and colors.¹⁴ To unlock further customization options on the user side, Microsoft unveiled Internet Explorer 4 in 1999, adding support for “Explorer Bars” and context menu add-ons.¹⁵ By the time the 2000s rolled in, major web companies were starting to take notice of these new features—for instance, Google introduced its popular “Google Toolbar” for Internet Explorer 5 in 2000.¹⁶

One major drawback of Internet Explorer’s approach to browser extendibility was its barrier to entry. For one, Internet Explorer toolbars needed to be written in C++,¹⁷ a language which is not commonly used in web development. Toolbars also had to be distributed by each individual developer on their own website. This was inconvenient for both toolbar developers and end users: hosting a server to distribute a plugin required significant effort and money, and the unmoderated nature of the internet meant end users were susceptible to unknowingly installing malicious toolbars. Furthermore, toolbars tended to be insecure because they were native binaries with access to users’ entire systems.¹⁸ As toolbar developer Erik Thompson pointed out in an online tutorial in 2001, Internet Explorer’s extendibility APIs certainly had “some room for improvement.”¹⁹

14. *User Stylesheets in CSS*, accessed December 7, 2022, <https://dbaron.org/css/user/>.

15. *Microsoft Releases Internet Explorer 4.0 for UNIX* [in en-US], February 1998, accessed December 7, 2022, <https://news.microsoft.com/1998/02/24/microsoft-releases-internet-explorer-4-0-for-unix/>.

16. *Google Launches The Google Toolbar – News announcements – News from Google – Google*, accessed December 7, 2022, <http://googlepress.blogspot.com/2000/12/google-launches-google-toolbar.html>.

17. *Google Launches The Google Toolbar – News announcements – News from Google – Google*.

18. *Answer to “How malicious 3rd party toolbars on IE could be?”*, August 2015, accessed December 8, 2022, <https://security.stackexchange.com/a/96172>.

19. Erik Thompson, *Internet Explorer Toolbar (Deskband) Tutorial* [in en-US], August 2001, accessed December 7, 2022, <https://www.codeproject.com/Articles/1323/Internet-Explorer-Toolbar-Deskband-Tutorial>.

The Rise of Mozilla Firefox

In 2002, Mozilla (formerly known as Netscape) unveiled Firefox,²⁰ and three years later, it implemented a novel interface for what it coined as “extensions.”²¹ In contrast to Internet Explorer, Firefox provided an API based on JavaScript and XUL²²—this made extensions significantly easier to develop for the majority of web developers. Furthermore, Mozilla created an official extensions marketplace, easing the burden on extension developers and making extension installations more safe and convenient for end users.²³ These improvements helped Firefox eat away at Internet Explorer’s popularity—by the end of 2009, Firefox claimed about 21% of the market in comparison to Internet Explorer’s 66.4%, firmly establishing itself as the second most popular browser.²⁴

Google Lurks in the Shadows

Throughout this time, Google was quietly monitoring the situation from the sidelines. Eric Schmidt, Google’s then CEO, was acutely aware of how competitive the browser market is—in fact, he resisted Google employees’ push to build its own browser for six years to avoid dipping its toes into the “browser wars.”²⁵ Despite this internal pushback, Google began to silently hire former Mozilla developers and kickstarted development of an entirely new browser named Chrome in 2006.²⁶

When designing Chrome, Google made use of all of its learnings accumulated over years of observing other browser vendors. For one, Google chose to make Chromium, the

20. *Mozilla.org launches Mozilla 1.0* [in en-US], accessed December 8, 2022, <https://blog.mozilla.org/press/2002/06/mozilla-org-launches-mozilla-1-0>.

21. *Extensions - Devmo*, October 2005, accessed December 8, 2022, <https://web.archive.org/web/20051029013647/http://developer.mozilla.org/en/docs/Extensions>.

22. *How to develop a Firefox extension* [in en-US], accessed December 7, 2022, <https://blog.mozilla.org/addons/2009/01/28/how-to-develop-a-firefox-extension/>.

23. Rietta Inc, *Overview, Firefox Extension Development Tutorial* [in en], accessed December 8, 2022, <https://rietta.com/firefox/Tutorial/overview/index.html>.

24. *TheCounter.com: The Full-Featured Web Counter with Graphic Reports and Detailed Information*, accessed December 7, 2022, <https://web.archive.org/web/20091207111955/http://www.thecounter.com/stats/2009/December/browser.php>.

25. *Sun Valley: Google CEO Eric Schmidt Didn’t Want to Build Chrome Initially, He Says - Digits - WSJ*, accessed December 8, 2022, <https://web.archive.org/web/20120320220630/http://blogs.wsj.com/digits/2009/07/09/sun-valley-schmidt-didnt-want-to-build-chrome-initially-he-says/tab/print/>.

