MV3: Providing security, privacy, and performance at the expense of innovation

Azhaguramyaa V R[1], Janet J[2], Mohammed Farhan P[3], Alen Morres[3] and T M Keerthi[3]

[1]Assistant Professor, [2]Professor, [3]PG Sholars

Department of Computer Science and Engineering,

Sri Krishna College of Engineering and Technology, Coimbatore.

{19epci019@skcet.ac.in}

Abstract. Browser extensions in modern browsers have evolved to a point where they not only extend browser capabilities but redefine the way we approach browser customization. Nearly all modern day browsers follow manifest version-2 specification for developing and implementing the extensions in the browsers. But in 2018 Google proposed to make some foundational changes to the chrome's extension framework (MV3). Migration of extensions from MV2 to MV3 has already started in the 3rd quarter of 2021. In this paper, we will analyze and discuss the key changes made by MV3, Also criticize certain changes that appear to be detrimental to the Browser Extensibility Ecosystem's innovative aspect.

Keywords: Chrome extensions, mv3, security, performance

1 Introduction

Browser extension is a tiny piece of software/code that enhances the functioning or capabilities of a web browser. A browser extension, sometimes known as plug-ins or add-ons, may utilize the same application programming interfaces (APIs) as JavaScript on a web page, but it can also access its own set of APIs, enabling it to do more. The exact distinction between plug-ins, add-ons, and extensions is difficult to establish since terminology varies depending on the web browser you're talking about

and the users' personal viewpoint [31]. Web browsers have become the day-to-day medium to access the internet. Modern web browsers have advanced to the point where they may be considered their own operating system. Browsers' pervasiveness has solidified their importance in consumer productivity and self-efficacy. Due to its empathetic evolution from a simple client application that displayed sheer pages of static content into a labyrinth of networks tasked with managing and facilitating an intuitive experience for the user. At the initial stages of browsers, it was a way to access the internet. The growing popularity of browsers meant increasingly diversified types of users. On that note, browsers, like operating systems, provide a fair amount of user customization, and browser extensions are one of them. Browser extensions extend browser functionality by interposing on and interacting with browser-level events and data to assist and fulfill the diverse demands of a large user base [3]. Some extensions are straightforward and just make minor modifications to the look of Web Pages. Other extensions offer more advanced functioning capabilities. The scope of what a browser extension can accomplish is determined by the browser. Browser extensions had enormous power and access to the inner workings of a web browser in their early days. However, this opened up a new attack vector for bad actors, leading to abuse of the power granted. Nowadays, coding methodologies have evolved, and access controls have been implemented to prevent bad actors from getting access and executing malware. Most browsers have their own marketplace for adding extensions, such as Chrome's "Web Store", Mozilla "Add-ons," Opera "Add-ons," and Microsoft Edge "Add-ons." In brief, we will look at the history and process of extension creation, analyze the development methodologies of browser extensions such as User scripts and Web Extension APIs, as well as how these APIs paved the way for standardization in browser extension development and offered cross-platform compatibility. Following that, discuss how the transition from manifest version 2 to version 3 affects extension development. What are the advantages in terms of user privacy, performance and security, and how does this move assist developers in creating and deploying new cross-platform compatible extensions.

2 Literature Survey

2.1 History of Internet Browser Distribution

As we all know, a website browser, or browser for short, is your gateway to the World Wide Web. Browsers are used to search and display web pages. Browsers were the most crucial and valuable contributors to the creation of the internet as we know it today. We would still be living in a primitive world if they did not exist. It would only seem fair and right to provide an essential perspective on the world of browsers from its origin with the launch of Netscape Navigator to the most popular and well-known

browser today, the Chrome Browser. We'll go over the list of browser distributions chronologically [15].

NCSA Mosaic (1993). The National Center for Supercomputing Applications (NCSA) was a University of Illinois research group that created the revolutionary web browser Mosaic in 1993. Mosaic played a critical role in making the web more user-friendly. The term Mosaic related to its capacity to support several internet protocols, such as NNTP, Gopher, and FTP, in addition to the fundamental Hypertext Transfer Protocol (HTTP) on which the web was constructed. It was developed by a team led by Marc Andreessen and Eric Bina.

