From Manifest V2 to V3: A Study on the Discoverability of Chrome Extensions

Valerio Bucci and Wanpeng Li

Department of Computing Science,
University of Aberdeen, UK, AB24 3UE
{v.bucci.23, wanpeng.li}@abdn.ac.uk

Abstract. Browser extensions allow users to customise and improve their web browsing experience. The Manifest protocol was introduced to mitigate the risk of accidental vulnerabilities in extensions, introduced by inexperienced developers. In Manifest V2, the introduction of web-accessible resources (WARs) limited the exposure of extension files to web pages, thereby reducing the potential for exploitation by malicious actors, which was a significant risk in the previous unrestricted access model. Building on this, Manifest V3 coupled WARs with match patterns, allowing extension developers to precisely define which websites can interact with their extensions, thereby limiting unintended exposures and reducing potential privacy risks associated with websites detecting user-installed extensions. In this paper, we investigate the impact of Manifest V3 on WAR-enabled extension discovery by providing an empirical study of the Chrome Web Store. We collected and analysed 108,416 extensions and found that Manifest V3 produces a relative reduction in WAR detectability ranging between 4% and 10%, with popular extensions exhibiting a higher impact. Additionally, our study revealed that 30.78% of extensions already transitioned to Manifest V3. Finally, we implemented *X-Probe*, a live demonstrator showcasing WAR-enabled discovery. Our evaluation shows that our demonstrator can detect 22.74% of Manifest V2 and 18.3% of Manifest V3 extensions. Moreover, within the 1000 most popular extensions, the detection rates rise to a substantial 58.07% and 47.61%, respectively. In conclusion, our research shows that developers commonly associate broad match patterns to their WARs either because of poor security practices, or due to the inherent functional requirements of their extensions.

Keywords: Browser extension fingerprinting · Web-accessible resources · Browser extension detection.

1 Introduction

With the rapid proliferation of internet-based technologies and applications, web security has become a primary concern in the modern digital era. The emergence of browser extensions has played a significant role in enhancing the user experience by enabling users to customise and augment their browsing activities with a vast array of functionalities. However, the development and design

of browser extensions often present a difficult trade-off between usability and security, necessitating careful consideration in order to achieve an optimal balance [2, 3, 6, 29, 39].

In particular, as most extensions are developed by non-professional programmers, they may exhibit unintentional vulnerabilities, exposing users to network attackers or malicious websites. To address this issue, Barth et al. [3] introduced in 2010 a browser architecture which implements the principles of least privilege, privilege separation, and process isolation. Their solution involves a protocol called "*Manifest*", which requires extension developers to declare in advance a list of required and optional permissions. In the original Manifest protocol, however, websites could potentially access all resources within a browser extension. This posed a significant security risk because malicious websites could exploit this access to manipulate the extension's functionality or to exfiltrate sensitive information. Thus, in 2012, Manifest V2 introduced the concept of web-accessible resources (WARs), allowing developers to explicitly expose certain files to the web, thereby providing a more controlled and secure environment from potential misuse or unintentional vulnerabilities. Furthermore, WARs are accessed from a URL which embeds the associated extension's identifier, used by browsers to validate the integrity of its exposed files.

However, in 2017, Sjösten et al. [42] found that this measure allows the discovery of installed extensions by requesting large quantities of known URLs, associated to publicly-declared WARs. Consequently, by verifying the existence of a certain resource, adversarial websites can unequivocally conclude that the corresponding extension is installed on a visiting browser. Such exposure can lead to serious privacy infringements. For instance, it can reveal personal information about a user, such as the use of specific extensions like password managers, ad-blockers, or accessibility tools. Moreover, this exposure can enhance fingerprinting, as the combination of detected extensions may significantly boost the uniqueness of browser fingerprints, favouring stateless identification and cross-site tracking [12,21,23,32,48]. The uniqueness of these profiles may be further exacerbated when extensions are installed from different, interoperable web stores, such as those of Opera and Edge, as their WARs are reached via different URLs, thereby expanding the fingerprint complexity. Furthermore, this form of tracking can occur without knowledge or consent by users, further exacerbating the privacy concerns associated with browser extensions [16, 18].

As a mitigation to WAR-enabled extension discovery, in 2020, Manifest V3 introduced the concept of match patterns, allowing developers to further control the exposure of their WARs through predefined URL restrictions. Nevertheless, the efficacy of match patterns in thwarting WAR-enabled extension discovery is contingent upon their adoption by extension developers. Not only is the adoption rate of Manifest V3 yet to be reported on, but it is also unclear whether match patterns have produced a significant impact in curtailing extension discovery.

1.1 Our Contributions

In order to understand how match patterns are affecting WAR-enabled extension discovery, in this paper, we conduct an empirical study on the Chrome Web Store. With this focus, we compare the relative difference in discoverability between Manifest V2 and V3 extensions. Thus, our contributions are as follows:

1. We provide the first research effort of its kind to evaluate the impact of Manifest V3 on WAR-enabled discovery, by conducting an empirical study of the Chrome Web Store, and observe that 30% of extensions already transitioned to Manifest V3. However, most implemented match patterns do not preclude discovery. Overall, we measure discoverable Manifest V2 and V3 extensions to be 22.74% and 18.3%.
2. We introduce *X-Cavate*, a framework to construct a database of WARs, allowing to identify discoverable extensions. X-Cavate produces a database which can be updated regularly and yields a selection of WARs to be surveyed for conducting extension discovery.
3. We implement a live demonstrator, called *X-Probe*. Based on our evaluation, X-Probe can identify 21.34%, 38.16%, 54.9%, and 63% of Chrome Web Store extensions overall, and within the top 10,000, 1,000, and 100 most popular extensions, respectively. Additionally, we compare the performance of X-Probe against existing work on extension detection.
4. We propose additional measures for mitigating WAR-enabled discovery.

The remainder of our paper is structured as follows. Section 2 provides an overview of the Chrome Extension System. Section 3 describes how WARs can be exploited to detect installed extensions. Section 4 delineates the methodology of our study, illustrates the X-Cavate framework, and evaluates the X-Probe demonstrator. Section 5 showcases the results of our empirical study and provides a comparison of our results with previous literature. Section 6 discusses the implications of our results and drafts our conclusions. Section 7 identifies defensive measures to further limit WAR probing. Section 8 identifies the state of the art relatively to browser fingerprinting and extension detection. Section 9 summarises our contributions, highlights our key takeaways, and identifies potential avenues for future work.

