Analysis of insecure configurations in Chrome Web Store extensions: Impact on user security and privacy

José R. S. Silva, Luiz H. B. A. Silva, Vitória B. A. Silva, Maria Gabriela Lima Damasceno, Caio B. de Souza, Andson Balieiro

[1]Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE),
50670-901 – Recife – PE, Brasil

{jrss,lhbas,vbas2, mgld, cbbs, amb4}@cin.ufpe.br

Abstract. *Browser extensions, popular for enhancing functionality, pose risks to user privacy and security. This study analyzes insecure configurations in Chrome Web Store extensions, assessing their functional justification. Auditing 287 extensions revealed that many employ excessive permissions or unnecessary access to sensitive data without clear justification. Findings show that risky practices persist even in widely used tools, highlighting flaws in app stores' verification processes. The research provides empirical evidence of these issues and advocates for stricter policies for developers, platforms, and regulators to balance functionality and security in the ecosystem.*

1. Introduction

The increasing adoption of the Internet for everyday activities such as communication and financial transactions has driven the use of browser extensions, which offer additional functionality such as ad blocking, password management, and protection against cyber threats [Eriksson et al. 2022]. Studies show that the majority of modern browser users have at least one extension installed, highlighting its importance in enhancing the browsing experience [Johnson et al.]. For example, the most popular browser for personal computers, with 66% of the market, has almost 125,000 extensions, totaling over 1.6 billion active users [Hsu et al. 2024].

Despite their benefits, browser extensions can also introduce significant risks to users' privacy and/or security. Such tools can incorporate vulnerabilities and malicious behaviors, such as injecting unwanted code or leaking sensitive information, compromising users' data [Sam and Jenifer 2023][Carlini et al. 2012][Eriksson et al. 2022]. One of the main factors contributing to these risks is the permissions model adopted by browsers such as Google Chrome and Mozilla Firefox. In these environments, extensions request permissions to access system resources or interact with web pages. However, many extensions request excessive permissions, such as the use of generic permissions of the type $///\star$, which allow unrestricted access to any page, without a clear functional justification [Johnson et al.]. Furthermore, users tend to trust extensions, granting permissions without fully understanding their implications, which increases the potential for abuse [Senapati et al. 2024].

Different studies have been conducted on the security risks and vulnerabilities of browser extensions. For example, analyzing 133,365 extensions available in the Google Chrome Web Store (CWS), [Eriksson et al. 2022] identifies three main types of attacks on

extensions (password theft, traffic redirection and cross-extension attacks) and suggests countermeasures that range from the concrete declaration of permission for the desired action (e.g. screenshot) and the adoption of new versions of the extensions' manifest file, indicating how they will handle user requests. [Picazo-Sanchez et al. 2022] uses machine learning and the download pattern of extensions in the CWS to identify malicious and suspicious extensions. The study by [Nguyen 2018] analyzes 50 Chrome extensions, identifying critical vulnerabilities such as privilege escalation and flaws in content *scripts*. Although the authors offer a valuable contribution, their analysis was conducted under the paradigm of Manifest V2 - a model whose security policies have been radically reformulated in the current Manifest V3. Furthermore, similar to [Eriksson et al. 2022] and [Picazo-Sanchez et al. 2022], [Nguyen 2018]'s investigation did not include a comprehensive assessment of permissions or essential configurations such as the *Content Security Policy* (CSP). These limitations, together with the restricted sample (50 extensions), reduce the generalizability of the results and highlight the urgent need for further studies.

Given this scenario, this work investigates potentially insecure configurations in Chrome Web Store extensions, analyzing aspects such as sensitive permissions, manifest version, and *content security policy* (CSP). An experimentation methodology is adopted, which combines empirical analysis of extensions with configuration auditing, to classify extensions as vulnerable when they present inadequate practices. The results indicate that a significant number of extensions use insecure configurations without plausible justification, reinforcing the need for improvements in the review processes of official browser stores. The study contributes to the literature by highlighting the importance of a careful review of configurations and permissions, aiming to protect users and guide regulators and developers in the implementation of more robust security policies [Sam and Jenifer 2023][Kim and Lee 2023]. This study can support the creation of stricter guidelines, ensuring greater protection for users of browser extensions.