Netscape Navigator (1994). Netscape Communications Corporation was created by Andreessen and Bina after they left the NCSA to launch the Mosaic Communications Corporation, which they co-founded with other former colleagues. Netscape was the name they gave their browser, and it was by far the most powerful and user-friendly browser available at the time. Only four months after its release, it had the most used browser at that time. The firm was renamed Netscape Communications Corporation to avoid naming conflicts with its prior employer. Notably, Netscape went open-source in 1998, paving the way for Mozilla to subsequently develop their own browser, Firefox.

Opera Browser (1994). In a similar vein, Opera's history also began in 1994, when the browser market was still in its infancy. Netscape Navigator was the market leader at the time, and it was around this time that Telenor, Norway's state-owned telecoms firm, decided to build software to surf the intranet. The project spun off into its own business, Opera Software ASA, in 1995, with the first publicly accessible version published in 1996. Jon Stephenson von Tetzchner and Geir Ivarsoy were the brains behind the project.

Internet Explorer (1995). The "Browser Wars" got underway with the release of Internet Explorer in 1995. Although Microsoft Internet Explorer (MSIE) or Internet Explorer (IE) came standard with Windows 95, users had to explicitly install it. It had less functionality and features when it was first released. Later, further features were introduced to succeeding versions of Internet Explorer. By 2002, Internet Explorer had grown to be a behemoth, with well over 90% of internet users using it. Thus began Internet Explorer's reign [27]. The Internet Explorer was scrapped and replaced with the Microsoft Edge, Which was first built in-house and was released in 2015. Edge was initially developed in-house and was based on HTML and Chakra engines. That, too, was decommissioned. Edge was redesigned in 2019 on top of the chromium google’s open-source Bowser project, which used the Blink and V8 engines [4].

Mozilla Firefox (1998). Firefox is often regarded as the successor to Netscape Navigator, given the Mozilla community was created by Netscape previous to their acquisition by AOL in 1998. With the release of the Netscape browser suite source

code in 1998 [29], the Mozilla project was started. Its objective was to embrace the creative force of hundreds of internet developers and create new possibilities for growth in the internet industry. Mozilla 1.0, the first major version, was launched in 2002 after many months of development. This version brought several enhancements to the browser, email client, and other apps in the suite, but it was not widely used. Firefox version 1.0 was released on November 9, 2004, within about nine months claiming 60 million downloads, challenging Internet Explorer's monopoly.

Chrome Browser (2008). The next significant shift in browsers occurred when Google launched their own web-browser Chrome. The Chrome browser was released at an optimal time when developers were struggling with Internet Explorer. It was the most advanced browser bundled with rich features and a highly dynamic web user interface, complying to the most modern Web standards, when it was released in 2008. Chrome was built using its counterpart Chromium, Google's open-source browser. It was labeled "Fresh Take on Browser" [30] at the time of its release. It included features such as sandboxing and a fresh scripting engine. Apple's Web Kit and Mozilla Firefox components were used in the development. Soon, Chrome became the browser industry's leader and the creation of new standards.

2.2 History of Browser Extensibility Ecosystem

Plug-ins (1995). Browser plug-ins has been around since 1995, the early web only allowed for static material to be displayed, which were images with inline texts. They were called plug-ins because they inserted a piece of code called “Native Executables” into the browser, which helped to render dynamic content in the browser. Native Executables have long been a source of insecure and unreliable browsing experiences. Despite the security flaws, the move from outdated web development technologies to newer technologies took some time. As a result, end users were skeptical about the legitimacy of the “plug-ins” they were installing on their devices.

Bookmarklet (1998). "Bookmarklet" is the next modest improvement in the history of browser extensions, Bookmarklets are saved as a bookmark but contains JavaScript code that adds additional capabilities to the browser. Bookmark applets, favlets, and JavaScript bookmarks are other names for bookmarklets. It was thought to be the pinnacle of user customization because anyone, with a little understanding of JavaScript, could code them to their liking and needs. Bookmarklets met a snag in 2004 with the adoption of the W3C's Content Security Policy (CSP) specification [33]. CSP was created to safeguard websites from dangerous user material and third-party code, by letting site owners to restrict code sources and network requests.