26. *Sun Valley: Google CEO Eric Schmidt Didn’t Want to Build Chrome Initially, He Says - Digits - WSJ*.

internal base of Chrome, fully open-source.²⁷ This move was certainly inspired by the fact that the W3C’s open web standards had come to dominate over Microsoft’s proprietary web standards in the previous decade.²⁸ Google understood that making Chromium open source would allow other vendors to buy into its standards, all while it retains full authority over the specifics of the standards themselves. Indeed, in Google’s official comic book from 2008, Open Source Programs Manager Chris DiBona remarks that “[s]ome of [Google Chrome’s practices] may become standards—some might not. But—since it’s open source—other browser developers can take what they want out of it.”²⁹

Additionally, while observing the browser wars from the sidelines for many years, Google internally conducted research to analyze flaws in other browsers’ extendibility features. For instance, in a technical paper titled “Protecting Browsers from Extension Vulnerabilities” published in 2010, Google engineers identified many security flaws in the Firefox extension API and proposed the Manifest standard as a more robust alternative.³⁰ These technical improvements are the subject of the next section of this paper.

Step 2: *Propose a Technically Superior Standard*

Although Mozilla vastly improved upon Internet Explorer by providing a JavaScript API and hosting an official extensions store, Firefox still shared many of the same security vulnerabilities as Internet Explorer. For instance, extensions still had access to system-level privileges—in one demonstration, an attacker could trick an extension into installing arbitrary code, thereby forfeiting control to the user’s mouse and keyboard to a remote attacker.³¹ At Google, these vulnerabilities “raise[d] the question of whether browser ex-

27. *A fresh take on the browser* [in en], accessed December 7, 2022, <https://googleblog.blogspot.com/2008/09/fresh-take-on-browser.html>.

28. *How JavaScript Became the Dominant Language of the Web | Blog | Lform* [in en-us], accessed December 8, 2022, <https://lform.com/blog/post/how-javascript-became-the-dominant-language-of-the-web/>.

29. *Google Chrome: Behind the Open Source Browser Project*, accessed December 7, 2022, https://www.google.com/googlegbooks/chrome/big_00.html.

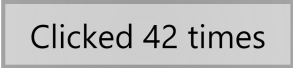
30. Adam Barth et al., “Protecting Browsers from Extension Vulnerabilities,” in *Network and Distributed System Security Symposium* (2010).

31. Christiaan008, *DEFCON 17: Abusing Firefox Addons*, January 2011, accessed December 8, 2022, <https://www.youtube.com/watch?v=vffa4FshXWY>.

tensions require such a high level of privilege,” leading the Chrome engineers to “propose a new browser extension system that improves security by using least privilege, privilege separation, and strong isolation.”³² Throughout this section, we will dissect some parts of Google’s Manifest standard to understand how Google’s proposal constituted an undeniable improvement over the Firefox extension API.

A Brief Technical Overview of Websites and Firefox Extensions

Every website consists of two parts: the “window” and the “document.” The window is primarily responsible for executing a website’s code and managing its hidden internal states. In contrast, the document represents what is rendered on the screen to be displayed to the user. For example, consider the simple click-counter website below:



Clicked 42 times

In this example, the window keeps track of how many times the button has been clicked. Meanwhile, the clickable button with the “Clicked *X* times” text resides within the document, which is displayed to the user. When the button is clicked, the document notifies the window of this event—conversely, when the internal click counter state changes, the window notifies the document to update the displayed text.

When Mozilla implemented its extension API, it chose to provide extensions with direct access to both the window and the document.³³ This meant that extensions had virtually full control over any page; while this privilege was useful in some regards, it came with severe side effects. Most crucially, bugs in both extensions and Firefox itself often enabled websites to acquire privileged access over extensions through the shared window.³⁴ As mentioned before, extensions were granted system-level privileges; this meant that websites, which are supposed to be fully sandboxed from the rest of the system, could potentially gain unrestricted access by hijacking a vulnerable extension. These security

32. Barth et al., “Protecting Browsers from Extension Vulnerabilities.”

33. *How to develop a Firefox extension*; Barth et al., “Protecting Browsers from Extension Vulnerabilities.”

34. Barth et al., “Protecting Browsers from Extension Vulnerabilities.”

holes were not purely theoretical: for instance, a popular extension called GreaseMonkey contained a critical vulnerability which enabled any website to circumvent the Cross Origin Resource Sharing policy (CORS) to make requests to any other website illegally.³⁵

Google Chrome’s Manifest Standard

In response to these blatant security flaws, Google proposed several degrees of separation between websites, extensions, and the browser. Most revolutionary was the idea of a “content script”—content scripts allowed extensions to safely access websites’ documents without exposing the window.³⁶ In doing so, Google essentially minimized the “attack surface” of extensions and prevented websites from gaining any privileged access. While this also meant that extensions could not modify websites’ windows directly, this was only a minor drawback as extensions could instead “inject” code into the document of a webpage, which is then run in the webpage window.³⁷ Script injection ensured that an extension can manipulate webpages’ windows, but not the other way around—this change was beneficial for both extension developers and end users.