The remainder of this paper is organized as follows. Section 2 presents some related studies. The methodology adopted in this study is described in Section 3. Results and analyses are addressed in Section 4. Finally, Section 5 concludes this paper and points out future research directions.

2. Related Work

Different studies have been conducted on the security risks and vulnerabilities of browser extensions. For example, the authors in [Eriksson et al. 2022] analyze 133365 extensions available in the Google Chrome Web Store (CWS) and identify three main types of attacks on extensions: password theft, traffic redirection, and cross-extension attacks. They suggest countermeasures such as specifically declaring permission for the desired action (e.g., screenshot capture, *captureVisibleTab*), adopting a new version of the extension manifest file that indicates how the extensions will handle user requests, and making it mandatory to define the externally connectable key (*externally_connectable key*) or adopting secure mode by default, respectively. [Picazo-Sanchez et al. 2022] combines machine learning, security analysis, and the pattern of the number of extension downloads in the CWS to identify malicious and suspicious extensions. The authors analyze the extension download history for 6 months, comprising 1212 active extensions, finding 135 groups and identifying 61 of them with at least 80% malicious extensions. Additionally, they show that when combined with code similarity analysis, this identification is improved.

Although the analysis of insecure configurations in browser extensions [Allauddin and Lokhande 2024] and the use of excessive permissions and bad development practices is little explored in the literature, these aspects can create attack scenarios. In this sense, the study [Kim and Lee 2023] presents an analysis of vulnerabilities in the extension architecture of modern browsers. The authors identify 59 vulnerabilities in 40 extensions, highlighting how failure to correctly implement the principle of least privilege can allow privilege escalation attacks, such as the execution of malicious *scripts* and the theft of sensitive data. To combat these weaknesses, the study proposes FISTBUMP, an extension architecture that, through process isolation and the use of the DOMProxy mechanism, protects content *scripts* by executing them in an environment isolated from the rendering processes. This approach mitigates identified risks and preserves compatibility with existing extensions, contributing to the advancement of security defenses in web environments.

Aimed at preventing attacks, the work of Sam and Jenifer [Sam and Jenifer 2023] proposes a framework that integrates static and dynamic analysis techniques to perform a continuous evaluation of installed extensions, enabling the real-time detection of malicious behaviors and vulnerabilities, such as XSS, CSRF, and other threats. The study details the implementation of a monitoring system and emphasizes safe development guidelines, highlighting the importance of educating users to adopt safer browsing practices. The experimental results demonstrate the effectiveness of the proposed method, contributing to improving online security by offering practical tools to identify and mitigate risks associated with extensions.

In their study, [Johnson et al.] helps us understand the impact of permissions on security by analyzing the weaknesses present in Google Chrome extensions. The authors show how different attack vectors, such as the use of background tabs, exploitation of iframes for stealth operations and the falsification of user data, can be used to access and manipulate sensitive information. They show that, even with the restrictions imposed by component isolation and privilege separation, extensions can pose a significant security risk, especially when used maliciously.

Although the cited works address important aspects such as the detection of malicious extensions through machine learning, behavior analysis, excessive permissions, and vulnerabilities in the architecture of extensions, few studies systematically investigate the vulnerabilities created by configurations present in extensions published in the Chrome Web Store. Unlike these approaches, this work focuses on the empirical analysis of configurations such as the use of sensitive permissions, outdated manifest versions, and permissive content security policies (CSP), proposing a detailed audit to classify potentially vulnerable extensions based on inappropriate practices. This perspective complements previous efforts by exploring a dimension that is still little studied, but essential for the security of the extension ecosystem.

3. Methodology

This study aims to investigate vulnerabilities present in web extensions that may allow a legitimate user, when using an apparently secure extension, to be targeted by attacks due to flaws in the extension itself. To achieve this objective, the research was structured in four main steps, as illustrated in Fig. 1, described below.

1. **Data Collection and Storage:** Collects and stores data on available extensions from the Chrome Web Store to build a comprehensive dataset of the extensions and their declared metadata.
2. **Filtering, Risk Classification, and Choosing Settings:** Filters the dataset to focus on high-impact extensions based on user count, establishes a preliminary risk model, and uses this classification to select the specific configurations for in-depth analysis.
3. **Detailed Manual Settings Usage Checks:** Conducts detailed manual checks on the selected configurations, providing a qualitative assessment to establish a framework for judging if a permission is justified by an extension's core functionality.
4. **Automated Static Permission Analysis:** Performs a static analysis of the extensions code to complement the manual approach, automatically identifying requested permissions that are unused.
5. **Automated Statistics Collections:** Aggregates the findings from all previous stages through the automated collection of statistics, providing a final quantitative overview of the study's results.