According to the CSP Specification, to maintain extensibility, Bookmarklets and other extensibility features should be immune from CSP limitations. Bookmarklets in

Firefox are conditional on following the site's CSP [32], which prevents them from running on pages. Chrome, on the other hand, does not apply CSP to bookmarklets. By default, Chrome shows bookmarks on the New Tabs page, users may configure the bookmarks bar if they want to run bookmarklets on other pages.

Browser Extension (1999). The first major browser to support browser extensions was Internet Explorer 4.0. [34]. "Explorer Bars" were the first Internet Explorer extensions. Browser extensions led to users being able to customize the browser's interface, change the interface of the web page and execute JavaScript within the context of the webpage. Extensions, on the other hand, didn't become the powerful tools that they are today until much later. In 2004, Firefox was the first browser platform to support extensions and to have a marketplace called "Mozilla Update ", Opera followed in 2009, Google Chrome and Safari in 2010.

3 Methodologies for Developing Extensions

3.1 User Style Sheets

In 1994, The World Wide Web Consortium (W3C) was instituted by Tim Berners-Lee to develop cross-browser standards, which helped to resolve the inconsistencies in rendering a site across browsers. The Cascading Style Sheets (CSS) specification from 1996 was one of these standards. CSS defines "stylesheets," which inform browsers how to style the content on a page. Opera 3.5 was the first browser to offer CSS, as well as user stylesheets, in 1998. Internet Explorer followed suit in version 4. However, user style sheets did not have much of an influence because they would only benefit a small number of consumers who were familiar with CSS or other web technologies. The bulk of consumers were not benefited by this. Also, consumer styling began to transition to style manager browser extensions. Jason Barnabe's Stylish add-on made its debut in 2005. Unlike user stylesheets, style managers offer additional control, sharing capabilities, and ease of use. In today's browser extension marketplace, the number of styling extensions has skyrocketed. However, it has recently been deprecated in some browsers. While many other browsers still continue to support User style sheets. [14]

3.2 Userscripts

Userscripts have a number of advantages over bookmarklets, including the ability to run on a page automatically, the lack of length limits, and the ability to run outside a site's CSP.

Userscripts are bits and pieces of JavaScript code that allows customization of one or a group of web pages to their own needs by uploading the scripts in the browser environment and running them locally at the client side. It is coded in JavaScript and offers the user extensive control over the mechanics of the web browser, allowing them to manipulate the display of web pages based on their preferences and needs. When compared to standard browser extensions, Userscripts are far more powerful. To list a few distinctions between userscripts and browser extensions, Browser extensions load whenever the user opens the browser, but userscripts load only when the specified website is visited. Userscripts are very light, website specific and require fewer system resources, allowing the tasks to be completed quickly and effectively. Userscripts were popular in the early days of browsers because it permitted the users to easily install and run them. However, the support for the Userscript directory has been removed from 2011 in Chrome [35]. Certain extensions have been designed to allow users to launch and execute scripts in order to circumvent this limitation. They are not as widespread as browser extensions, since downloading and installing scripts requires a higher level of operational expertise than downloading and installing a browser plug-in from a web-store. Currently, Mozilla Firefox enables the installation and execution of userscripts natively [5]. Other browsers need the users to install third-party extensions such as Grease monkey, Tamper monkey and Violent monkey in order to run userscripts.

3.3 Mozilla XUL

The user interface of Mozilla Firefox was created using the XUL programming language. The XML User Interface Language (XUL) is a markup language that allows you to create sophisticated dynamic user interfaces. It is included with the Mozilla browser and associated applications. It enables them to modify key elements of the user interface.

However, Firefox deprecated XUL for extensions in 2017 [12], citing vulnerabilities and causing the browser's UX to be unstable when many XUL extensions were loaded at the same time.

