Work-in-Progress: Manifest V3 Unveiled: Navigating the New Era of Browser Extensions

Nikolaos Pantelaios
North Carolina State University
npantel@ncsu.edu

Alexandros Kapravelos
North Carolina State University
akaprav@ncsu.edu

***Abstract*—Introduced over a decade ago, Chrome extensions now exceed 200,000 in number. In 2020, Google announced a shift in extension development with Manifest Version 3 (V3), aiming to replace the previous Version 2 (V2) by January 2023. This deadline was later extended to January 2025. The company's decision is grounded in enhancing three main pillars: privacy, security, and performance. This paper presents a comprehensive analysis of the Manifest V3 ecosystem. We start by investigating the adoption rate of V3, detailing the percentage of adoption from its announcement up until 2024. Our findings indicate, prior to the 2023 pause, less than 5% of all extensions had transitioned to V3, despite the looming deadline for the complete removal of V2, while currently nine out of ten new extensions are being uploaded in Manifest V3. Furthermore, we compare the security and privacy enhancements between V2 and V3 and we evaluate the improved security attributable to V3's safer APIs, examining how certain APIs, which were vulnerable or facilitated malicious behavior, have been deprecated or removed in V3. We dynamically execute 517 confirmed malicious extensions and we see a 87.8% removal of APIs related to malicious behavior due to the improvements of V3. We discover that only 154 (29.8%) of these extensions remain functional post-conversion. This analysis leads to the conclusion that V3 reduces the avenues for abuse of such APIs. However, despite the reduction in APIs associated with malicious activities, the new Manifest V3 protocol is not immune to such behavior. Our research demonstrates, through a proof of concept, the adaptability of malicious activities to V3. After the proof of concept changes are applied, we showcase 290 (56%) of the examined malicious extensions retain their capability to conduct harmful activities within the V3 framework. They can achieve this by incorporating web accessible resources, a method that facilitates the injection of third-party JavaScript code. Conclusively, this paper also pioneers by documenting the impact of user and community feedback in the transition from V2 to V3, analyzing the percentage of initial issues that have been resolved, and proposing future directions and mitigation strategies for the continued evolution of the browser extension ecosystem.**

I. INTRODUCTION

Browser extensions have become an integral component of today's web browsing experience, offering a range of functionalities that enhance user productivity and security. The typical browser user often has multiple extensions installed, spanning various categories such as ad blockers like uBlock Origin [36], password managers such as LastPass [26], and productivity tools like BlockSite [9]. These extensions are of particular interest to the web security community due to their access to high-privilege APIs. These APIs enable extensions to modify web pages and bypass the browser's Same Origin Policy (SOP), among other capabilities. Consequently, a persistent challenge for webstore platforms is distinguishing between benign and malicious extensions and effectively mitigating the latter before they impact users.

To tackle the existing problem of extensions violating user's privacy and browser security, Google announced in 2020 changes in extensions design, reflected by upgrading Manifest version 2 (Manifest V2 or just V2) to Manifest version 3 (Manifest V3 or just V3). Google's transition from Manifest V2 to V3, initially planned for early 2023 but delayed to address developer concerns, aimed to enhance Chrome's extension ecosystem's security, privacy, and performance. V3 faced criticism for potentially impacting ad blockers and other extensions, leading to a more nuanced rollout. In contrast, Firefox implemented V3 differently, focusing on preserving ad-blocker functionality. As of late 2023 and early 2024, Google continued refining V3 based on community feedback, aiming to balance security improvements with the functionality desired by extension developers and users.

The transition to V3 has sparked diverse reactions, particularly among developers. Key changes in V3 include the discontinuation of third-party code inclusions and the replacement of certain APIs, which, while aimed at enhancing security, have raised concerns about potential compatibility issues and limitations in extension functionality. One of the most notable changes is the shift from the *webRequest* API to *declarativeNetRequest*, which, despite aiming for improved performance and privacy, has introduced complexities and bugs, potentially impacting the security of V3. Additionally, the new protocol limits extension collaboration, posing challenges for privacy-related extensions which now face restrictions in filter categories and request rates. These changes, while designed to streamline extension capabilities, have inadvertently limited their functionality and interoperability. Furthermore, the need for privacy-related extensions to maintain dynamic blocklists within the extension code has introduced additional overhead and frequent updates, possibly affecting browser performance and the web developer experience. V3's categorization encompasses remote code inclusion, API changes, content security policy (CSP) rules, and the introduction of service workers. Notably, the prohibition of third-party JS code inclusion and the alterations in API usage, like the restricted usage of *xhr*

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2024
1 March 2024, San Diego, CA, USA
ISBN 979-8-9894372-2-1
https://dx.doi.org/10.14722/madweb.2024.23080
www.ndss-symposium.org