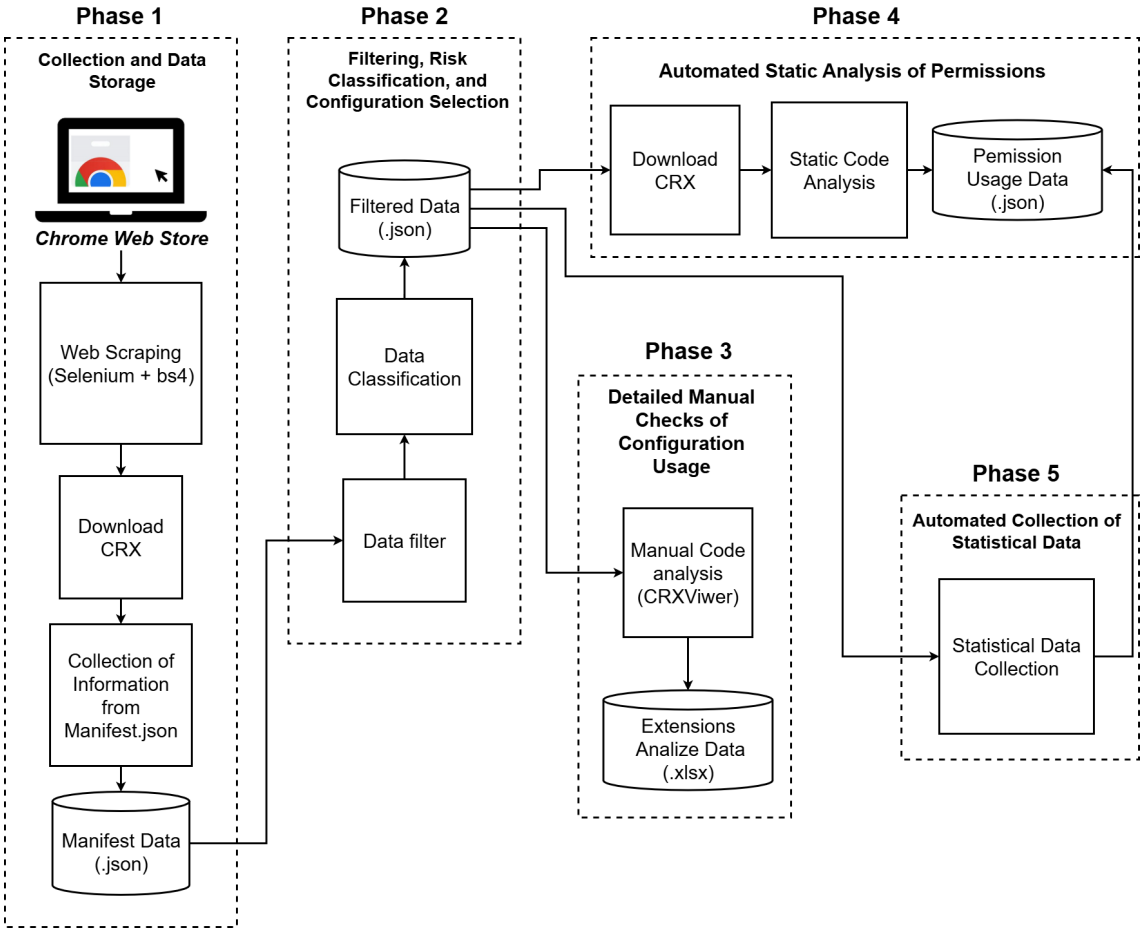

Figure 1. Methodology Flow

For the purpose of this study, a requested permission was defined as *unjustified* (also referred to as *inappropriate* or *excessive*) if it met one or both of the following evidence-based criteria:

1. **Functional Non-Essentiality:** The permission was determined to be non-essential for the extension's core functionality. This was established through a comparative analysis where alternative, popular extensions in the same category delivered the same primary benefit without requiring said permission, as detailed in our manual analysis framework (Section 3.3).
2. **Unused Declaration:** The permission was declared in the `manifest.json` file but our automated static analysis pipeline (Section 3.4) found no corresponding API calls within the extension's JavaScript source code, indicating it was requested but not used.