3.4 Chrome WebExtension API's

Prior to the release of Google's Chrome browser, each browser had its own APIs and manifest syntax for creating extensions. This resulted in a fairly difficult procedure for developers who wanted their extensions to be available in several web-stores. Chrome documented the extension's development in Manifest Version 1(MV1) as early as 2009 [6]. The manifest file is a configuration file that instructs the browser on how to display information and injects scripts. The term "Manifest" refers to both technological interconnection points and regulations. A browser enforces when reviewing extensions for its Web Store. Chrome Browser was still relatively new at the time, when Google introduced its own APIs for extension development, that did not make much of an impact. Google started their web store in 2010. In mid-2012, manifest version 1 was announced to be deprecated [7] and Manifest Version 2 became the new standard.

3.5 MV2

When Manifest Version 2 (MV2) was released, it included a list of advanced APIs [9] as well as certain security enhancements. However, it was the proper move forward that allowed the internet to grow with extensions. Because of their widespread usage, chrome extensions have become a popular attack vector for malicious actors to exploit due to the access they provide [8]. Chrome developers quickly saw a pattern of malicious extensions exploiting and gaining unauthorized access to a user's browser window via these attack vectors. Soon after, they took action by discontinuing quiet extensions installs, prohibiting "Multi-purpose Extensions" and discontinuing inline installation from extensions [10]. In comparison to other browsers, using cutting-edge Web technologies, as opposed to Internet Explorer and Mozilla Firefox at the time and utilizing the most recent HTML, CSS, and JavaScript standards. Creating extensions was made simple with MV2. As a result, developers inundated the web stores with extensions for practically any online operation a person could think of, and users welcomed them because they improved user experience and convenience of use. Because of its popularity, other browsers quickly followed suit and began implementing the Chrome WebExtensions API introduced in MV2. For reasons such as cross-platform compatibility, this allows developers to easily port extensions to different browser distributions. Opera was the first to switch from their traditional presto engine to chromium in 2013 [11]. For similar reasons, Mozilla Firefox deprecated their XUL programming language used to develop user extensions to Chrome's De-facto WebExtensions API in 2017 [12], followed by Microsoft Edge [4] and Apple's Safari [13]. With practically all browsers supporting Chrome's web API extensions, security concerns about extensions were once again highlighted. However, there are security flaws in MV2 which permits arbitrary code execution (from the server), also known as remote code execution. To

address all issues at once, Google has published a timeline indicating the support and subsequent deprecation of MV2 [22].

3.6 MV3

Google proposed the new version of manifest (MV3) in 2018 with better and safer Extension APIs [23], with intentions to adopt it and deprecate MV2. As earlier said, nearly all online browsers currently employ some form of Chrome WebExtension API, so change was inevitable. With so many variables and stakes at play, a single extension vulnerability might cause a chain reaction across the internet. Currently, the entire ecosystem of browser extensions is reliant on Chrome's API. To address the matter front on, some fundamental modifications were made to MV3. The MV3 extension support change went active in Chrome 88 [24]. Key changes in terms of user security and privacy are listed below.

- Remotely hosted configuration files instead of code.
- Service workers instead of a background page.
- Network request modification: declarativeNetRequest (DNR) instead of webRequest API.

The first significant enhancement is the prohibition of code execution from remotely hosted servers. Previously utilized as an attack vector for data exfiltration, this will undoubtedly improve user security.

The second is to use service workers as a new execution environment for the background process. Service workers aren't exactly a new feature, as they've already been used to assist websites push and cache messages. They are described as a "give and take" model by Chrome. In terms of security, they are designed to prevent extensions from running code in the background and to load only when the user invokes them. Service workers are in a way similar to userscripts, which can run on specific web pages that are invoked by the user. Because background threads are terminated while not in use, it can theoretically use less memory and make the browser faster. However, because Chrome extensions have gotten considerably more complicated and provide more functionality than basic userscripts, It has yet to be determined whether service workers function smoothly in complex browser extensions. This may result in greater user control over extension access, but it may also result in feature loss with particular extensions that require users to run scripts in the background at all times. Service workers have limited access to Document Object Model (DOM), they also lack support for JavaScript modules In the manifest, only one script can be used.