2 The Chrome Extension System

Extensions are small programs which run in the browser, allowing user to customise their browsing experience. By interfacing with browser APIs, extensions may execute various functions such as manipulating web content, managing active tabs, accessing browsing history, and more. However, the use of extensions can introduce novel security risks, often as a consequence of their development by inexperienced programmers. This lack of expertise can lead to privilege escalation vulnerabilities, exposing users to exploitation by adversarial websites or network attackers. In 2009, Liverani and Freeman demonstrated such security

risk in Firefox extensions [28]. Successively, Barth et al. identified that the Firefox architecture allowed full access to its powerful API [3], making most extensions over-privileged. Thus, they proposed a novel browser architecture, implementing the principles of *least-privilege*, *privilege separation*, and *process isolation*. Their architecture lays the foundation for the Chrome Extension System, and was also adopted by other popular browsers, including Firefox and Safari.

2.1 Architecture Overview

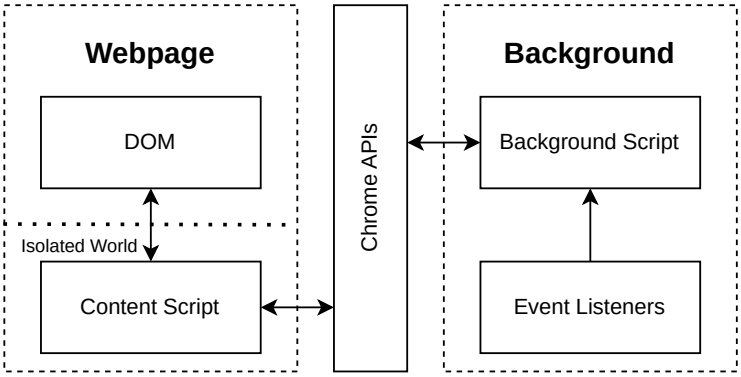

Fig. 1: A simplified representation of the Chrome Extension System.

The Chrome Extension System, illustrated in Figure 1, aims to strike a balance between flexibility and security, ensuring that developers can create powerful extensions while minimizing the risks associated with accidental exposures. It mainly consists of two interdependent components implementing privilege separation and process isolation: the *content script* and the *background script*[1]. Furthermore, the *Manifest* protocol is aimed at standardising the development of extensions and enforcing least-privilege by providing developers with granular access to the Chrome APIs.

Chrome APIs. The Chrome APIs form the backbone of the Chrome Extension System, providing a set of JavaScript interfaces for accessing browser functionalities. APIs are grouped by category, each providing access to distinct capabilities. Namely, the `tabs` API allows to manipulate browser tabs, while the `storage` API provides methods for storing and retrieving data. Furthermore, the `runtime` API facilitates communication between extension components, such as

[1] For the sake of simplicity, we omitted various secondary components. Further information can be found on Google's official documentation at `https://developer.chrome.com/docs/extensions/mv3/architecture-overview/`.

the content script and background script. This granular approach allows extensions to carry out tasks while adhering to the principle of least-privilege, with developers declaring required permissions via the *Manifest* protocol.

Manifest. Manifests consist in JSON files, conveying metadata about extensions, including their name, version number, description, and required permissions. At the time of writing, Google is spearheading a shift from Manifest V2 to V3, with browser extensions gradually transitioning to the latest iteration. Such transition is equally significant and controversial, as it redefines the permissions and capabilities of background scripts [11].

Background Script. Background scripts, as the name suggests, run in the background of extensions. They access the Chrome and JavaScript APIs to perform tasks that do not require user interaction, such as listening for events, sending HTTP requests to the Web, running timers, storing data, and broadcasting messages to other components. In Manifest V2, background scripts could be either *persistent* or *non-persistent*. Persistent background scripts remain active for the whole duration of a browser session. Non-persistent background scripts, instead, are automatically unloaded when idle. In Manifest V3, "traditional" background scripts were deprecated in favour of *service workers*. In contrast, service workers are non-persistent, event-driven, and guarantee unique instance behaviour across all extension pages, windows, or tabs. However, the transition to fully-asynchronous service workers has sparked controversy due to its limitations on synchronous functionalities, namely, in the `webRequest` API [4, 11, 13]. Nevertheless, independently of the Manifest version, background scripts detain most operational capabilities, except for DOM manipulation, which is delegated to the content script via message passing.

Content Script. Content scripts are closely tied to webpages, with each tab or window initializing its own content script. While interacting with unsanitised webpages, content scripts face significant restrictions in accessing Chrome APIs. To circumvent these limitations, they delegate operations requiring broader access to the background script. Additionally, content scripts operate within a specialized, sandboxed environment known as *isolated world*. The isolated world is a distinct JavaScript runtime that provides a unique space for interacting with the DOM, preventing exposures to the host runtime environment. Consequently, their access to extension files is also limited to the Web-Accessible Resources (WARs) declared in the Manifest.

Web-Accessible Resources. Before Manifest V2 introduced WARs in 2012, websites could access all files within extensions installed on browsers. This design was insecure because it allowed for malicious websites to perform fingerprinting or detect exploitable vulnerabilities in installed extensions [19, 20]. Additionally, it could lead to unintentional exposures of sensitive data by developers. In Manifest V2, the DOM is restrained from accessing extension filesystems. Instead,

Listing 1: WARs declaration in Manifest V2 (left) and V3 (right).

```
{
  ...
  "manifest_version": 2,
  ...
  "web_accessible_resources": [
    "images/*.png",
    "extension.css"
  ],
  ...
}
```

```
{
  ...
  "manifest_version": 3,
  ...
  "web_accessible_resources": [{
      "resources": ["images/*.png"],
      "matches": [
        "https://*.google.com/*"
      ]
    }, {
      "resources": ["extension.css"],
      "matches": ["<all_urls>"]
    }],
  ...
}
```

developers can optionally specify a list of WARs to be injected into the DOM, such as scripts, style sheets, or images. Each WAR can be defined as a specific path or as a wildcard encompassing a group of files. Consequently, WARs will be exposed to webpages at a specialised URL, embedding the extension identifier and the relative file path: `chrome-extension://[EXTENSION ID]/[PATH]`. The identifier is unique to each extension, and it allows browsers to validate the integrity of exposed files. Using Manifest V2, declared WARs are exposed to any arbitrary website. In contrast, as shown in Listing 1, Manifest V3 allows developers to implement further accessibility restrictions via match patterns.