This two-pronged definition provides a systematic and replicable basis for identifying over-privileged extensions, grounding the concept of 'inappropriate use' in both functional context and code-level evidence.

3.1. Data Collection and Storage

Browser extensions are typically stored in vendor-managed repositories, where extension developers make them available for distribution. The Chrome Web Store (CWS) is the Chrome browser repository, managed by Google [Eriksson et al. 2022], and served as the source for our data collection. The goal was to analyze up to 1,500 most relevant extensions by the number of downloads in each of the 18 categories defined by the CWS, totaling 23,363 extensions.

Since the CWS does not provide a public Application Programming Interface (API), a web scraping methodology was adopted [Vila et al. 2017]. The data gathering process was performed in two main stages.

In the first stage, a script using the **Selenium Python framework** navigated through the CWS category pages. The **Beautiful Soup** (bs4) library then parsed the loaded HTML to extract a single piece of information for each listed extension: its unique **extension ID**.

In the second stage, the script iterated through the collected IDs to acquire the core data. For each ID, the corresponding `.crx` file was downloaded directly from Google's update services. Concurrently, **Selenium** accessed the specific details page of each extension to scrape its **user count**. After the download, the contents of the `.crx` file were analyzed locally to extract the `manifest.json` file, which defines its permissions and security policies. All collected data—the information from the `manifest.json`, the extension ID, and the user count—was then aggregated and stored in a single `.json` file for subsequent analysis.

The collected records contained attributes essential for analysis, some of the most relevant among them are:

- `manifest_version`: Version of the manifest, indicating the structure and security policies applied.
- `permissions`: List of permissions requested by the extension, including, for example, access to storage and, in some cases, cookie manipulation.
- `host_permissions`: Permissions associated with access to specific domains. Allows the extension to access certain URLs or websites, which can be an attack vector if the extension has excessive permissions.

- `content_scripts`: *Scripts* JavaScript that the extension injects into web pages. These *scripts* have access to the content of the pages and can manipulate the DOM. They can be used maliciously if the extension is vulnerable, such as in Cross-Site Scripting (XSS) attacks.
- `content_security_policy`: Security policy that defines where resources can be loaded from, such as *scripts*, images, and other content. Helps prevent malicious code execution by restricting the origin of resources loaded by the extension.
- `externally_connectable`: Setting that determines which external origins can communicate with the extension. This can allow external websites to interact with the extension and, if misconfigured, can be exploited to perform attacks such as Cross-Site Request Forgery (CSRF).
- `storage`: Defines the extension's use of storage, such as data saved locally in the browser. This can include user preferences or temporary data. If not properly secured, it can be exploited to store sensitive data such as credentials or browsing information.

3.2. Filtering, Risk Classification and Choosing Settings

After collecting the data, an automated Python script was developed to filter the extensions stored in the `.json` file. This made it possible to apply filters based on a minimum number of downloads and the values of the `manifest.json` fields, using the `AND` logic to combine the selected criteria. This approach ensured efficient and targeted filtering for the analysis of potentially vulnerable extensions.

Since the Chrome Web Store has a very high number of extensions, a filter was applied that considered only those with the number of downloads equal to or greater than 500,000, resulting in 287 extensions. This criterion was adopted to prioritize extensions of greater relevance, since widely used extensions tend to have a greater impact on users' security and privacy.

A script was developed to classify the risk of the analyzed extensions. This classification was based on a careful analysis of several factors, including the requested permissions, the version of `manifest.json`, and other critical aspects such as security policies, content *scripts*, and external communication settings. The goal was to categorize the extensions into three risk levels: **high**, **medium**, and **low**, as described below:

- **High Risk**:
  - Extensions that request excessive permissions, such as access to `<all_urls>`, `cookies`, `tabs`, `webRequest`, or `debugger`, without a clear justification for their functionality.
  - Use of `content_scripts` that inject *scripts* into web pages without proper sanitization, which can facilitate Cross-Site Scripting (XSS) attacks.
  - Insecure `externally_connectable` configurations, which allow communication with untrusted external sources.
  - Missing or misconfigured `content_security_policy`, allowing loading of resources from insecure sources.
  - Using `manifest_version 2` (older and less secure) instead of version 3, which introduces stricter security policies.