The third is Web Request API and Declarative Net Request API, which are briefly described below [17].

Table 1. Comparison between Web Request API and Declarative Net request API

Web Request API	Declarative Net Request API
Chrome transmits all the data in a network request to the extension that is listening for it when you use WebRequest. The extension can then assess the request and inform Chrome on what to do with it: allow it, block it, or transmit it with some changes.	Chrome does not need to communicate all details about a request to the listening extension when using Declarative Net Request. Instead, the extension registers rules with Chrome that instruct the browser what to do if particular sorts of requests are encountered.

Chrome used to pass over all network request data to the extensions listening via the Web Request API, which reads and alters everything a user does on the web. This is a lot of power given to developers, and it is utilized responsibly, Web Request API has been reported for approximately 42% exploitation since January 2018 [37]. Therefore, Declarative Net Request API was the way to go, requiring extensions to describe rules that would be matched instead of initiating any kind of network request at all. Because network requests are not sent out frequently to extensions, this can improve extension performance in low-resource contexts.

Manifest V2 required the extensions to declare its intent in the "permissions" field of the manifest, allowing the extension to exercise the declared permissions persistently. The fact that invoking these critical permissions is not moderated and allowed to run makes it a security risk, as this can be taken advantage of by malicious extensions. As a result, changes were made in MV3 to the way the extensions acquired permissions. In-order to make the operation of the extensions more secure and give the user more control, the host permissions specifications are declared under the "host_permission" key, which will require approval from the user during runtime. "permissions" and "optional_permissions" are reserved for API permissions which will be granted when it detects signals of user Intent, such as an action button click, keyboard shortcut or the context menu entry. Rather than forcing the user to provide all permissions, Chrome will grant the extension temporary permissions to visit the site during runtime and then withdraw the permissions, preventing the extension from having Persistent Host Access. Additionally, extension developers are encouraged to create extensions that require minimal rights during installation and then request additional permissions upon invocation. If extensions are not constructed with minimal permissions, unexpected behavior will occur when the user declines access.

Developers had to use different libraries to “promisify” Chrome’s extension APIs in MV2. But in MV3, chrome’s extensions API can now return promises [36]. Instead of passing callbacks into a function, a promise is a returned object to which you may attach callbacks, which will result in a more streamlined error handling and easier-to-manage code. Not all methods in extensions API support promises yet, although

chrome is actively trying to provide support for more methods in MV3. Multiple asynchronous functions are easily managed while using promises instead of callbacks. In addition, MV3 introduces a number of other modest feature modifications. A slew of new APIs were released. Action API consolidation, which combines browser and page actions into a single action. MV3 also changes how permissions are granted to chrome extensions. Furthermore, announcements of a new Scripting API that can register and unregister content scripts at runtime were made [16].

Mozilla, Microsoft, and Apple have all agreed to these new improvements. It may be expected to see some sort of MV3 implementation with platform-specific tweaks [20]. However, browsers such as Brave, Vivaldi, and Opera have opposed the move and stated that changes will not be implemented in respective platforms, despite the fact that they are built on chromium and share the same code base [21].

4 Analysis

Manifest V3 (MV3) is a significant step forward for the extensions platform. While retaining and expanding its foundation of capability and "webbiness", MV3 prioritizes privacy, security, and performance. The developer and end-user experience will ameliorate with the growth of extensions platform.

The following advantages have been broken in the categories below:

Privacy. To offer means for extensions to function properly without needing to access user data. Improve user permission control by notifying users about the activities of extensions and allowing them to give rights in real time and context.

Performance. To ensure that extensions don't have any runtime speed concerns that interfere in the browsing experience. Chrome runs smoothly even with a significant number of extensions installed.

Security. To make it more difficult for extensions to access resources outside the extension context by implementing stronger protocols.

Webbiness. To embrace the web platform's organizational culture to reduce developer engagement constraints and gain advantages as the web platform grows [38].