Another innovation was the introduction of fine-grained permissions alongside a “background page.”³⁸ In essence, the content script was solely dedicated to manipulating webpages, and it could not be used to take privileged actions such as monitoring and overwriting network traffic. Such APIs were made exclusive to “background scripts” within each extension’s persistent background page, and users were required to grant these permissions prior to installation.³⁹ The Manifest standard also simplified the developer workflow

35. Simon Willison, *Understanding the Greasemonkey vulnerability* [in en-gb], accessed December 7, 2022, <http://simonwillison.net/2005/Jul/20/vulnerability/>.

36. Barth et al., “Protecting Browsers from Extension Vulnerabilities”; *Content Scripts - Google Chrome Extensions - Google Code*, August 2012, accessed December 7, 2022, https://web.archive.org/web/20120813045827/http://developer.chrome.com/extensions/content_scripts.html.

37. Rob W, *Answer to "Chrome extension - retrieving global variable from webpage"*, March 2012, accessed December 7, 2022, <https://stackoverflow.com/a/9636008>.

38. *Background Pages - Google Chrome Extensions - Google Code*, August 2012, accessed December 7, 2022, https://web.archive.org/web/20120812230011/http://developer.chrome.com/extensions/background_pages.html.

39. Barth et al., “Protecting Browsers from Extension Vulnerabilities”; *Background Pages - Google Chrome Extensions - Google Code*.

for specifying these permissions: Manifest introduced the “manifest.json” file,⁴⁰ which was more intuitive for extension developers to work with than Firefox’s “install.rdf” file. Overall, these changes amounted to a significant improvement in both the developer and user experience, and over time, both extension developers and end users were lured into switching away from Firefox and Internet Explorer.

Step 3: *Iterate* to Corner the Competition

Chrome Web Store and Manifest V2

Google’s plans with extensions were only just getting started—in 2010, Google copied Mozilla’s extensions store by launching a centralized extension marketplace called the Chrome Web Store.⁴¹ The Chrome Web Store acquired nearly as many extensions as Mozilla’s addons marketplace within one year,⁴² further reinforcing Chrome as the go-to browser for both end users and extension developers. Google also made the Chrome Web Store compatible with all Chromium-based browsers and disabled installations from outside the Chrome Web Store in 2014⁴³—this allowed Google to effectively moderate all extensions to ensure their safety and legitimacy. Furthermore, in 2012, Google announced Manifest Version 2 (Manifest V2), an incremental improvement over Manifest Version 1 (Manifest V1).⁴⁴ To some developers’ dismay, Manifest V2 was not backwards-compatible in that some changes were required on the extension developers’ side to migrate to the new standard.⁴⁵ However, these changes constituted only a minor incon-

40. *Management - Google Chrome Extensions - Google Code*, August 2012, accessed December 7, 2022, <https://web.archive.org/web/20120812230116/http://developer.chrome.com/extensions/management.html>.

41. Rory Reid, *Google launches Web Store for Chrome apps* [in en], accessed December 7, 2022, <https://www.cnet.com/tech/services-and-software/google-launches-web-store-for-chrome-apps/>.

42. M. G. Siegler, *Chrome Appears To Have Hit 10,000 Extensions, Inching Closer To Firefox* [in en-US], December 2010, accessed December 7, 2022, <https://techcrunch.com/2010/12/10/chrome-extension-numbers/>.

43. *Make Sure to Get Your Extension in the Chrome Web Store* [in en], accessed December 7, 2022, <https://blog.chromium.org/2014/02/make-sure-to-get-your-extension-in.html>.

44. *Tutorial: Migrate to Manifest V2 - Google Chrome*, accessed December 7, 2022, https://web.archive.org/web/20121110195941/http://developer.chrome.com/extensions/tut_migration_to_manifest_v2.html.