Match Patterns. Match patterns specify which websites an extension can interact with. They are composed by a combination of URLs and wildcards. For example, a pattern such as `https://*.google.com/*` would match any URL within the Google domain. Since the inception of the Manifest protocol, match patterns have been used, namely, to grant host permissions, determining which websites an extension can access and modify. Additionally, they are also employed to dictate where content scripts should be injected, enhancing both performance and security. With the advent of Manifest V3, the role of match patterns has been expanded to compound WARs, allowing developers to restrict the exposure of WARs to specific websites, as shown in Listing 1. This measure provides developers with a more granular control over the accessibility of extension resources, mitigating potential misuse and reducing the risk of unexpected behaviour. However, extensions needing to inject their WARs in all websites require highly permissive match patterns. For instance, these include `https://*/*`, `*://*/*` and `<all_urls>`. Therefore, the effectiveness of match patterns is not only contingent on their thoughtful implementation by developers, but it can also be constrained by the functional necessities of extensions.

3 Probing WARs To Detect Extensions

As explained in Section 2.1, WARs are defined in the Manifest, and are accessed within the context of a webpage via extension-specific URLs. Such URLs embed unique extension identifiers, assigned by the publishing extension store. Consequently, as illustrated in Listing 2, a webpage could fetch a known WAR and observe whether the request is fulfilled. If so, the webpage can unequivocally determine that the corresponding extension is installed on the visiting browser. Furthermore, by collecting manifests from online stores, an adversarial website could survey a large dataset of known WARs to detect installed extensions. However, it is important to note that this technique has its limitations, as not all extensions employ WARs. Moreover, such exposure could be further mitigated in Manifest V3, provided developers employ stringent match patterns.

Listing 2: Working example to detect a password management extension.

```
fetch("chrome-extension://hdokiejnpimakedhajhdlcegeplioahd/overlay.html")
    .then(response => {
        if (response.ok) {
            console.log("LastPass is installed.");
        }
    }).catch(error => {
        console.log("LastPass is not installed.");
    });
```

4 Methodology

This section reports on the methodology employed in our study, conducted in March 2023, to evaluate the susceptibility of Chrome Web Store extensions to WAR-enabled discovery. To determine the popularity of each extension, we focused sequentially on the number of ratings, number of downloads, and star rating, all in descending order, to sort extensions from the most popular to the least popular. We prioritised the number of ratings as a metric due to observed inconsistencies and potential artificial inflation in download numbers, evidenced by some extensions having substantial downloads yet zero ratings.

In Section 4.1 we delineate the *X-Cavate* framework, employed to (a) collect identifiers and popularity metrics; (b) download extensions and extract their manifests; (c) construct a database of available extensions and associated features; and (d) produce a sample of WARs exposed to any arbitrary URL. Successively, in Section 4.2 we introduce our online demonstrator *X-Probe*, which implements extension discovery through the collected dataset.

4.1 X-Cavate Framework

For the purpose of this study, we developed a data collection utility following the *X-Cavate* framework, shown in Figure 2. While in this study we focused

our efforts on the Chrome Web Store, X-Cavate is aimed at automating the collection of extensions from any given online store. We present below the main modules constituting our proposed framework:

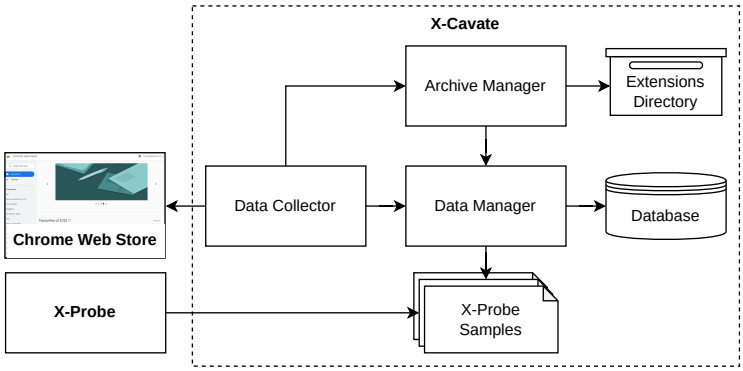

Fig. 2: Abstract structure of the X-Cavate Framework.

Data Collector. The Data Collector refers to a configuration file containing a URL to an online store and CSS selectors to the extension identifiers and associated metrics (e.g. downloads, ratings, and category) to be scraped from the DOM. After collecting an identifier, it downloads the corresponding extension. Finally, it provides the downloaded extensions and the scraped details to the Archive Manager and Data Manager, respectively.

Archive Manager. The Archive Manager handles downloaded extensions, which consist of CRX files – i.e. ZIP archives with additional headers. After an extension is downloaded, the Archive Manager stores it into a structured directory tree, strips the CRX headers, and extracts the `manifest.json` file. Successively, it redacts a list of exposed files, by matching the declared WARs (if any) with the files located in the archive. Therefore, if a WAR is not located in its specified path, it is not inserted in the list. Finally, the Archive Manager provides the extrapolated subset of WARs to the Data Manager.

Data Manager. The Data Manager processes online details scraped by the Data Collector and downloaded extensions' metadata, extrapolated by the Archive Manager. It maintains a normalised database by validating inserted records and ensuring relational consistency and integrity. The database architecture connects online information with data extrapolated from downloaded archives. Thus, it supports repeated insertions of various extension releases overtime, enabling researchers to perform long-term studies. Finally, based on popularity metrics, it compiles a series of datasets to be employed by the X-Probe demonstrator.

4.2 X-Probe Demonstrator

We introduce *X-Probe*, a practical implementation of WAR-enabled discovery, targeted at Chrome extensions[2]. X-Probe was evaluated against the four extensions shown in Table 1, each exposing their WARs with exclusively one vulnerable match pattern. We repeated our evaluation on various Chromium-based browsers (i.e. Chrome, Brave, Opera, and Edge) and observed consistent results.

Table 1: Extensions used to validate discoverable patterns.

Extension	Release	Manifest	Pattern	Detected?
Speed Dial	81.3.1	V3	<all_urls>	yes
Custom Cursor	3.3.0	V3	*://*/*	yes
Talend API Tester	25.11.2	V3	https://*/*	yes
ShopQuangChauVN	5.1	V3	http://*/*	yes
Hola VPN	1.208.689	V2	N/A	yes

X-Probe relies on four JSON-formatted datasets produced with the X-Cavate framework, containing extension identifiers, each paired with one exposed WAR. Each dataset represents either the top 100, 1000, and 10,000 most popular extensions on the Chrome Web Store. Additionally, a dataset containing all discoverable extensions was included in the demonstration. While the comprehensive dataset provides a thorough but resource-intensive scan, smaller datasets offer a quicker analysis, albeit limited. Based on our evaluation, X-Probe can identify in the varying datasets 21.34%, 38.16%, 54.9%, and 63% of extensions, overall and within the 10,000, 1,000, and 100 most popular extensions, respectively.