Capability. To maintain the platform's capability, power, and feature-richness so that extensions may continue to develop and provide users with ever-greater value.

The Changes to the Manifest gives users more visibility and control over how extensions access their data and other resources, allowing them to better manage how extensions use their data and other resources.

5 Criticism

Unfortunately, the makers of Privacy Filter and Content Blocking (AKA Ad-Blockers) extensions were outraged by these new changes to the manifest. To keep criticism to a minimum many references have been given to articles, papers and blog posts written. According to the Privacy filter developers, these changes have a significant impact on the operation of their filtering engines, rendering their product obsolete [18][39][40][41][42][43][44]. Apart from privacy filter developers, browser extension developers in general have expressed concern that this change may introduce unnecessary complexity [19]. Despite these reservations, there are some far more significant challenges that must be addressed at the core most level. The discussion has turned from limiting developers to whether this change has any effect on user privacy. [25] It is acknowledged that MV3 does a poor job of preventing extensions from exfiltrating data (browser history) and sharing it with objectionable data aggregators. Since, no modifications have been made to the observational APIs [26], which continue to allow extensions to track and monitor users' online activity. Furthermore, there are no changes to the way content scripts work. Content scripts are responsible for enabling extensions to provide useful functionality. They can also be used to extract user browsing data in another method. With these considerations, it is debatable if Google's privacy claims are valid. Regarding user performance arguments, the research paper [1] discusses the benefits of privacy-based browser extensions, known to actually increase user experience and browser performance rather than slowing it down. Finally, the user's security arguments are shown. (DoubleX) [28][45] is a tool developed by researchers from the CISPA Helmholtz Center for Information Security in Germany, that aids in discovering insecure data flows in browser extensions. According to their paper [2], harmful extensions account for a relatively small part of the extensions that pose security and privacy problems. Meanwhile, these seemingly harmless extensions with vulnerable code are being used by malicious extensions and web pages to execute scripts, exfiltrate data, trigger downloads, and do other things. "Harmless extensions" are regarded as more worrisome because they cannot be discovered and remain undetected while inflicting damage.

6 Conclusion

It can be summarized that, despite the changes intended to provide security and performance while protecting privacy, By expanding the approach in secure coding

methodologies and a newer permissions model have caused a great deal of worries amongst extension developers. The changes may hinder rather than accelerate the expansion of the innovation in the extension community, Due to the added level of dexterity required in developing an extension with MV3. The direct ramifications of MV3 in the browser extensibility ecosystem have yet to be foreseen. It is unclear which path Userscripts, User style sheets, and Chromium-based browsers will be compelled to take.

References

1. Borgolte, K., & Feamster, N. *Understanding the Performance Costs And Benefits Of Privacy-focused Browser Extensions,* (2021)

2. Fass, A., Somé, D. F., Backes, M., & Stock, B., *DoubleX: Statically Detecting Vulnerable Data Flows In Browser Extensions At Scale,* (2021)

3. Bandhakavi, S., T. King, S., M., & Winslett, M, *VEX: Vetting Browser Extensions For Security Vulnerabilities* (2010)

4. New Microsoft Edge To Replace Microsoft Edge Legacy With April's Windows 10 Update Tuesday Release - Microsoft Tech Community, (2021) https://techcommunity.microsoft.com/t5/microsoft-365-blog/new-microsoft-edge-to-replace-microsoft-edge-legacy-with-april-s/ba-p/2114224

5. UserScripts - Mozilla | MDN userScripts - Mozilla | MDN. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/userScripts

6. Developers, Google. "Google Chrome Extensions: How To Build an Extension." *YouTube,* Www.youtube.com, (2009) https://www.youtube.com/watch?v=e3McMaHvlBY

7. Manifest Version - Chrome Developers. (n.d.). Chrome Developers. https://developer.chrome.com/docs/extensions/mv2/manifestVersion/#manifest-v1-support-schedule