45. W, *Answer to "Chrome extension - retrieving global variable from webpage"*.

venience because Manifest V2 did not take away from the underlying functionality of Chrome’s existing extension API.⁴⁶

Manifest V2’s Innovations

Most impactful of Manifest V2’s changes were Web Accessible Resources (WAR) and the “Content Security Policy” (CSP), two security-oriented mechanisms for further protecting extensions against attacks from websites. With the content script, Manifest V1 prevented websites from directly hijacking extensions; however, it did not prevent websites from accessing extensions’ static assets (such as icons, images, fonts, etc.).⁴⁷ Furthermore, during the two years in which Manifest V1 had been in effect, buggy usages of “inline” and “eval” scripts in extensions were creating alarming security holes.⁴⁸ The introduction of WAR and the CSP effectively sealed these issues in a semi-discretionary way—Manifest V2 provided strict default options, but allowed extension developers to “relax” these options if needed.⁴⁹ Migration from Manifest V1 to Manifest V2 was thus fairly straightforward for most developers, and this shift was completely unnoticeable for the majority of Chrome users. Existing Chrome extensions carried on as if nothing had changed, only with enhanced security under the hood.⁵⁰

Competing Vendors Struggle to Respond

While Manifest became more polished and mature, other vendors struggled to match Chrome’s robust API. In particular, Internet Explorer had made little to no changes to its API since it first launched, and Mozilla could not replicate Chrome’s security features in Firefox. As time went on, Internet Explorer and Firefox users became increasingly

46. *More secure extensions, by default* [in en], accessed December 8, 2022, <https://blog.chromium.org/2012/02/more-secure-extensions-by-default.html>.

47. *Manifest - Web Accessible Resources* [in en], accessed December 7, 2022, https://developer.chrome.com/docs/extensions/mv2/manifest/web_accessible_resources/.

48. *Content Security Policy* [in en], accessed December 7, 2022, <https://developer.chrome.com/docs/apps/contentSecurityPolicy/>.

49. *Manifest - Web Accessible Resources; Content Security Policy*.

50. *More secure extensions, by default*.

concerned about the safety of their extensions. For example, Colby University’s IT department advised its students to “not install any add-ons [in Internet Explorer], such as the ‘Ask toolbar,’”⁵¹ and the UK Government warned that “[o]rganisations should consider the increased exposure to malware when enabling [Firefox] add-ons as they do not usually run inside a sandbox.”⁵²

2011 also saw Apple’s introduction of Safari Extensions with Safari 5.⁵³ Apple adopted Google’s philosophy of browser extendibility with Safari Extensions: for instance, Apple implemented its own version of content scripts called “injected scripts.”⁵⁴ Apple even addressed Chrome extension developers directly **with bold text** in its documentation to emphasize that “[i]njected scripts run in their own “isolated world” in Safari, just as in Chrome.”⁵⁵ Although Apple’s standard was conceptually similar to that of Chrome, it was more primitive and required developers to engineer tedious workarounds to carry out basic tasks such as intercepting network requests.⁵⁶ Additionally, despite Apple’s best efforts, Safari’s developer documentation seemed too “intimidating” for the average extension developer.⁵⁷ Due to these factors, the Safari Extension API also failed to gain traction.

As Chrome emerged as the most popular browser over the next several years,⁵⁸ it became abundantly clear that Google’s Manifest standard had won the battle between extension standards. It would only be a matter of time before every browser vendor surrendered and followed Google’s lead.

51. *Internet Browser Security Recommendations | Information Technology Services* [in en-US], accessed December 7, 2022, <https://www.colby.edu/its/internet-browser-security-recommendations/>.

52. *Browser Security Guidance: Mozilla Firefox - Publications - GOV.UK*, accessed December 8, 2022, <https://web.archive.org/web/20160416101110/https://www.gov.uk/government/publications/browser-security-guidance-mozilla-firefox>.

53. *Apple Releases Safari 5* [in en-US], accessed December 7, 2022, <https://www.apple.com/newsroom/2010/06/07Apple-Releases-Safari-5/>.

54. *Safari Extensions Conversion Guide: Understanding the Extensions Architecture*, June 2010, accessed December 7, 2022, <https://web.archive.org/web/20100613010707/http://developer.apple.com/safari/library/documentation/UserExperience/Conceptual/SafariExtensionsConversionGuide/Chapters/Architecture.html>.

55. *Safari Extensions Conversion Guide*.

56. qdev, *Answer to “Writing an equivalent to Chrome’s onBeforeRequest in a Safari extension”*, March 2015, accessed December 7, 2022, <https://stackoverflow.com/a/28808411>.

57. Senior Contributor, *How to create a Safari extension* [in en], accessed December 7, 2022, <https://www.macworld.com/article/209928/safariextension.html>.

58. *Chrome | Description, Features, & Facts | Britannica* [in en], accessed December 8, 2022, <https://www.britannica.com/technology/Chrome>.