5 Results

In March 2023 we scraped details for 111,467 extensions, of which 108,416 were successfully downloaded and analysed. Of the 3051 failures, 105 were due to corrupted archives, while 2946 were visible on the Chrome Web Store, but no longer available for downloading. We ranked extensions by their rating count, download count, and average rating. We prioritised the rating count as it is a continuous metric, and because reviews can only be provided by authenticated Google users. Finally, we grouped extensions in the *Top 100*, *Top 1000*, and *Top 10,000* popularity groups, shown in Figure 3.

5.1 Susceptibility to WAR-Enabled Discovery

In total, 23,132 extensions are detectable via WAR-enabled discovery, accounting for 21.35% of the analysed set. Figure 4 shows a positive correlation between

[2] X-Probe can be tested on all Chromium-based browsers at: `https://xprobe.dev`

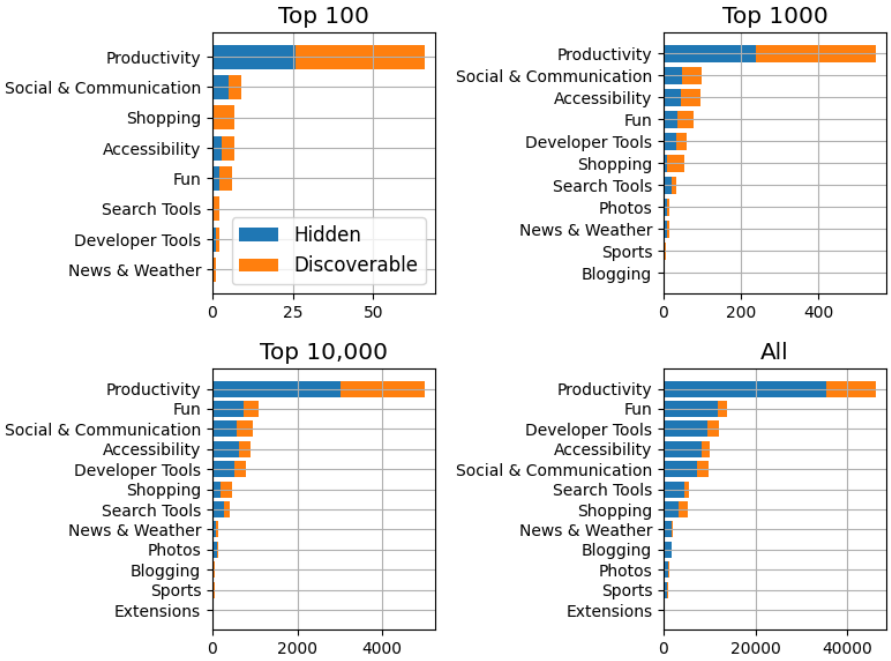

Fig. 3: Categories overview split across popularity groups.

popularity and discoverable proportions, with a detection rate of 64%, 59.4%, and 38.14% in Top 100, Top 1000, and Top 10,000 groups, respectively. Additionally, we observe that some categories are more susceptible to WAR-enabled discovery. Namely, "Shopping" extensions consistently exhibit higher detection rates than other categories. We performed a one-sided Mann-Whitney U test to determine if there was a significant difference in the distribution of rating counts between discoverable and hidden extensions. A statistic of approximately 1.33×10^{12} and p-value < 0.05, rejected the null-hypothesis. Therefore, we conclude that discoverable extensions have a higher rating count than hidden ones.

5.2 Manifest V3 Adoption Rate

In total, 75,048 extensions use Manifest V2, while 33,368 transitioned to V3, corresponding to 30.78%. As shown in Figure 5, Manifest V3 extensions account for 33%, 29.4%, and 28.89% of the Top 100, Top 1000, and Top 10,000 groups, respectively. Notably, none of the "Developer Tools" and "News & Weather" extensions in the Top 100 group transitioned to V3. Furthermore, "Blogging" extensions consistently exhibit lower adoption rates than other categories. This might be due to the requirements of extensions which extensively use background scripts and broad permissions to capture, modify, and deliver information: tasks

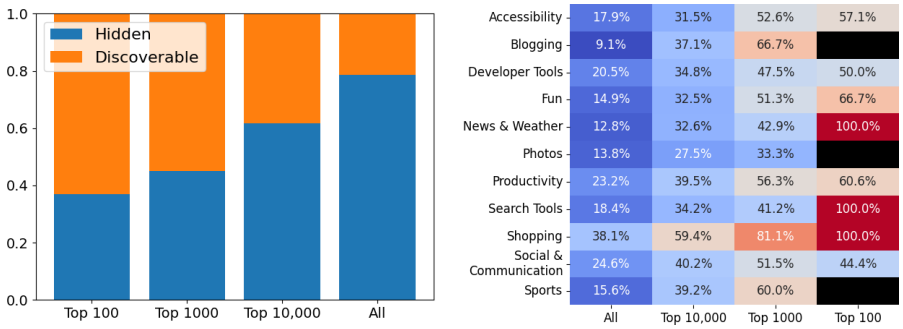

Fig. 4: WAR-enabled discovery rates.

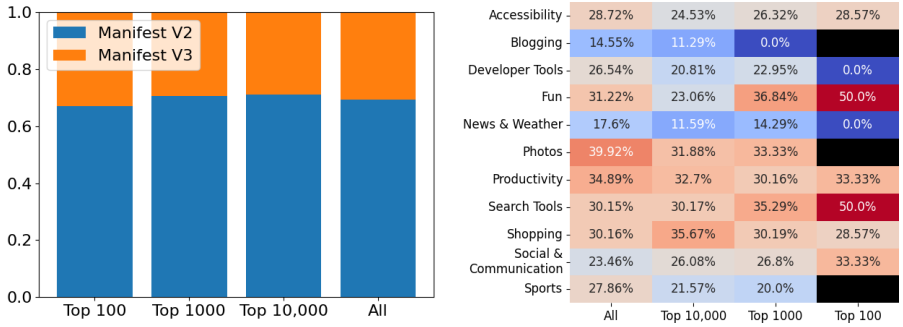

Fig. 5: Manifest V3 rate of adoption.

that could become increasingly challenging to perform under the more restrictive and event-driven environment of Manifest V3.