8. Birsan, A., *Owned By Chrome - Security Issues With Browser Extensions - Infosec Resources.* Infosec Resources., (2021) https://resources.infosecinstitute.com/topic/owned-by-chrome-extensions/

9. About Manifest V2 - Chrome Developers. (n.d.). Chrome Developers. https://developer.chrome.com/docs/extensions/mv2/

10. Shah, H., *From 0 To 70% Market Share: How Google Chrome Ate the Internet.* Nira, (2019) https://nira.com/chrome-history/

11. Anthony, S., *Opera Admits Defeat, Switches To Google's Chromium - ExtremeTech.,* (2013) https://www.extremetech.com/computing/148312-opera-drops-presto-switch-to-google-and-apples-webkit-rendering-engine

12. Vaughan-Nichols, S. J., *Mozilla Drops XUL, Changes Firefox APIs; Developers Unhappy | ZDNet.* (2015) https://www.zdnet.com/article/mozilla-changes-firefox-apis-developers-unhappy/

13. Inc., A., *Easily Create Web Extensions for Safari - News - Apple Developer.,* (2020) https://developer.apple.com/news/?id=kuswih5l

14. Meiert, J. O., *User Agent Style Sheets: Basics And Samples · Jens Oliver Meiert,* (2007) https://meiert.com/en/blog/user-agent-style-sheets/

15. Schiller, T., *A Brief History Of Browser Extensibility | By Todd Schiller | Brick By Brick | Medium,* (2021). https://medium.com/brick-by-brick/a-brief-history-of-browser-extensibility-bcfeb4181c9a

16. Overview Of Manifest V3 - Chrome Developers. (n.d.). Chrome Developers. https://developer.chrome.com/docs/extensions/mv3/intro/mv3-overview/#network-request-modification

17. Siddiqui, A., *Google's Manifest V3 Will Change How Ad Blocking Chrome Extensions Work: Is It To Cripple Them, Or Is It for Security?*.xda-developers, (2019) https://www.xda-developers.com/google-chrome-manifest-v3-ad-blocker-extension-api/

18. Ghostery.com. (2021) https://www.ghostery.com/blog/manifest-v3-the-ghostery-perspective

19. Cimpanu, C., *Chrome API Update Will Kill a Bunch Of Other Extensions, Not Just Ad Blockers | ZDNet,* (2019) https://www.zdnet.com/article/chrome-api-update-will-kill-a-bunch-of-other-extensions-not-just-ad-blockers/

20. Espósito, F., *Apple Teams Up With Google, Mozilla, And Microsoft To Improve Browser Extensions - 9to5Mac,* (2021) https://9to5mac.com/2021/06/04/apple-teams-up-with-google-mozilla-and-microsoft-to-improve-universal-web-browser-extensions/

21. Cimpanu, C., *Opera, Brave, Vivaldi To Ignore Chrome's Anti-ad-blocker Changes, Despite Shared Codebase | ZDNet.* (2019) https://www.zdnet.com/article/opera-brave-vivaldi-to-ignore-chromes-anti-ad-blocker-changes-despite-shared-codebase/

22. Perrigo, M., *Chrome Extensions Will Be Much Safer By 2023 When Google Plans To Fully Snuff Out MV2.* Chrome Unboxed - The Latest Chrome OS News, (2021) https://chromeunboxed.com/chrome-extensions-manifest-v2-timeline.

23. Perrigo, M,. *Chrome 88 Introduces the Controversial Manifest V3 Which Seeks To Fix the 'extensions Problem'.* Chrome Unboxed - The Latest Chrome OS News, (2021) https://chromeunboxed.com/chrome-88-manifest-v3-controversy-ad-blockers-ublock.