Step 4: *Await Concession*

Vendors Jump Ship, One After Another

In 2015, Microsoft shocked the web developer community when it announced that it would be adopting Google’s Manifest standard for its all-new Edge browser.⁵⁹ This move was surprising because Edge was not built on the Chromium project—rather, it ran on its own EdgeHTML and Chakra engines.⁶⁰ By implementing Manifest in Edge, Microsoft had admitted defeat and conceded control over its standard in exchange for increased developer and user-friendliness. With this change in place, “existing Chrome extension developers [were] able to migrate their extensions to Microsoft Edge (EdgeHTML) with minimal changes,”⁶¹ and sure enough, popular extensions such as Adblock, LastPass, and Evernote appeared on the Microsoft Store soon after launch.⁶² Microsoft was not alone in abandoning its own standard for long—only several months after the release of Edge, Mozilla announced its plans to move away from its XPCOM/XUL APIs in favor of its own adaptation of the Manifest standard dubbed “WebExtensions.”⁶³ Apple also immediately followed suit: in 2016, less than a year since the release of Edge, Apple released Safari Technology Preview Release 109 with an official toolkit for developers to “port” their Manifest-based extensions to Safari.⁶⁴

Just two years later, Microsoft went one step further: it announced that it will be replacing its own EdgeHTML and Chakra engines with Google’s Chromium project in 2018.⁶⁵

59. MSEdgeTeam, *Extensions - Legacy Microsoft Edge developer docs* [in en-us], accessed December 7, 2022, <https://learn.microsoft.com/en-us/archive/microsoft-edge/legacy/developer/extensions/>.

60. *Inside Microsoft’s New Rendering Engine For The “Project Spartan”* [in en], Section: General, 0, accessed December 8, 2022, <https://www.smashingmagazine.com/2015/01/inside-microsofts-new-rendering-engine-project-spartan/>.

61. MSEdgeTeam, *Extensions - Legacy Microsoft Edge developer docs*.

62. *Microsoft is working on a Chrome extension porting tool, a CSS tutorial, and Searchkit 0.8—SD Times news digest: March 21, 2016* [in en-US], March 2016, accessed December 7, 2022, <https://sdtimes.com/chrome/microsoft-is-working-on-a-chrome-extension-porting-tool-a-css-tutorial-and-searchkit-0-8-sd-times-news-digest-march-21-2016/>.

63. *The Future of Developing Firefox Add-ons* [in en-US], accessed December 7, 2022, <https://blog.mozilla.org/addons/2015/08/21/the-future-of-developing-firefox-add-ons/>.

64. *Release Notes - Safari Technology Preview - Safari - Apple Developer*, accessed December 7, 2022, <https://developer.apple.com/safari/technology-preview/release-notes/>.

65. Windows Experience Blog, *Microsoft Edge: Making the web better through more open source collaboration* [in en-US], December 2018, accessed December 7, 2022, <https://blogs.windows.com/>

Previously, Chrome extension developers needed to “port” their extensions to Edge—with this monumental transition, Microsoft invested full trust in Google’s Manifest standard and opened the doors for Edge users to directly install Chrome extensions from the Chrome Web Store.⁶⁶ Apple also became remarkably invested in the Manifest standard: in early 2019, it completely deprecated “legacy” Safari extensions in favor of its Manifest-based Web Extension APIs with Safari Technical Preview Release 80 in 2019.⁶⁷

Google Patiently Waits

While vendors scrambled to raise their white flags, Google remained completely still. Manifest V2 came only two years after Manifest V1—in contrast, Manifest V2 had been in effect for over 7 years by 2019, with almost no changes since 2012. Even though Manifest V2’s ideas had withstood the test of time, some of its practices had become somewhat outdated. For example, the release of ECMAScript 6 (ES6) in 2015 popularized the use of “promises” over “callbacks,”⁶⁸ but over four years later, Chrome’s APIs still depended on callbacks. This stagnancy was wholly deliberate—although Google could easily integrate promises (as many “polyfill” developers had already done),⁶⁹ it prioritized stability over innovation to lure other browser vendors into fully adopting its standard more quickly. Had Manifest V2 been a moving goalpost while Microsoft developed Edge, for example, Microsoft may not have so readily committed to fully integrating Chromium and Manifest. For Google, staying still paid off: by mid-2019, every major browser vendor had adopted the Manifest standard. Google was now firmly in control.

windowsexperience/2018/12/06/microsoft-edge-making-the-web-better-through-more-open-source-collaboration/.

66. Paul Thurrott, *Yes, the New Microsoft Edge Will Support Chrome Extensions* [in en-US], December 2018, accessed December 7, 2022, <https://www.thurrott.com/windows/windows-10/194838/yes-the-new-microsoft-edge-will-support-chrome-extensions>.

67. *Apple Releases Safari 5*.

68. *A quick guide to JavaScript Promises* [in en], accessed December 7, 2022, <https://www.twilio.com/blog/2016/10/guide-to-javascript-promises.html>.

69. john ktejik john, *How do I use promises in a Chrome extension?*, Forum post, June 2018, accessed December 7, 2022, <https://stackoverflow.com/q/50844405>; *WebExtension browser API Polyfill*, original-date: 2016-10-06T19:02:50Z, December 2022, accessed December 7, 2022, <https://github.com/mozilla/webextension-polyfill>.