5.3 Manifest V3 Impact on WAR-Enabled Discovery

We employed Manifest V2 extensions to establish a baseline for evaluating the impact of Manifest V3 in mitigating WAR-enabled discovery. Successively, we compared the relative proportions of discoverable extensions to assess whether there was a difference between each Manifest iteration across popularity groups. Table 2 shows a consistent positive correlation between popularity and detectable rates in both Manifest iterations. However, we also observe an increasing mitigatory effect of Manifest V3, with a relative reduction ranging between 4% and 10%. In contrast, Manifest V2 extensions in the Top 100 group are less likely to be detectable than V3 extensions.

Table 2: Discoverable rates across popularity groups, measured between Manifest iterations ($\Delta = V2 - V3$).

Group	Overall	V2	V3	Δ
All	21.35%	22.74%	18.3%	4.44%
Top 10,000	38.14%	40.47%	32.39%	8.08%
Top 1000	55%	58.07%	47.61%	10.46%
Top 100	65%	62.68%	66.67%	-3.99%

Table 3: Comparison with previous work.

Paper	Attack Class	# Extensions	% Detectable
[48]	DOM analysis	10,000	9.2%
[24]	DOM analysis (CSS rules)	116,485	3.8%
[44]	DOM analysis (user actions)	102,482	2.87%
[16]	Multi-class	102,482	28.72%*
[30]	Multi-class	91,147	17.68%*
[42]	WAR probing	43,429	28%*
Ours	WAR probing	108,416	21.34%

Note: We report percentages over the complete set of analysed extensions when the original work reports absolute numbers. *Evaluated on WARs prior to Manifest V3.

5.4 Comparative Evaluation

Table 3 provides a comparison between our work and the existing literature in the field of extension discovery. Despite DOM analysis being a popular method, its effectiveness is limited, with detection percentages below 10%. A substantial leap in detectability was observed with the advent of WAR probing, as demonstrated by [42]. In the work of [16], a multi-class strategy was employed, merging WAR probing with DOM analysis and interception broadcast communication by extension components, achieving even higher detection rates. Notably, within their approach, WAR probing alone was able to detect 25.24% of extensions, underscoring its superior efficacy in extension discovery. Similarly, [30] implemented a strategy combining WAR probing with DOM analysis, although, their results were significantly lower than [16], with 11.37% of extensions being detected through WAR probing alone. However, these studies were conducted with WARs under the Manifest V2 framework. With the transition to the more restrictive Manifest V3, a new set of challenges emerges, leaving the effectiveness of these previous methods in the updated constraints uncertain. Our research addresses this challenge by adapting WAR probing to the Manifest V3 environment, providing an up-to-date study on the effectiveness of this strategy.

6 Discussion

6.1 Popular Extensions Are More Discoverable

The results from our analysis highlight a paradoxical correlation: as an extension's popularity increases, so too does their susceptibility to WAR-enabled discovery. In all analysed groups, there is a consistent trend of increased discoverability with increased popularity – as determined by rating count, download count, and average rating. However, this pattern has significant implications for user security and privacy. A particular concern lies in browser fingerprinting, as the set of detected extensions on a browser can greatly enhance the complexity of fingerprints, especially when coupled with other identifiable attributes. Thus, detectable extensions provide adversarial entities with an expanded toolset to uniquely identify and track users across the web in a stateless manner. This raises several questions regarding the security implications for the most used extensions, as they appear to be more visible and hence potentially more exposed to attacks. Extensions' popularity, in this context, might inadvertently serve as a double-edged sword. On the one hand, it makes these extensions more accessible to users, thereby contributing to their popularity. On the other hand, it simultaneously exposes them to potential malicious entities, aiming to carry out targeted attacks and cross-site tracking.

6.2 Developers Employ Broad Match Patterns

Manifest V3 allows developers to enforce least-privilege in their WARs through match patterns. Although Manifest V3 produces a quantifiable impact, its limitations lay in the assumption that developers will enact appropriate restrictions. Furthermore, many extensions provide functionalities which require exposing WARs to all websites. For example, extensions that modify the appearance of webpages, such as themes or custom cursors, need to inject their resources across all domains. Similarly, extensions that provide web development tools, like colour pickers or CSS inspectors, also require broad access to function effectively. In these cases, the use of highly permissive match patterns becomes a necessity rather than a choice. This implies that even with the best intentions, developers may be forced to compromise on least-privilege due to the inherent requirements of their extensions. Consequently, while Manifest V3's approach to WARs is a step in the right direction, it may not fully eliminate the risk of WAR-enabled discovery, especially for extensions that inherently require broad access.

7 Recommended Mitigation Measures

Based on our findings and expertise acquired while developing this project, we propose the following countermeasures for mitigating WAR-enabled discovery.

Limiting failed requests. WAR-enabled discovery involves probing a large sample of WARs. For instance, our demonstrator X-Probe employs a sample of 23,132 discoverable extensions. Consequently, the vast majority of surveyed WARs is expected to return an error from the browser API. Therefore, since this unreasonable amount of requests is unlikely to be performed for legitimate purposes, we propose the introduction of a failed-request cap in the browser API. Such cap could be enforced on the offending webpage by preventing its JavaScript environment from requesting further resources.

Limiting accessibility. Content scripts use the Chrome API method `getUrl` to obtain a resource's URL given its relative path in the extension repository. The process-isolation pattern allows browsers to determine whether `getUrl` was called from a content script or a webpage. Therefore, we propose a "*gate*" system, which only exposes WARs to a webpage after they are requested by the content script. While this measure would not prevent access to WARs of highly-active extensions (e.g. password managers), it may severely limit extension fingerprinting capabilities. Furthermore, as Manifest V3 introduces background service workers, webpages could be blocked from accessing the WARs of idle extensions.

User-enforced least-privilege. Manifest V3 empowered extension distributors to arbitrarily restrict WAR exposure. However, users intending to replicate such restriction have to disable extensions from their browser settings, or open an "incognito" session. Realistically, the average user is unlikely to perform this procedure each time they visit a new website. Therefore, we recommend that browsers introduce functionalities allowing users to enact such restrictions on demand. This would involve blocking all WAR requests originating from untrusted websites, and informing the user about request attempts. Finally, the user could either deny or authorise all requests. Alternatively, they could decide to expose specific extensions. Naturally, users should be also allowed to tailor default settings based on their privacy needs to avoid hindering usability.