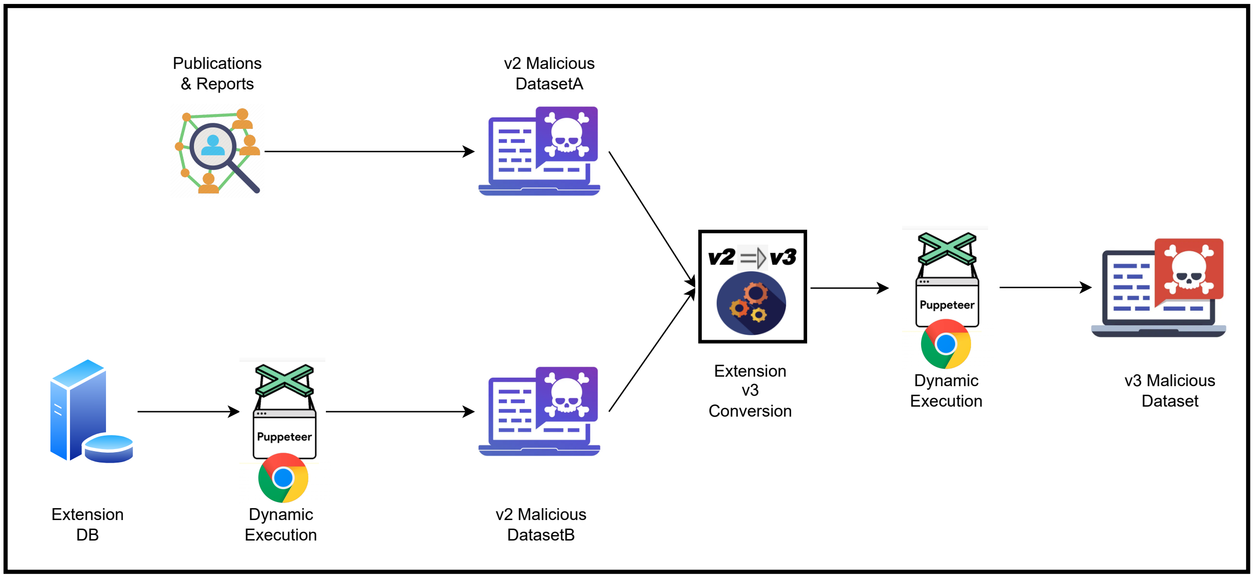

Fig. 1. A full description of our architecture including the data collection process, the conversion to Manifest V3 and the dynamic analysis stage.

and *executeScript*, signify a considerable shift in extension development. The stringent CSP rules in V3, particularly the deprecation of the *unsafe-eval* rule, further dictate the security landscape, restricting arbitrary code execution. Lastly, replacing background scripts with service workers aims at enhancing browser performance, though it introduces complexities in extension behavior and performance dynamics.

In our research, we extensively analyze the Manifest V3 ecosystem, focusing on the adoption rate and the security and privacy enhancements over V2. Our study involves converting V2 malicious extensions to V3 and dynamically analyzing them. Notably, we find that V3's implementation results in the removal of 87.8% of APIs related to vulnerable or malicious behavior, demonstrating a significant increase in security. However, our dynamic testing reveals that some extensions still manage to exhibit malicious behavior in V3, highlighting the need for ongoing vigilance in the extension ecosystem despite the considerable security improvements.

Our study compiles a total of 517 malicious V2 extensions from dynamic analysis and prior research. We then adapt these extensions to V3, following rule-based conversions outlined in official documentation. Our dynamic testing reveals that 154 (29.8%) of these extensions remain functional post-conversion. Moreover, 290 (56%) extensions can function in V3 with a *web_accessible_resources* declaration in the manifest file. The full extent of our architecture can be found in Figure 1 and we open source our dataset in Github [1].

Our paper makes the following contributions:

- We showcase the improvement of V3 against V2, with 87.8% of the APIs related to malicious or vulnerable JavaScript code not being available in V3 due to deprecation or replacement of those APIs
- We present a proof of concept to include 3rd-party resources in extensions developed in Manifest V3 to study malicious code that can still exist in the new ecosystem and propose mitigations
- We present the first study on the evolution and the adoption rate of the new Manifest V3 extension design over the past four years it was announced
- We open-source our historical dataset with the malicious extensions in both the Manifest V2 and Manifest V3 versions for future comparisons and to kickstart future research on this domain

II. BACKGROUND

A. Manifest V3 Community Collaboration

Google's development of Manifest V3 focuses on enhancing the security, privacy, and performance of browser extensions, with promises of improvements acknowledged in their statement and supported by sources like Ghostery [4], [1]. Google's iterative improvements to V3, such as the transition to the *declarativeNetRequest* API and adjustments to cosmetic rules filters, demonstrate their commitment to refining the extension platform while addressing developer concerns. Efforts to maintain and enhance extension capabilities, alongside a responsive strategy to community feedback, show Google's dedication to evolving the extension ecosystem to meet security and privacy standards effectively.

B. Categorizing Manifest V3 Changes

Code Inclusion A key change in Manifest V3 is the prohibition of third-party JavaScript (JS) code inclusion, targeting extensions that inject third-party code. Only code bundled with the extension is permitted, with *web_accessible_resources* declarations allowing for limited third-party interactions.

API Changes Manifest V3 introduces API modifications like replacing *webRequest* with *declarativeNetRequest* and restricting *xhr* usage, enhancing security and performance. The

---

[1] Link to Github

Malicious category	Have source code (multi-labels)
Click scam	33
Ad replacement	112
User data analytics	356
Credentials stealing	3
Browser modification	111
Other (crypto, mining, gambling, phishing)	90
Total (unique)	517

TABLE I. MALICIOUS EXTENSIONS CATEGORIES BASED ON MANUAL LABELING. EACH MALICIOUS EXTENSION CAN HAVE MULTIPLE LABELS.

executeScript API moves to *scripting*, and several APIs are deprecated or replaced to improve security [2], [17], [5].

CSP Rules Manifest V3 tightens CSP rules, notably deprecating *unsafe-eval* and narrowing permissible CSP values to enhance security.

Service Workers The shift to *service workers* from *background scripts* aims to boost performance and align with browser architectures, emphasizing asynchronous, event-driven operations [3].

C. Extensions Structure

The source code of an extension is comprised of a combination of JavaScript (JS), HTML, CSS files, media files, and JSON files. The central configuration file, known as *manifest.json*, specifies the extension's permissions. This manifest file delineates all the JS files associated with the extension. In Manifest V2, these JS files are categorized as content scripts and background scripts. However, in Manifest V3, the background scripts have been replaced by service workers. The key distinction between these lies in the set of APIs accessible to each script and the context in which the script operates.