Step 5: *Exploit* Monopoly Power

Seeing that competing browser extension standards were on their way out, Google started work on Manifest Version 3 (Manifest V3) in late 2018.⁷⁰ In a blog post, Google declared that its goals with Manifest V3 were “to create stronger security, privacy, and performance guarantees” for its users.⁷¹ To achieve these ideals, Google laid out several major changes: it finally replaced callbacks with promises, introduced “service workers” as a replacement for background scripts, and deprecated the “blocking Web Request” API in favor of “Declarative Net Requests.”⁷² Manifest V3 was officially announced in 2020,⁷³ and as of the time of writing (December 2022), it is scheduled to come into full effect by mid-2023.⁷⁴

Manifest V3: The Good

Although Manifest V3 was poorly received, some of its changes are arguably positive. For one, promises are a significant stylistic improvement over callbacks.⁷⁵ Additionally, service workers can theoretically be more efficient than persistent background scripts because they are only activated when needed.⁷⁶ Even Mozilla, an outspoken critic of Manifest V3, supports Google’s service worker concept—in late 2019, it stated that it is “working on extension Service Workers in Firefox [not only] for compatibility reasons,” but also because the team “like[s] that they’re an event-driven environment with defined lifetimes, already part of the Web Platform with good cross-browser support.”⁷⁷ While a transition

70. *Trustworthy Chrome Extensions, by default* [in en], accessed December 7, 2022, <https://blog.chromium.org/2018/10/trustworthy-chrome-extensions-by-default.html>.

71. *Trustworthy Chrome Extensions, by default*.

72. *Overview of the Chrome Extension Manifest V3* [in en], accessed December 7, 2022, <https://developer.chrome.com/docs/extensions/mv3/intro/mv3-overview/>.

73. *Overview of the Chrome Extension Manifest V3*.

74. *Chrome Extensions Manifest V2 support timeline* [in en], accessed December 7, 2022, <https://developer.chrome.com/docs/extensions/mv3/mv2-sunset/>.

75. *A quick guide to JavaScript Promises*.

76. *Migrating from background pages to service workers* [in en], accessed December 8, 2022, https://developer.chrome.com/docs/extensions/mv3/migrating_to_service_workers/; *Service worker overview* [in en], accessed December 8, 2022, <https://developer.chrome.com/docs/workbox/service-worker-overview/>.

77. *Mozilla’s Manifest v3 FAQ* [in en-US], accessed December 8, 2022, <https://blog.mozilla.org/addons/2019/09/03/mozillas-manifest-v3-faq/>.

to promises and service workers may be inconvenient and burdensome in the short term, it will likely encourage extension developers to write more efficient event-based code in the future, improving the overall state of the browser extension ecosystem.

Manifest V3: The Bad

With Manifest V3, Google forcibly shoved a highly opinionated standard down browser vendors' and extension developers' throats. For example, Google was harshly criticized for its decision to deprecate the "blocking Web Request" API in favor of the more limiting "Declarative Net Request" API.⁷⁸ Previously, an extension under Manifest V1 and V2 (when granted relevant permissions) could intercept and rewrite network traffic imperatively and synchronously.⁷⁹ For example, adblocker extensions such as Adblock Plus, uBlock Origin, Ghostery Privacy Ad Blocker, and AdGuard heavily relied on the blocking Web Request API in the background script to act as ad-filtering middlemen.⁸⁰ Advertisement companies (like Google) constantly attempt to evade adblockers by changing their code⁸¹—to shoot down these moving targets, adblocker extensions previously leveraged the blocking Web Request API to dynamically match, scan, and block unwanted content.⁸²

With Manifest V3, this type of dynamic filtering was made impossible; under Declarative Net Requests, each extension must declare a static list of rules for the browser to

78. Barnett, *Chrome Users Beware*; Cyphers, *Google's Manifest V3 Still Hurts Privacy, Security, and Innovation*.

79. *chrome.declarativeNetRequest* [in en], accessed December 7, 2022, <https://developer.chrome.com/docs/extensions/reference/declarativeNetRequest/>; *webRequest.onBeforeRequest - Mozilla | MDN* [in en-US], accessed December 8, 2022, <https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/onBeforeRequest>.

80. Thomas Claburn, *Ad-block developers fear end is near for their extensions* [in en], accessed December 7, 2022, https://www.theregister.com/2022/06/08/google_blocking_privacy_manifest/; *Manifest V3: The Ghostery perspective*, accessed December 8, 2022, <https://www.ghostery.com/blog/manifest-v3-the-ghostery-perspective>; *AdGuard publishes the world's first ad blocker built on Manifest V3* [in en], accessed December 7, 2022, <https://adguard.com/en/blog/adguard-mv3.html>.