8 Related Work

8.1 Extension Detection

Over the past decade, extensions have begun to emerge as a new area of study, with researchers exploring its potential uses and challenges in the context of online privacy and security [4–6, 8, 10, 16, 18, 33–36, 39–41, 46]. In this emerging field, different strategies have been developed for detecting extensions, mostly focused on analysing changes to the DOM and probing WARs.

DOM analysis. Starov and Nikiforakis [48] examined the top 10,000 Chrome Web Store extensions, and showed that at least 9.2% introduced detectable DOM changes on any arbitrary URL. Additionally, they developed a proof-of-concept

script, able to identify 90% of the analysed 1,656 identifiable extensions. Extending this research, Starov et al. [47] revealed that 5.7% of the 58,034 analysed extensions were detectable due to unnecessary DOM changes. Consequently, Laperdix et al. [24] investigated how CSS rules injected by content scripts can be used to detect installed extensions, revealing that 3.8% of the 116,485 analysed extensions could be uniquely identified with this method. Building on this existing body of knowledge, Solomos et al. [44] highlighted that user-triggered DOM changes had been overlooked by previous research. Thus, they identified 4,971 extensions, including over a thousand that were undetectable by previous methods. Additionally, they revealed that about 67% of extensions triggered by mouse or keyboard events could be detected through artificial user actions. Additionally, Solomos et al. [45] proposed continuous fingerprinting, a technique capable of capturing transient modifications made by extensions, previously undetectable due to their ephemeral nature. This technique substantially increases the coverage of extensions detectable through their DOM modifications.

WAR probing. Before the introduction of Manifest V3, Sjösten et al. [42] conducted the first comprehensive study of non-behavioral extension discovery, focusing on the detection of WARs in both Chrome and Firefox. Their empirical study, found that over 28% of the 43,429 analysed Chrome extensions could be detected. Building on their work, Gulyas et al. [12] conducted a study on 16,393 participants for evaluating how browser extensions detected with WARs contributed to the uniqueness of users. They found that 54.86% of users which installed at least one extension were uniquely identifiable. Additionally, they found that testing 485 carefully selected extensions produced the same level of uniqueness. Subsequently, Sjösten et al. [43] further examined the issue of detecting browser extensions by web pages, particularly focusing on the recent introduction of randomised WAR URLs by Mozilla Firefox, which they found could potentially compromise user privacy rather than protect it. They introduced "revelation attacks", to detect these randomised URLs in the code injected by content scripts, thereby enabling enhanced user tracking.

Combined techniques. Karami et al. [16] implemented a multi-class approach which involves DOM analysis, WAR probing, and interception of broadcast messages by extension components. Their technique detected 29,428 out of 102,482 extensions, demonstrating resilience against countermeasures to DOM analysis proposed by [50]. Although their results are best-performing among present literature, their evaluation dates before the diffusion of Manifest V3 on the Chrome Web Store. On the other hand, Lyu et al. [30] presented their approach comprising of DOM analysis and WAR probing, detecting 16,166 extensions out of 91,947, with 11,856 being detectable by their WAR probing approach. However, there is no mention of WAR match patterns or Manifest V3 throughout their paper. Furthermore, it is unclear why their WAR discovery rate (i.e. 13.01%) is comparatively lower than in previous literature and our results.

8.2 Browser Fingerprinting

There is a growing body of work exploring browser fingerprinting [51] and the ways it can be augmented by the virtually-unlimited combinations of potentially installed extensions [12, 17, 50]. Additionally, much work has been conducted on devising defensive and mitigatory measures [7, 9, 14, 15, 22, 25, 31, 49]. Finally, literature has focused on the utilisation of fingerprinting as a tool for streamlining user authentication [1, 37, 38], although, some have highlighted the security limitations of such methods [26, 27].

9 Conclusion

Manifest V3 coupled WARs with match patterns to further mitigate the exposure of extensions to webpages. We presented an empirical study on 108,416 Chrome Web Store extensions, with focus on WAR-enabled discovery. To the best of our knowledge, we are the first to evaluate the impact of Manifest V3 match patterns applied to WARs. Our results show that Manifest V3 produces a relative reduction in detectability, growing from 4% to 10% as extensions become more popular. In contrast, Manifest V3 extensions among the 100 most popular, exhibit a relative increase of 4% in detectability, when compared to V2. Furthermore, independently of the adopted Manifest iteration, popular extensions are more likely to be discoverable. We argue that match patterns do not fully eliminate the risk of WAR-enabled discovery, both because some developers neglect least-privilege practices, and due to inherent extension functionalities which require universal exposure of resources. Therefore, we proposed a range of defensive measures to be implemented on the browser side. Through a combination of these measures, we anticipate a significant improvement in preventing unwarranted probing of WARs.

In addition, we devise the *X-Cavate* framework to repetitively collect extensions from online stores and extract their Manifests to maintain a structured database overtime. Alongside X-Cavate, we developed a live demonstrator called *X-Probe* to emphasize the efficacy of WAR-enabled discovery. Based on our evaluation, X-Probe has proven its capability in detecting 22.74% of Manifest V2 and 18.3% of Manifest V3 extensions, overall. Moreover, relatively to the 1000 most popular extensions, the detection rates rise to a substantial 58.07% and 47.61%, respectively, further highlighting the severity of this exposure.

Future work could involve the integration of a diverse array of extension discovery techniques, alongside the development of a demonstrator that can function across various browser architectures. In addition, once the transition to Manifest V3 is fully completed, it could be especially insightful to conduct an updated user study. This would allow for an examination of the uniqueness of extension fingerprints, presenting a valuable opportunity to better understand and further contribute to this evolving field.