Following the specification of all the JS and media files used by the extension, it is submitted to the webstore for approval. Once accepted, the extension is assigned a unique 32-byte hash identifier, referred to hereafter as the 'id'.

D. Tools

Playwright [31] is a browser automation tool that operates in a secure and isolated environment. *Playwright* is capable of simulating a variety of browsers, including Chrome, Firefox, Opera, Safari, and Edge. While our analysis currently focuses solely on Chrome extensions, we utilize Playwright to facilitate potential future expansions to other browsers. We utilized *Playwright* for installing extensions while emulating browser behavior. Due to the high volume of requests generated during simulation, we employed a webpage record and replay tool named *Catapult* [13] to minimize redundant requests.

For testing all malicious extensions in Manifest V3, a conversion process from V2 is necessary. This conversion utilizes a blend of pre-existing tools and our own modifications to address specific cases not covered by standard tools. The foundation of our conversion process is the *extension manifest converter* tool [19], which translates both the manifest and JS files. Key manifest fields we convert include *host_permissions*, *content_security_policy*, *background.script*, and *sandbox*. Additionally, for APIs, we adapt *browser_action* to *action*, *tabs.executeScript* to *scripting*, and modify *tabs.insertCSS*.

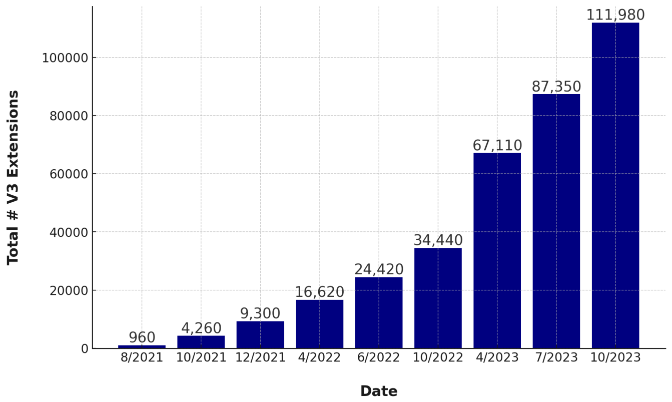

Fig. 2. Number of V3 extensions in our database over time

III. MALICIOUS EXTENSION DATASET

A. Extension Sources from Past Reports

To create our dataset, we source extensions from various origins, predominantly security companies and previous scientific studies. Our collection focuses on malicious extensions, categorized based on their reported activities. The main source of our dataset is a resource gathered from previous malicious extension reports from the past seven years [41]. This collection comprises hashes of extensions organized based on significant incidents within the malicious ecosystem. This collection of reports offers information on malicious extension packages and has been utilized in prior research [46].

These reports include extensions stealing user data [39], [8], acquiring Facebook information [40], capturing sensitive data [24], [34], spying on users [16], compromising credentials [38], [29], exploiting Chrome features [7], and bypassing authentication [22], [35]. Noteworthy in our collection is *The Great Suspender*, implicated in user data theft [27], [37], and extensions targeting accounts and passwords [6], [25], including those from foreign entities like North Korea [18].

B. Our Historical Dataset

Although the resource provides access to reports on malicious extensions, it lacks the source code. Therefore, we maintain a historical dataset of extension source codes. To compile this dataset, we download all extensions from the webstore that were updated in the last 24 hours, every day. Over six years, this method has enabled us to gather a comprehensive dataset that includes a total of 289,000 extensions. This dataset serves as a valuable tool for conducting detailed analyses, queries, and comparisons between various versions of extensions.

Using this historical dataset, we have been able to identify malicious versions for 517 IDs. Our dataset is the main source for identifying 85% of these malicious extensions. The remaining 15% are identified with the help of the *chrome-stats.com* tool [14], which assists in finding versions that our dataset does not cover.

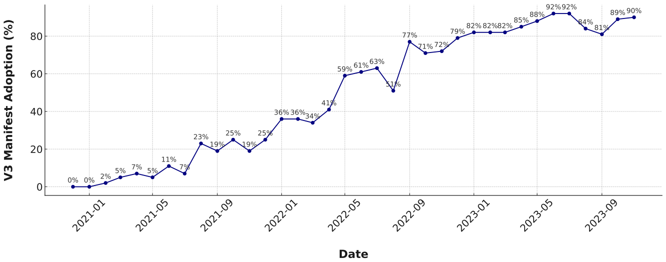

Fig. 3. Manifest V3 monthly update rate over the past four years to showcase how fast the adoption is happening.

IV. METHODOLOGY

A. *Architecture*

In Figure 1, we present the complete architecture and methodology used in this study. The process starts with the collection of malicious data in the Manifest V2 format. We then develop a converter to transition these extensions to the Manifest V3 compatible ecosystem. After the conversion, we run each extension through a dynamic analysis tool. This step ensures the conversion's success and checks if the default settings can monitor the malicious behavior. We run it on URLs the extension has access to run on to ensure functionality. Finally, if malicious behavior exists in the Manifest V3 ecosystem, we perform a manual verification.

B. *Gathering & Verifying the Malicious Datasets*

As outlined in the data analysis section (§ III-A), due to the scarcity of reports on malicious Manifest V3 extensions, our investigation centers on those developed in Manifest V2. Initially, we merge our two partial datasets to create the final dataset. To analyze these extensions, we utilize *Playwright* for automating a browser environment. This approach facilitates automated tests, with each test involving a predefined browser session featuring one of the extensions. During each test, the extensions are navigated through a series of predetermined *URLs* they are allowed to run on based on information from the *manifest.json* configuration file.