81. *I'm seeing ads in YouTube videos* [in en-US], accessed December 7, 2022, <https://helpcenter.getadblock.com/hc/en-us/articles/9738509282451-I-m-seeing-ads-in-YouTube-videos>; *The mystery of how intrusive YouTube ads are evading ad blockers* [in en], accessed December 7, 2022, <https://eyeo.com/blog/mystery-how-intrusive-youtube-ads-are-evading-ad-blockers>.

82. Claburn, *Ad-block developers fear end is near for their extensions*; *AdGuard publishes the world's first ad blocker built on Manifest V3*.

filter on its behalf.⁸³ Although some adblocker developers have successfully converted many of their dynamic rules into the declarative format, the static nature of the Declarative Net Request API still severely cripples adblockers across thousands of sites.⁸⁴ For instance, the developers of AdGuard lament that “[t]he problem with declarative rules is rather obvious: their syntax severely limits what our extension can do.”⁸⁵ Furthermore, the AdGuard developers express frustration with the reality that “there is nothing [the AdGuard team] can do about it, other than hope that the Chrome developers will improve it over time.”⁸⁶

As an advertising company, Google directly benefits from its own decision to cripple adblockers. Google does not publicly admit this motive, however; instead, Google insists that its Declarative Net Request implementation is the optimal way forward because it allows extensions to “perform content blocking without needing access to all of a user’s personal information.”⁸⁷ However, the Declarative Net Request API is far from the best solution to privacy—quite ironically, its limitations severely cripple privacy-focused tracking prevention extensions.⁸⁸ For example, Privacy Badger previously blocked “invisible” trackers on the internet by leveraging blocking Web Requests under Manifest V2, but can no longer do so with Manifest V3.⁸⁹ In Privacy Badger developer Alexei Miasnikov’s words, “[t]he declarativeNetRequest API is an entirely inadequate replacement as it supports onBeforeRequest blocking and redirection only (not header/body inspection or modification), and seems to support (a limited number of) hardcoded rules only.”⁹⁰

Given that background scripts and blocking Web Requests played such an important role in privacy-related extensions, it is difficult to argue that Google’s changes are a mar-

83. *Web Request and Declarative Net Request*.

84. *AdGuard publishes the world’s first ad blocker built on Manifest V3*.

85. *AdGuard publishes the world’s first ad blocker built on Manifest V3*.

86. *AdGuard publishes the world’s first ad blocker built on Manifest V3*.

87. *Web Request and Declarative Net Request*.

88. Barnett, *Chrome Users Beware; Cyphers, Google’s Manifest V3 Still Hurts Privacy, Security, and Innovation*.

89. *Will the proposed Manifest V3 changes to Chrome break Privacy Badger?* · Issue #2273 · EFF-Forg/privacybadger [in en], accessed December 7, 2022, <https://github.com/EFForg/privacybadger/issues/2273>.

90. *Will the proposed Manifest V3 changes to Chrome break Privacy Badger?*

ketable improvement for end users’ privacy over Manifest V2. Rather, Google’s main business interests suggest that Manifest V3’s restrictions are designed to further strengthen Google’s stranglehold on the online advertisement industry.

Competing Vendors Respond

Competing vendors’ responses to Manifest V3 were mixed. Some vendors supported Google’s efforts: for example, Microsoft added support for Manifest V3 in 2020, stating that the standard aligns with its “dedication to enhance privacy, security[, and] performance for the benefit our end users as well as to allow developers to extend [and] provide rich experiences in Microsoft Edge.”⁹¹ Apple was also accepting of Google’s proposals—it shipped Safari 15.4 with support for Manifest V3 extensions in early 2022, citing its “ongoing commitment to a cross-browser interoperable model for extensions.”⁹² Although Microsoft and Apple seem to be accepting of Manifest V3, is important to note that as of the time of writing, neither has finalized plans to remove existing Manifest V2 extensions from their respective extension marketplaces. This indicates that both vendors still have enough lingering reservations about Manifest V3 to warrant holding off Google’s recommended deprecation of existing Manifest V2 extensions.

Among all other browser vendors, reception of Manifest V3 was abysmal. Many vendors of Chromium-based browsers such as Vivaldi and Brave were especially outspoken about the negative effects of Manifest V3.⁹³ For example, Brave publicly criticized Google for “harming privacy and limiting user choice” in a series of accusatory Tweets.⁹⁴

91. Microsoft Edge Blog, *Manifest V3 changes are now available to test in Microsoft Edge* [in en-US], October 2020, accessed December 7, 2022, <https://blogs.windows.com/msedgedev/2020/10/14/extension-manifest-chromium-edge/>.

92. Jen Simmons, *New WebKit Features in Safari 15.4*, Section: News, March 2022, accessed December 7, 2022, <https://webkit.org/blog/12445/new-webkit-features-in-safari-15-4/>.