24. Siddiqui, A., *Manifest V3 Changes for Browser Extensions Will Go Live In Google Chrome 88.* xda-developers, (2020) https://www.xda-developers.com/manifest-v3-changes-browser-extensions-go-live-google-chrome-88/

25. Miagkov, A., Gillula, J., & Cyphers, B., *Google's Plans for Chrome Extensions Won't Really Help Security | Electronic Frontier Foundation,* (2019)https://www.eff.org/deeplinks/2019/07/googles-plans-chrome-extensions-wont-really-help-security

26. Cronin, D, *Iterating On Manifest V3.* Iterating on Manifest V3, (2019)https://groups.google.com/a/chromium.org/g/chromiumextensions/c/WcZ42Iqon_M/m/lT-Y5aZiAwAJ

27. Broersma, M., *Netscape Market Share Hits New Low | ZDNet.* ZDNet, (2002) https://www.zdnet.com/article/netscape-market-share-hits-new-low/

28. *GitHub - Aurore54F/DoubleX: Statically Detecting Vulnerable Data Flows In Browser Extensions At Scale.* (2021). GitHub. https://github.com/Aurore54F/DoubleX

29. Press Release. (n.d.). Press Release. https://web.archive.org/web/20021001071727/wp.netscape.com/newsref/pr/newsrelease558.html

30. Pichai, S., & Upson, L., *A Fresh Take On the Browser.* Official Google Blog, (2008) https://googleblog.blogspot.com/2008/09/fresh-take-on-browser.html

31. Firefox - Exact Difference Between Add-ons, Plugins And Extensions - Stack Overflow, (2015) https://stackoverflow.com/questions/33462500/exact-difference-between-add-ons-plugins-and-extensions

32. Genestoux, J. *866522 - Bookmarklets Affected By CSP,* (2013) https://bugzilla.mozilla.org/show_bug.cgi?id=866522

33. Org, W., *Content Security Policy Level 3.* Content Security Policy Level 3, (2021) https://www.w3.org/TR/CSP3/#extensions

34. *Create Custom Explorer Bars, Tool Bands, And Desk Bands - Win32 Apps.* Microsoft Docs., (2021) https://docs.microsoft.com/en-us/windows/win32/shell/band-objects

35. Google Git chromium / chromium / src / (2021) https://chromium.googlesource.com/chromium/src/+/4c14ce487f93032da0ccc0d93ef42dd31303e0e6

36. Developers, G., *Using Promises - Chrome Developers.* Chrome Developers, (2021) https://developer.chrome.com/docs/extensions/mv3/promises/

37. Kan, M. (2019, June 12). *Google: We're Not Killing Ad Blocking Chrome Extensions | PCMag,* (2019)https://www.pcmag.com/news/google-were-not-killing-ad-blocking-chrome-extensions

38. Developers, G., *Extensions Platform Vision - Chrome Developers, (*2020) https://developer.chrome.com/docs/extensions/mv3/intro/platform-vision/

39. Meshkov, A., Objects In Manifest V3 Are Closer Than They Appear — And It's Not Good News, (2021) https://adguard.com/en/blog/manifestv3-timeline.html

40. Claburn, T. (n.d.). *Legacy Chrome Extensions To Stop Working From January 2023.* https://www.theregister.com/2021/09/27/google_chrome_manifest_v2_extensions/

41. Barnett, D., *Chrome Users Beware: Manifest V3 Is Deceitful And Threatening,* Electronic Frontier Foundation, (2021) https://www.eff.org/deeplinks/2021/12/chrome-users-beware-manifest-v3-deceitful-and-threatening

42. Hancock, A, *Manifest V3: Open Web Politics In Sheep's Clothing ,* Electronic Frontier Foundation, (2021) https://www.eff.org/deeplinks/2021/11/manifest-v3-open-web-politics-sheeps-clothing

43. Jennings, R. *Google Nukes Ad-Blockers—Manifest V3 Is Coming.* Security Boulevard, (2021) https://securityboulevard.com/2021/12/google-nukes-ad-blockers-manifest-v3-is-coming/

44. Polanco, T., *New Chrome Extension Rules Could Hobble Ad Blockers, (*2021) https://www.tomsguide.com/news/new-chrome-extension-rules-could-hobble-ad blockers

45. Claburn, T, New Tool Flags Up Benign-but-exploitable Chrome Extensions. New tool flags up benign-but-exploitable Chrome extensions, (2021) https://www.theregister.com/2021/11/11/chrome_extension_analyzer/