Following this process, we undertake the verification of the malicious behavior in all 517 extensions identified as suspicious. This verification process is twofold: we manually examine the code where possible, and in instances where the report lacks details about the malicious behavior, we dynamically execute and monitor the extension for any signs of potentially unwanted behavior. This comprehensive approach ensures a thorough assessment of each extension's activities and functions.

C. *Malicious Categories Labeling*

We categorize malicious extensions into six types to understand their behaviors and persistence in Manifest V3. These include click scams, ad replacement, user data analytics, credentials stealing, browser modification, and *other* activities like crypto theft and phishing. User data analytics is the most common, capturing 68% of labels, indicating extensive personal information collection. Browser modifications and ad replacements are also significant, each at 21%, while credentials stealing is less common with eight instances, reflecting its high impact and execution complexity. The distribution across categories is detailed in Table I, noting some extensions fall into multiple categories.

(OLD) Manifest V2	(NEW) Manifest V3
manifest_version: 2	manifest_version: 3
background_scripts	service_workers
chrome.browseraction	chrome.action
CSP policies (any)	CSP policies (self, none, localhost, 127.0.0.1)
chrome.extension.* [45]	chrome.runtime.*
chrome.tabs.* [45]	chrome.[VARIES].*

TABLE II. CATEGORIES OF MANIFEST V3 CHANGES OUR AUTOCONVERTER HANDLES SUCCESSFULLY.

Manual Verification: The manual verification process involves spending 15 minutes per extension to examine the code, culminating in approximately 130 hours of manual review.

D. *V3 Conversion*

In the process of adapting extensions from Manifest Version 2 to Version 3, our autoconverter, which is based on official documentation, focuses on crucial conversions as outlined in Table II. The conversion process updates the *manifest.json* file, which is essential for the functionality of the extension, as well as deprecated APIs in the source code of the extension. The *manifest.json* changes include updating the *manifest_version* to 3 to indicate compliance with the most recent specifications. Background scripts are replaced with *service_workers* to align with Manifest Version 3's emphasis on more efficient and secure background processes. Furthermore, the transition from *chrome.browseraction* to *chrome.action* aligns with the new Manifest standards, and Content Security Policy (CSP) policies are updated to values accepted by Manifest Version 3.

The converter also addresses the deprecation of certain APIs by transitioning from the deprecated *chrome.extension.** to *chrome.runtime.**, for the subset of APIs affected and reported in the official website [45]. Nevertheless, our converter encounters challenges with extensions that rely heavily on background scripts for interacting with the Document Object Model (DOM) or require complex modifications to the *manifest.json* file, such as adapting scripts to work with service workers, which results to our converter not providing 100% coverage for all testcases. Furthermore, we note an open-source project known as the extension manifest converter [19] which takes some of these conversions into account. However, this project is not actively maintained and supports a more limited range of APIs compared to our autoconverter, highlighting the more comprehensive scope and enhanced capabilities of our tool.

Benign vs Malicious Extensions: The conversion process, including manifest and API changes, applies equally to both benign and malicious extensions, ensuring that both types

webstore ID	**# Extensions (%)**
Online	2,649 (84.7%)
Offline	479 (15.3%)
Total	**3,128 (100%)**

TABLE III. EXTENSIONS THAT WERE ROLLED-BACK FROM V3 BACK TO V2 AND ONLINE AVAILABILITY IN THE WEBSTORE.

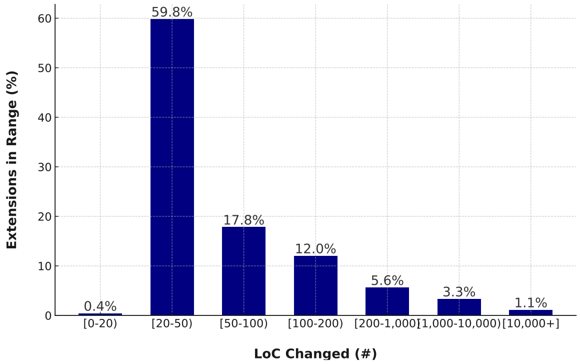

Fig. 4. How many LoC needed to convert the V2 malicious extensions to a V3 equivalent version before running the dynamic analysis on the V3 ones.

have the same likelihood of functioning after conversion. This is due to a broad set of changes affecting all extensions regardless of their intent. The neutrality of our conversion process means that the outcome for each extension, whether benign or malicious, is determined by its compatibility with Manifest Version 3 specifications, not its original purpose.

E. Dynamic Analysis

To test if a malicious extension remains functional after conversion, we employ playwright instrumentation to load the extension in a Chromium browser. We then navigate to websites allowed by the extension's manifest.json configuration. Successful website visits indicate that our conversion process has preserved the extension's functionality.

F. Malicious Extensions Definition

In our final methodology phase, we dynamically test converted V2 extensions in the Manifest V3 framework to assess their active status. We use *Catapult* for secure session recording and replay, preventing direct external server communication. By comparing sessions with and without the extension, we identify extension-specific requests and check them against a list of known malicious *URLs*, including *C&C* servers and domains listed on *EasyList* and *EasyPrivacy* [43].

Functionally Active: We categorize an extension as a functionally active malicious extension in Manifest V3 when it meets specific criteria: it has a history of verified malicious behavior, has been removed from the Google Webstore, falls into one of the malicious categories after manual verification of its behavior, successfully converts to Manifest V3 and loads correctly, and attempts to initiate a request to any *URL* from a list of known malicious domains. These criteria are vital for assessing whether extensions, previously recognized as malicious in their V2 form, continue to operate maliciously under Manifest V3.