References

1. Andriamilanto, N., Allard, T., Le Guelvouit, G., Garel, A.: A Large-scale Empirical Analysis of Browser Fingerprints Properties for Web Authentication. ACM Transactions on the Web **16**(1), 4:1–4:62 (Sep 2021). https://doi.org/10.1145/3478026
2. Bandhakavi, S., Tiku, N., Pittman, W., King, S.T., Madhusudan, P., Winslett, M.: Vetting browser extensions for security vulnerabilities with Vex. Communications of the ACM **54**(9), 91–99 (Sep 2011). https://doi.org/10.1145/1995376.1995398
3. Barth, A., Felt, A.P., Saxena, P., Boodman, A.: Protecting Browsers from Extension Vulnerabilities. In: Network and Distributed System Security Symposium (2010)
4. Borgolte, K., Feamster, N.: Understanding the Performance Costs and Benefits of Privacy-focused Browser Extensions. In: Proceedings of The Web Conference 2020. pp. 2275–2286. ACM, Taipei Taiwan (Apr 2020). https://doi.org/10.1145/3366423.3380292
5. Bui, D., Tang, B., Shin, K.G.: Detection of Inconsistencies in Privacy Practices of Browser Extensions. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 2780–2798 (May 2023). https://doi.org/10.1109/SP46215.2023.10179338
6. Calzavara, S., Bugliesi, M., Crafa, S., Steffinlongo, E.: Fine-Grained Detection of Privilege Escalation Attacks on Browser Extensions. In: Vitek, J. (ed.) Programming Languages and Systems. pp. 510–534. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46669-8_21
7. Datta, A., Lu, J., Tschantz, M.C.: Evaluating Anti-Fingerprinting Privacy Enhancing Technologies. In: The World Wide Web Conference. pp. 351–362. WWW '19, Association for Computing Machinery, New York, NY, USA (May 2019). https://doi.org/10.1145/3308558.3313703
8. Eriksson, B., Picazo-Sanchez, P., Sabelfeld, A.: Hardening the security analysis of browser extensions. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. pp. 1694–1703. SAC '22, Association for Computing Machinery, New York, NY, USA (May 2022). https://doi.org/10.1145/3477314.3507098
9. FaizKhademi, A., Zulkernine, M., Weldemariam, K.: FPGuard: Detection and Prevention of Browser Fingerprinting. In: Samarati, P. (ed.) Data and Applications Security and Privacy XXIX. pp. 293–308. Lecture Notes in Computer Science, Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-20810-7_21
10. Fass, A., Somé, D.F., Backes, M., Stock, B.: DoubleX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 1789–1804. CCS '21, Association for Computing Machinery, New York, NY, USA (Nov 2021). https://doi.org/10.1145/3460120.3484745
11. Frisbie, M.: Understanding the Implications of Manifest V3. In: Frisbie, M. (ed.) Building Browser Extensions: Create Modern Extensions for Chrome, Safari, Firefox, and Edge, pp. 167–185. Apress, Berkeley, CA (2023). https://doi.org/10.1007/978-1-4842-8725-5_6
12. Gulyas, G.G., Some, D.F., Bielova, N., Castelluccia, C.: To Extend or not to Extend: On the Uniqueness of Browser Extensions and Web Logins. In: Proceedings of the 2018 Workshop on Privacy in the Electronic Society. pp. 14–27. WPES'18, Association for Computing Machinery, New York, NY, USA (Jan 2018). https://doi.org/10.1145/3267323.3268959

13. Gunnarsson, P., Jakobsson, A., Carlsson, N.: On the Impact of Internal Webpage Selection when Evaluating Ad Blocker Performance. In: 2022 30th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). pp. 41–48 (Oct 2022). https://doi.org/10.1109/MASCOTS56607.2022.00014
14. Hiremath, P.N., Armentrout, J., Vu, S., Nguyen, T.N., Minh, Q.T., Phung, P.H.: MyWebGuard: Toward a User-Oriented Tool for Security and Privacy Protection on the Web. In: Dang, T.K., Küng, J., Takizawa, M., Bui, S.H. (eds.) Future Data and Security Engineering. pp. 506–525. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-35653-8_33
15. Iqbal, U., Englehardt, S., Shafiq, Z.: Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1143–1161. IEEE Computer Society (May 2021). https://doi.org/10.1109/SP40001.2021.00017
16. Karami, S., Ilia, P., Solomos, K., Polakis, J.: Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In: Proceedings 2020 Network and Distributed System Security Symposium. Internet Society, San Diego, CA (2020). https://doi.org/10.14722/ndss.2020.24383
17. Karami, S., Kalantari, F., Zaeifi, M., Maso, X.J., Trickel, E., Ilia, P., Shoshitaishvili, Y., Doupé, A., Polakis, J.: Unleash the Simulacrum: Shifting Browser Realities for Robust {Extension-Fingerprinting} Prevention. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 735–752 (2022)
18. Kariryaa, A., Savino, G.L., Stellmacher, C., Schöning, J.: Understanding Users' Knowledge about the Privacy and Security of Browser Extensions. In: Seventeenth Symposium on Usable Privacy and Security (SOUPS 2021). pp. 99–118 (2021)
19. Kettle, J.: Skeleton Scribe: Sparse Bruteforce Addon Detection. https://www.skeletonscribe.net/2011/07/sparse-bruteforce-addon-scanner.html
20. Krzysztof Kotowicz: Intro to Chrome addons hacking: Fingerprinting. http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html
21. Laperdrix, P., Bielova, N., Baudry, B., Avoine, G.: Browser Fingerprinting: A Survey. ACM Transactions on the Web **14**(2), 8:1–8:33 (Apr 2020). https://doi.org/10.1145/3386040
22. Laperdrix, P., Rudametkin, W., Baudry, B.: Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification. In: 2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 98–108 (May 2015). https://doi.org/10.1109/SEAMS.2015.18
23. Laperdrix, P., Rudametkin, W., Baudry, B.: Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 878–894 (May 2016). https://doi.org/10.1109/SP.2016.57
24. Laperdrix, P., Starov, O., Chen, Q., Kapravelos, A., Nikiforakis, N.: Fingerprinting in Style: Detecting Browser Extensions via Injected Style Sheets. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2507–2524 (2021)
25. Li, T., Zheng, X., Shen, K., Han, X.: FPFlow: Detect and Prevent Browser Fingerprinting with Dynamic Taint Analysis. In: Lu, W., Zhang, Y., Wen, W., Yan, H., Li, C. (eds.) Cyber Security. pp. 51–67. Communications in Computer and Information Science, Springer Nature, Singapore (2022). https://doi.org/10.1007/978-981-16-9229-1_4