93. *Manifest V3, webRequest, and ad blockers* [in en], September 2022, accessed December 8, 2022, <https://vivaldi.com/blog/manifest-v3-webrequest-and-ad-blockers/>; Brave Software [@brave], *With Manifest V3, Google is harming privacy and limiting user choice. The bottom line, though, is that Brave will still continue to offer leading protection against invasive ads and trackers. Keep your privacy protected by switching to Brave today:* <https://t.co/4wVWi9bfK3> [in en], Tweet, September 2022, accessed December 8, 2022, <https://twitter.com/brave/status/1574822800971501568>.

94. Brave Software [@brave], *With Manifest V3, Google is harming privacy and limiting user choice. The bottom line, though, is that Brave will still continue to offer leading protection against invasive ads and*

Outside of the Chromium circle, Mozilla was also unhappy with Google’s changes. Among popular vendors, Mozilla was the most outgoing in terms of taking tangible action against Manifest V3. Most notably, it announced plans to both implement Manifest V3 and continue supporting Manifest V2 features such as blocking Web Requests for the foreseeable future “until there’s a better solution which covers all use cases [Mozilla] consider[s] important, since [Declarative Net Requests] as currently implemented by Chrome does not yet meet the needs of extension developers”.⁹⁵ For the majority of Chrome extension developers, however, developing a separate extension for users of a less popular browser such as Firefox may not be worth the additional work required. Therefore, many future Firefox extensions will likely be simple ports of Chrome extensions which do not make use of the blocking Web Request API. In short, advanced adblockers and privacy-enhancing extensions are likely to fade away in the long run, even if competing vendors continue to support features which Google removed in Manifest V3.

At this point, it is important to note that Google’s plans to “kill” adblockers and privacy-enhancing extensions would not have been possible had it not waited for competing vendors to concede in the previous step. For instance, both ES6 promises and service workers were well-known technologies by 2015.⁹⁶ Had Google kept up a consistent update cycle between Manifest versions and transitioned to Manifest V3 and Declarative Net Requests in 2015, Chrome’s adblockers and privacy-enhancing extensions would have been trumped by those of competing browsers such as Firefox or Edge. As a consequence of having less effective extensions, Chrome users could have plausibly switched to a different browser. It is only because Google had acquired a monopoly over extension standards that this hypothetical scenario did not play out—once every browser had adopted Manifest, Google could freely exploit its power without the risk of losing users to a competing browser.

trackers. Keep your privacy protected by switching to Brave today: <https://t.co/4wVWi9bfK3>.

95. *Manifest v3 update* [in en-US], accessed December 7, 2022, <https://blog.mozilla.org/addons/2021/05/27/manifest-v3-update/>.

96. *Service worker overview*.

Conclusion

Through an in-depth historical account of browser extensions, we have come to understand that Google’s monopolization and subsequent exploitation of browser extension standards was deliberate and methodical. In the first phase, Google observed its future competitors and took notes, jotting down features to copy and practices to avoid. In the second phase, Google leveraged its newfound knowledge to propose the Manifest standard, which significantly improved upon ideas introduced by Internet Explorer and Firefox while being more intuitive, secure, and user-friendly. In the third phase, Google incrementally iterated upon Manifest with Manifest V2 to corner the entire browser market. In the fourth phase, Google did very little—it intentionally lay dormant for several years to allow other vendors to concede control of their respective standards and adopt Manifest V2. Finally, in the fifth phase, Google exploited vendors’ investment in Manifest to introduce changes which cater to its main business of online advertisement and data collection.

With this birds-eye view of the past several decades, we can see that Google acquired most of its power over browser extension standards during the (3) *Iterate* and (4) *Await* phases of its strategy. Had a standard-setting agency such as the W3C begun drafting Manifest V2 as a democratic web standard during this time, perhaps this exploitative takeover could have been prevented. Indeed, in mid-2021, browser extension developers embraced the phrase “better late than never” by creating the WebExtensions Community Group within the W3C, with members including representatives from Google, Microsoft, Mozilla, and Apple.⁹⁷ While the group has some promise, it remains to be seen whether it will be effective given its blatant internal power imbalance and dormancy in the past several years.⁹⁸ In any case, it is my hope that an understanding of Google’s strategy will help future developers identify and terminate extension standard takeovers in emerging technology sectors before it is too late.

97. *Forming the WebExtensions Community Group* | *WebExtensions Community Group* [in en-US], accessed December 7, 2022, <https://www.w3.org/community/webextensions/2021/06/04/forming-the-wecg/>.

98. Alexis Hancock, *Manifest V3: Open Web Politics in Sheep’s Clothing* [in en], November 2021, accessed December 7, 2022, <https://www.eff.org/deeplinks/2021/11/manifest-v3-open-web-politics-sheeps-clothing>.