V. RESULTS

A. Manifest V3 Adoption Rate

The analysis of Manifest V3 adoption in the Chrome Webstore highlights a steady transition from V2 to V3. Initially, the V3 conversion rate was below 5%, with only about 30,000 extension versions updated to V3. This trend saw a significant increase, with 90% of new uploads being in V3 by late 2023/early 2024, according to the data depicted in Figure 2 and Figure 3. These figures illustrate the monthly adoption rates, showing a consistent rise in V3 adoption from 2% in January 2021 to 40% of new daily updates by the time the migration plan was paused.

Result	Number of extensions (%)
Success (Initial)	154 (29.8%)
Fail (Initial)	363 (70.2%)
Success (after war modifications)	290 (56.1%)
Fail (Final)	227 (43.9%)
Executed	**517 (100%)**

TABLE IV. OUTCOME OF DYNAMIC EXECUTION OF MALICIOUS V3 EXTENSIONS

Furthermore, our findings on rollback rates, where extensions reverted to V2 after initially updating to V3, involve 3,129 extensions. Table III details that 15.3% of these rolled-back extensions were removed from the Webstore, indicating challenges with V3 adoption due to its limitations or concerns over malicious behavior. This comprehensive analysis underscores a significant yet gradual shift towards Manifest V3, marked by initial hesitancy and eventual widespread acceptance among developers.

B. LoC Distribution after Conversion to V3

The conversion of malicious V2 extensions to Manifest V3 showed varied changes in lines of code (LoC), which we detail in Figure 4. Notably, conversions rarely involved fewer than 20 changed LoC. The majority, 89.6%, saw changes ranging from 20 to 200 LoC. Outliers, constituting 1.1% of the dataset, experienced changes exceeding 10,000 LoC, often in extensions with large source code bundles. This highlights the diverse effects of transitioning to V3, influenced by the extension's complexity and code structure.

The significant changes in lines of code are attributed to the size of libraries included in the extension source code and our modification process. Initially, we substitute APIs in the code, followed by code beautification to facilitate line count. If a large library contains approximately 10,000 lines of code on a single line and we perform a substitution there, then beautify it, this results in it being counted as 10,000 lines of code changed.

C. Improved Security of Manifest V3

Manifest V3 has improved security by altering API usage, with changes detailed in Table V, informed by prior work on vulnerable and malicious extensions [50], [55]. These changes target APIs previously linked to vulnerabilities or malicious activities. Table VI shows the usage of such APIs in malicious extensions, highlighting the impact of these modifications. For example, the *eval* function, highly susceptible to misuse, and APIs like *XMLHttpRequest* and *fetch*, are extensively used in malicious activities, as evidenced by their high usage rates.

D. Dynamic Execution for V3 Malicious Extensions

In our analysis, we dynamically tested all 517 extensions converted to Manifest V3, with 154 exhibiting malicious

API Category	**API Name**	**Vulnerability Related**	**Malicious code Related**
Background Pages Related APIs	runtime.sendMessage	✓	✗
	runtime.connect	✓	✗
	runtime.onMessage.addListener	✓	✗
	runtime.onConnect.addListener	✓	✗
Web Request API	webRequest	✓	✓
	webRequestBlocking	✓	✓
Content Scripts and Cross-Origin Requests	XMLHttpRequest	✓	✓
	fetch	✓	✓
Remotely Hosted Code	eval	✗	✓

TABLE V. APIS THAT WERE DEPRECATED, REPLACED OR OFFERED ALTERNATIVES IN MANIFEST V3 WHICH ARE ALSO RELATED TO VULNERABLE AND MALICIOUS JAVASCRIPT CODE IN BROWSER EXTENSIONS.

API Category	**API Name**	**Total API Hits**	**Unique Extension Hits**	**API exists (%)**
Background Pages Related APIs	runtime.sendMessage	1,386	112	21.7
	runtime.connect	242	69	13.3
	runtime.onMessage.addListener	874	88	17.0
	runtime.onConnect.addListener	50	21	4.1
Web Request API	webRequest	1,004	85	16.4
	webRequestBlocking	234	35	6.8
Content Scripts and Cross-Origin Requests	XMLHttpRequest	4,340	340	65.8
	fetch	3,972	312	60.3
Remotely Hosted Code	eval	6,654	454	87.8
Total Extensions	**N/A**	**N/A**	**517**	**100**

TABLE VI. TOTAL HITS OF APIS RELATED TO VULNERABLE AND MALICIOUS CODE IN DYNAMICALLY TESTED MALICIOUS EXTENSIONS.

behavior post-conversion, representing 29.8% of the tested set. Results in Table IV reveal 70.2% of extensions failed to display malicious activity due to *web_accessible_resources* or DOM issues. Table VII categorizes the behaviors of extensions that ran successfully, showing user data analytics and browser modifications as dominant categories, as initially categorized in Table I. The comprehensive results of the extension conversion process, post implementation of the proof of concept, are detailed in Table II. This table reveals that 56.1% of the total extensions retained functionality after conversion to V3 and the application of the proof of concept.

E. Proof of Concept for Code Injection

Manifest V3 changes how third-party code is included in extensions, requiring executable code to be part of the source code, yet allowing for external resource interaction through *web_accessible_resources* files declared in the *manifest.json*. This setup enables extensions to integrate third-party scripts, potentially creating a chain of JavaScript file inclusions. Listings 1 and 2 provide a proof of concept, showing the declaration for external interactions and how remote server files can inject further JS scripts. This method opens avenues for various malicious activities, including session cookie theft and password compromises.