26. Lin, X., Ilia, P., Solanki, S., Polakis, J.: Phish in Sheep's Clothing: Exploring the Authentication Pitfalls of Browser Fingerprinting. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 1651–1668 (2022)
27. Liu, Z., Shrestha, P., Saxena, N.: Gummy Browsers: Targeted Browser Spoofing Against State-of-the-Art Fingerprinting Techniques. In: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security. pp. 147–169. Lecture Notes in Computer Science, Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-09234-3_8
28. Liverani, R.S., Freeman, N.: Abusing firefox extensions. Defcon17 (Aug 2009)
29. Louw, M.T., Jin, S.L., Venkatakrishnan, V.N.: Extensible web browser security. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **4579 LNCS**, 1–19 (2007). https://doi.org/10.1007/978-3-540-73614-1_1
30. Lyu, T., Liu, L., Zhu, F., Hu, S., Ye, R.: BEFP: An Extension Recognition System Based on Behavioral and Environmental Fingerprinting. Security and Communication Networks **2022**, e7896571 (Feb 2022). https://doi.org/10.1155/2022/7896571
31. Moad, D., Sihag, V., Choudhary, G., Duguma, D.G., You, I.: Fingerprint Defender: Defense Against Browser-Based User Tracking. In: You, I., Kim, H., Youn, T.Y., Palmieri, F., Kotenko, I. (eds.) Mobile Internet Security. pp. 236–247. Communications in Computer and Information Science, Springer Nature, Singapore (2022). https://doi.org/10.1007/978-981-16-9576-6_17
32. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In: 2013 IEEE Symposium on Security and Privacy. pp. 541–555 (May 2013). https://doi.org/10.1109/SP.2013.43
33. Pantelaios, N., Nikiforakis, N., Kapravelos, A.: You've Changed: Detecting Malicious Browser Extensions through their Update Deltas. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 477–491. ACM, Virtual Event USA (Oct 2020). https://doi.org/10.1145/3372297.3423343
34. Perrotta, R., Hao, F.: Botnet in the browser: Understanding threats caused by malicious browser extensions. IEEE Secur. Privacy **16**(4), 66–81 (Jul 2018). https://doi.org/10.1109/msp.2018.3111249
35. Picazo-Sanchez, P., Eriksson, B., Sabelfeld, A.: No Signal Left to Chance: Driving Browser Extension Analysis by Download Patterns. In: Proceedings of the 38th Annual Computer Security Applications Conference. pp. 896–910. ACSAC '22, Association for Computing Machinery, New York, NY, USA (Dec 2022). https://doi.org/10.1145/3564625.3567988
36. Picazo-Sanchez, P., Ortiz-Martin, L., Schneider, G., Sabelfeld, A.: Are chrome extensions compliant with the spirit of least privilege? International Journal of Information Security **21**(6), 1283–1297 (Dec 2022). https://doi.org/10.1007/s10207-022-00610-w
37. Preuveneers, D., Joosen, W.: SmartAuth: Dynamic context fingerprinting for continuous user authentication. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. pp. 2185–2191. SAC '15, Association for Computing Machinery, New York, NY, USA (Apr 2015). https://doi.org/10.1145/2695664.2695908
38. Rochet, F., Efthymiadis, K., Koeune, F., Pereira, O.: SWAT: Seamless Web Authentication Technology. In: The World Wide Web Conference. pp. 1579–1589. WWW '19, Association for Computing Machinery, New York, NY, USA (May 2019). https://doi.org/10.1145/3308558.3313637

39. Sam, J., Ancy Jenifer., J.: Mitigating the Security Risks of Browser Extensions. In: 2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS). pp. 1460–1465 (Jun 2023). https://doi.org/10.1109/ICSCSS57650.2023.10169483
40. Sanchez-Rola, I., Santos, I., Balzarotti, D.: Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 679–694 (2017)
41. Schaub, F., Marella, A., Kalvani, P., Ur, B., Pan, C., Forney, E., Cranor, L.F.: Watching Them Watching Me: Browser Extensions Impact on User Privacy Awareness and Concern. In: Proceedings 2016 Workshop on Usable Security. Internet Society, San Diego, CA (2016). https://doi.org/10.14722/usec.2016.23017
42. Sjösten, A., Acker, S.V., Sabelfeld, A.: Discovering Browser Extensions via Web Accessible Resources. CODASPY 2017 - Proceedings of the 7th ACM Conference on Data and Application Security and Privacy pp. 329–336 (Mar 2017). https://doi.org/10.1145/3029806.3029820
43. Sjosten, A., Van Acker, S., Picazo-Sanchez, P., Sabelfeld, A.: Latex Gloves: Protecting Browser Extensions from Probing and Revelation Attacks. In: Proceedings 2019 Network and Distributed System Security Symposium. Internet Society, San Diego, CA (2019). https://doi.org/10.14722/ndss.2019.23309
44. Solomos, K., Ilia, P., Karami, S., Nikiforakis, N., Polakis, J.: The Dangers of Human Touch: Fingerprinting Browser Extensions through User Actions. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 717–733 (2022)
45. Solomos, K., Ilia, P., Nikiforakis, N., Polakis, J.: Escaping the Confines of Time: Continuous Browser Extension Fingerprinting Through Ephemeral Modifications. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2675–2688. CCS '22, Association for Computing Machinery, New York, NY, USA (Nov 2022). https://doi.org/10.1145/3548606.3560576
46. Somé, D.F.: EmPoWeb: Empowering Web Applications with Browser Extensions. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 227–245 (May 2019). https://doi.org/10.1109/SP.2019.00058
47. Starov, O., Laperdrix, P., Kapravelos, A., Nikiforakis, N.: Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In: The World Wide Web Conference. pp. 3244–3250. WWW '19, Association for Computing Machinery, New York, NY, USA (May 2019). https://doi.org/10.1145/3308558.3313458
48. Starov, O., Nikiforakis, N.: XHOUND: Quantifying the Fingerprintability of Browser Extensions. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 941–956 (May 2017). https://doi.org/10.1109/SP.2017.18
49. Torres, C.F., Jonker, H., Mauw, S.: FP-Block: Usable Web Privacy by Controlling Browser Fingerprinting. In: Pernul, G., Y A Ryan, P., Weippl, E. (eds.) Computer Security – ESORICS 2015. pp. 3–19. Lecture Notes in Computer Science, Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-24177-7_1
50. Trickel, E., Starov, O., Kapravelos, A., Nikiforakis, N., Doupé, A.: Everyone is Different: Client-side Diversification for Defending Against Extension Fingerprinting. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 1679–1696 (2019)
51. Zhang, D., Zhang, J., Bu, Y., Chen, B., Sun, C., Wang, T.: A Survey of Browser Fingerprint Research and Application. Wireless Communications and Mobile Computing **2022**, Article ID 3363335 (Nov 2022). https://doi.org/10.1155/2022/3363335