VI. DISCUSSION & MITIGATIONS

A. Pioneering Study & Community Response

This study is the first to analyze V3 adoption and its impact on malicious extensions, revealing that V3 does not eliminate any malicious category entirely. It also provides proof of concept for converting V2 extensions to V3, preserving their functionalities.

Manifest V3's rollout has sparked debate, especially among adblocker developers concerned about reduced efficacy [23], [20]. The transition to the *declarativeNetRequest (DNR)* API from *webRequest* has raised issues about extension capability and innovation [12], [28], [32], [33], despite some studies indicating potential performance benefits for privacy extensions [4], [48]. Challenges with service workers' DOM API access and library compatibility further highlight the need for V3 improvements.

B. Positives of V3 & Mitigation Strategies

Manifest V3 has introduced measures enhancing extension security, notably making malicious XMLHttpRequest (XHR) and fetch requests harder to execute by restricting XHR to content scripts and requiring Cross-Origin Resource Sharing (CORS) headers adjustment [15]. The review process for extensions is also being improved [11], [21], [10], [30], raising security standards.

To further mitigate risks, strategies include deploying 3rd-party watchers to monitor *web_accessible_resources* files for malicious changes, dynamically executing extensions to identify unauthorized redirects. Adjusting Content Security Policy (CSP) rules to limit malicious extensions without affecting

Malicious label	**Number of functional extensions**
Click scam	15
Ad replacement	77
User data analytics	123
Credentials stealing	2
Browser modification	88
Other	29

TABLE VII. MALICIOUS V3 EXTENSIONS THAT RAN SUCCESSFULLY PER MALICIOUS CATEGORY

```
// manifest.json
[..]
"content_security_policy": {
  "extension_pages": "script-src 'self'; object-
  src 'self'"},
"web_accessible_resources": [
  {"resources": ["src/injects_3rd_party.js"],
    "matches": ["https://*/*"]}]
[..]
```

Listing 1. 3rd party inclusion - manifest.json

```
// (a) injects_3rd_party.js
(function (e) {
    var site = "//malicious_site.com";
    var script = e.createElement("script");
    script.src = "https:" + site + "/js/
  malicious_3rd_party_payload.js";
    (e.head || e.body).appendChild(script)
}) (window, document);
---------------------------
// (b) malicious_3rd_party_payload.js
do_malicious_stuff()
```

Listing 2. Inclusion of 3rd party: (a) local extension file and (b) external payload

benign ones and enhancing manual reviews coupled with advocating for open-source code to allow community-driven security checks are proposed, building on top of tools that analyze extensions already like *CRXcavator* [42] and *LayerX* [44]. These strategies, although challenging due to potential evasion techniques [52], are crucial for enhancing V3 security and ensuring a safer extension ecosystem.

VII. LIMITATIONS & FUTURE WORK

A. *Evasion Techniques in Malicious Extensions*

Our evaluation of malicious extensions in V3 highlighted the impact of evasion techniques on our ability to detect malicious behaviors, as detailed in Table VIII. These techniques range from monitoring user actions to delaying malicious payloads, showcasing the adaptability of malicious actors. Addressing these sophisticated evasion tactics is crucial for enhancing security measures.

B. *Future work*

Future efforts will broaden our analysis to additional browsers like Firefox, Opera, Safari, and Edge, focusing on the adaptation to Manifest V3 and service workers. Gathering a comprehensive dataset of malicious extensions across these platforms presents a significant challenge. We also aim to assess the V3 ecosystem's evolution post-V2 phase-out, especially its impact on the extension landscape and security measures.

Evasion techniques
Deactivate if IP is from a university
Deactivate if user's search query contains the words ”C&C” or other similar phrases
Deactivate if user accesses locally hosted websites
Wait 3 days until malicious payload is downloaded
Check against a hard-coded list of privacy extensions
Check whether user is accessing the Developer Tools API
Check if user is tech-savvy (combination of above)

TABLE VIII. EVASION TECHNIQUES USED BY MALICIOUS EXTENSIONS

VIII. RELATED WORK

The extension ecosystem, active for over a decade, has been extensively researched by the scientific community.

A significant portion of this research focuses on identifying and analyzing malicious extensions. Kapravelos et al. [53] developed a method to detect such extensions using honey pages. Weissbacher et al. [59] investigated extensions leaking browser history, whereas Chen et al. [49] performed analysis based on data sources and sinks in extensions. Starov et al. [57] examined privacy diffusion in extensions, creating a tool named *PrivacyMeter* for this purpose. Research by Pantelaios et al. [55] involved identifying malicious extensions through clustering of similar JS API changes.

In the realm of privacy preservation and vulnerable data flows, Fass et al. [50] introduced *DoubleX*, a tool for detecting vulnerable data flows generated by extensions. Zhao et al. [61] focused specifically on privacy leaks in Chinese extensions. Starov et al. [58] created an extension dedicated to privacy preservation. The *Empoweb* tool, aimed at identifying APIs used by vulnerable extensions, is another notable contribution in this area [56]. Giuffrida et al. [51] developed a model for detecting privacy breaches in cross-browser extensions, and Li et al. [54] introduced *SpyShield*, a tool for preserving privacy in add-ons.

The study by Borgolte et al. [48] highlighted the performance benefits of privacy-focused extensions. This work was cited by ad blocker developers criticizing the limitations of Manifest V3 [23]. Xie et al. [60] investigated privacy leaks in Chrome extensions using *JTaint*, a JavaScript analysis tool. Zhu et al. [62] developed a lightweight, stealthy adblocking browser. Finally, Agarwal et al. identified that more than 2,400 extensions interfere with security headers in the domain of extension security headers [47].

IX. CONCLUSION

Manifest V3 significantly enhances Chrome extension security by deprecating or removing 87.8% of vulnerable APIs. Despite this, 154 (29.8%) of analyzed malicious extensions remain operational after conversion. Initially, less than 5% of extensions adopted V3, but now 90% of new uploads are in V3. Our analysis of 517 malicious extensions indicates a reduction in functionality post-V3, yet 290 (56%) adapt using web accessible resources for malicious activities. These findings underline the ongoing need for improvements in the extension ecosystem.

REFERENCES

[1] “Manifest v3,” https://developer.chrome.com/docs/extensions/mv3/intro/, 2020.

[2] “declarativenetrequest api,” https://developer.chrome.com/docs/extensions/reference/declarativeNetRequest/, 2021.

[3] “Service worker api,” https://developer.chrome.com/docs/workbox/service-worker-overview/, 2021.

[4] “Adblocker performance study by ghostery,” https://whotracks.me/blog/adblockers_performance_study.html, 2022.

[5] “Api deprecation in v3,” https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/, 2022.

[6] “arstechnica: extesions stealing user accounts,” https://arstechnica.com/information-technology/2020/10/popular-chromium-ad-blockers-caught-stealing-user-data-and-accessing-accounts/, 2022.

[7] “bleepingcomputer: extension abusing chrome,” https://www.bleepingcomputer.com/news/security/malicious-extension-abuses-chrome-sync-to-steal-users-data/, 2022.

[8] “bleepingcomputer: extesions stealing user analytics,” https://www.bleepingcomputer.com/news/security/facebook-sues-makers-of-malicious-chrome-extensions-for-scraping-data/, 2022.

[9] “Blocksite extension,” https://chrome.google.com/webstore/detail/blocksite-block-websites/eiimnmioipafcokbfikbljfdeojpcgbh, 2022.

[10] “Browser v3 adoption,” https://extensionworkshop.com/documentation/develop/manifest-v3-migration-guide/, 2022.

[11] “Browsers collaborating with developers,” https://blog.mozilla.org/addons/2022/05/18/manifest-v3-in-firefox-recap-next-steps/, 2022.

[12] “Browser’s existing policies solutions,” https://www.eff.org/deeplinks/2019/07/googles-plans-chrome-extensions-wont-really-help-security, 2022.

[13] “Catapult web page replay,” https://chromium.googlesource.com/catapult/+/HEAD/web_page_replay_go/README.md, 2022.

[14] “chrome-stats.com,” https://chrome-stats.com/, 2022.

[15] “Cors header updated in chrome v85,” https://www.chromium.org/Home/chromium-security/extension-content-script-fetches/, 2022.

[16] “dataspii: extensions stealing user information,” https://www.salon.com/2019/07/22/malicious-browser-extensions-are-stealing-personal-information/, 2022.

[17] “Extension content scripts,” https://developer.chrome.com/docs/extensions/mv3/content_scripts/, 2022.

[18] “Extension malware from north korea,” https://www.cisa.gov/uscert/ncas/alerts/aa20-301a, 2022.

[19] “Extension manifest converter,” https://github.com/GoogleChromeLabs/extension-manifest-converter, 2022.

[20] “Extensions’ potential malfunction article,” https://www.techrepublic.com/article/google-makes-the-perfect-case-for-why-you-shouldnt-use-chrome/, 2022.

[21] “Firefox collaboration on extensions,” https://blog.mozilla.org/addons/2021/05/27/manifest-v3-update/, 2022.

[22] “ghacks: extension faking authenticator,” https://www.ghacks.net/2021/05/18/dont-download-this-microsoft-authenticator-extension-for-chrome-it-is-fake/, 2022.

[23] “Ghostery extension report,” https://www.ghostery.com/blog/manifest-v3-the-ghostery-perspective, 2022.

[24] “Gigamon: extensions with criminal implications,” https://blog.gigamon.com/2018/01/18/malicious-chrome-extensions-enable-criminals-to-impact-half-a-million-users-and-global-businesses/, 2022.

[25] “kaspersky: extensions stealing user data,” https://www.kaspersky.com/blog/chrome-plugins-alert/38242/, 2022.

[26] “Lastpass extension,” https://chrome.google.com/webstore/detail/lastpass-free-password-ma/hdokiejnpimakedhajhdlcegeplioahd, 2022.

[27] “lifehacker: great suspender extension,” https://lifehacker.com/ditch-the-great-suspender-before-it-becomes-a-security-1845989664, 2022.

[28] “Manifest v3 can hurt innocation,” https://www.eff.org/deeplinks/2021/12/googles-manifest-v3-still-hurts-privacy-security-innovation, 2022.

[29] “medium: extensions stealing cryptowallet data,” https://medium.com/mycrypto/discovering-fake-browser-extensions-that-target-users-of-ledger-trezor-mew-metamask-and-more-e281a2b80ff9, 2022.

[30] “Microsoft v3 rollout,” https://docs.microsoft.com/en-us/microsoft-edge/extensions-chromium/developer-guide/manifest-v3, 2022.

[31] “Playwright browser simulation,” https://playwright.dev/, 2022.

[32] “Potential threats of v3,” https://www.eff.org/deeplinks/2021/12/chrome-users-beware-manifest-v3-deceitful-and-threatening, 2022.

[33] “Privacy extensions changes,” https://www.theregister.com/2022/06/08/google_blocking_privacy_manifest/, 2022.

[34] “radware: crypto stealing extensions,” https://blog.radware.com/security/2018/05/nigelthorn-malware-abuses-chrome-extensions/, 2022.

[35] “reddit: extension faking authenticator,” https://www.reddit.com/r/chrome/comments/hbpi7z/found_a_extension_that_contains_malware/, 2022.

[36] “ublock origin extension,” https://chrome.google.com/webstore/detail/ublock-origin/cjpalhdlnbpafiamejdnhcphjbkeiagm, 2022.

[37] “xda-devlopers: great suspender extension,” https://www.xda-developers.com/google-chrome-the-great-suspender-malware/, 2022.

[38] “zdnet: extensions stealing cryptowallet private keys,” https://www.zdnet.com/article/chrome-extension-caught-stealing-crypto-wallet-private-keys/, 2022.

[39] “zdnet: extensions stealing sensitive user data,” https://www.zdnet.com/article/google-removes-106-chrome-extensions-for-collecting-sensitive-user-data/, 2022.

[40] “zdnet: facebook suing extension developers,” https://www.zdnet.com/article/facebook-sues-two-chrome-extension-makers-for-scraping-user-data/, 2022.

[41] “Collection of malicious extensions report,” https://github.com/mallorybowes/chrome-mal-ids, 2024.

[42] “Crxcavator: Code analyzer,” https://crxcavator.io/, 2024.

[43] “Easylist blocklisted urls,” https://github.com/easylist/easylist, 2024.

[44] “Layerx: Code analyzer,” https://layerxsecurity.com/, 2024.

[45] “Unsupported apis,” https://developer.chrome.com/docs/extensions/develop/migrate/api-calls#replace-unsupported-apis, 2024.

[46] S. Agarwal, “Helping or hindering? how browser extensions undermine security,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 23–37. [Online]. Available: https://doi.org/10.1145/3548606.3560685

[47] S. Agarwal and B. Stock, “First, do no harm: Studying the manipulation of security headers in browser extensions,” *Proceedings 2021 Workshop on Measurements, Attacks, and Defenses for the Web*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233288141

[48] K. Borgolte and N. Feamster, “Understanding The Performance Costs and Benefits of Privacy-focused Browser Extensions,” in *Proceedings of the 29th The Web Conference (TheWebConf, formerly known as WWW)*, T.-Y. Liu and M. van Steen, Eds. International World Wide Web Conference Committee (IW3C2), 2020. [Online]. Available: http://dx.doi.org/10.1145/3366423.3380292

[49] Q. Chen and A. Kapravelos, “Mystique: Uncovering information leakage from browser extensions,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1687–1700. [Online]. Available: https://doi.org/10.1145/3243734.3243823

[50] A. Fass, D. F. Somé, M. Backes, and B. Stock, “Doublex: Statically detecting vulnerable data flows in browser extensions at scale,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1789–1804. [Online]. Available: https://doi.org/10.1145/3460120.3484745

[51] C. Giuffrida, S. Ortolani, and B. Crispo, “Memoirs of a browser: A cross-browser detection model for privacy-breaching extensions,” in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS ’12. New York, NY, USA: Association for Computing Machinery, 2012, p. 10–11. [Online]. Available: https://doi.org/10.1145/2414456.2414461

[52] J. Jueckstock, P. Snyder, S. Sarker, A. Kapravelos, and B. Livshits, "Measuring the Privacy vs. Compatibility Trade-off in Preventing Third-Party Stateful Tracking," in *Proceedings of The Web Conference (WWW)*, Apr. 2022.

[53] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson, "Hulk: Eliciting malicious behavior in browser extensions," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014.

[54] Z. Li, X. Wang, and J. Y. Choi, "Spyshield: Preserving privacy from spy add-ons," in *Recent Advances in Intrusion Detection*, C. Kruegel, R. Lippmann, and A. Clark, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 296–316.

[55] N. Pantelaios, N. Nikiforakis, and A. Kapravelos, *You've Changed: Detecting Malicious Browser Extensions through Their Update Deltas*. New York, NY, USA: Association for Computing Machinery, 2020, p. 477–491. [Online]. Available: https://doi.org/10.1145/3372297.3423343

[56] D. F. Somé, "Empoweb: Empowering web applications with browser extensions," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 227–245.

[57] O. Starov and N. Nikiforakis, "Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, p. 1481–1490. [Online]. Available: https://doi.org/10.1145/3038912.3052596

[58] ——, "Privacymeter: Designing and developing a privacy-preserving browser extension," in *Engineering Secure Software and Systems*, M. Payer, A. Rashid, and J. M. Such, Eds. Cham: Springer International Publishing, 2018, pp. 77–95.

[59] M. Weissbacher, E. Mariconti, G. Suarez-Tangil, G. Stringhini, W. Robertson, and E. Kirda, "Ex-ray: Detection of history-leaking browser extensions," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, ser. ACSAC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 590–602. [Online]. Available: https://doi.org/10.1145/3134600.3134632

[60] M. Xie, J. Fu, J. He, C. Luo, and G. Peng, "Jtaint: Finding privacy-leakage in chrome extensions," in *Information Security and Privacy*, J. K. Liu and H. Cui, Eds. Cham: Springer International Publishing, 2020, pp. 563–583.

[61] Y. Zhao, L. Yang, Z. Li, L. He, and Y. Zhang, "Privacy model: Detect privacy leakage for chinese browser extensions," *IEEE Access*, vol. 9, pp. 44 502–44 513, 2021.

[62] S. Zhu, U. Iqbal, Z. Wang, Z. Qian, Z. Shafiq, and W. Chen, "Shadowblock: A lightweight and stealthy adblocking browser," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2483–2493. [Online]. Available: https://doi.org/10.1145/3308558.3313558