- **Medium Risk**:
  - Extensions that request moderate permissions, such as `activeTab`, `storage`, or `notifications`, but can still pose risks when misconfigured.
  - Use of `content_scripts` with limited access to specific pages, but without full input sanitization.
  - `content_security_policy` settings that allow `unsafe-inline` or `unsafe-eval`, increasing the attack surface.
  - Use of `host_permissions` to access specific domains, but without clear justification for needing access to those domains.
- **Low Risk**:
  - Extensions that request only the minimum permissions necessary for their functionality, such as `alarms` or `idle`.
  - Use of `manifest_version 3`, which offers greater security and control over permissions and policies.
  - Robust `content_security_policy` settings, restricting resource loading to trusted sources and avoiding `unsafe-inline` and `unsafe-eval`.
  - Absence of `content_scripts` or safe use of these *scripts*, with adequate sanitization of inputs and outputs.
  - Restrictive `externally_connectable` settings, allowing communication only with trusted sources.

The classification obtained in step 2 helped in deciding which criteria (configurations) of the extensions would be used to evaluate them in step 3. The choice was based on a cost-benefit, where the cost was the difficulty and time to evaluate a certain configuration or permission, and the benefit was the level of risk of this configuration or permission, with the aim of prioritizing the greatest risk according to the risk classification. Therefore, the chosen configurations were:

- **manifest_version (MV)**: This setting, the manifest version of a web extension, determines its permissions, *scripts*, and behaviors. Outdated versions may lack newer security protections, such as the remote *scripts* blocking introduced in Google Chrome Manifest V3, making extensions vulnerable to attacks, including malicious code execution and abuse of excessive permissions. Currently, the latest version is Manifest V3. There is no technical justification for maintaining older versions, as this may expose users to previously documented vulnerabilities.
- **content_security_policy (CSP)**: This setting, the content security policy (CSP), defines which external content sources are allowed to load on a web page, restricting the execution of unauthorized *scripts*, styles, and other elements. During the analysis, overly permissive CSPs were identified, including those that allow JavaScript to be executed *inline* and the use of the **eval** command. These settings are especially critical as they can facilitate Cross-Site Scripting (XSS) attacks. Any extension that contained the following directives in the CSP was classified as vulnerable: **unsafe-inline**; **unsafe-eval**; **wasm-unsafe-eval**. There is no plausible technical justification for these settings, as there are safer alternatives for handling JavaScript code that do not cause any functional drawbacks.

- **content_scripts (CS)**: This setting defines the domains into which the extension can inject JavaScript code. *scripts* injection can be exploited by malicious extensions or by legitimate extensions that have vulnerabilities, allowing malicious code to be executed on the pages visited by the user. Extensions that configure **content_scripts** in a permissive manner, without justifiable need, have been classified as insecure.
- **identity (ID)**: This permission allows the extension to provide authentication functionality via OAuth, using the user's Google account. However, some extensions may request this permission unnecessarily, collecting data without any real need.
- **identity.email (IDM)**: This permission grants access to the user's email address. Many extensions use it inappropriately, without this functionality being essential for their operation, representing a risk to privacy.
- **externally_connectable (EC)**: This configuration allows web pages to communicate directly with the extension through APIs. If the extension's API has vulnerabilities, a malicious website could exploit this functionality to perform inappropriate actions in the user's browser.
- **history (HI)**: This permission grants access to the user's browsing history, which may compromise their privacy by exposing behavior patterns and personal preferences.
- **cookies (CO)**: This permission allows the extension to read and manipulate browser cookies, not limited to the extension's own cookies. This functionality can be exploited to track and collect data about user behavior, posing a risk to their privacy. Many extensions request this permission without real need, aiming to obtain valuable information that can be commercially exploited [Somé 2019][Malgieri and Custers 2017].

3.3. Manual Analysis of Permission Justification

Following the risk classification, a qualitative analysis was performed by the authors on the 287 filtered extensions. The objective of this manual stage was to establish and apply a systematic framework for judging whether the high-risk configurations and permissions, identified in Section 3.2, were functionally justified. To ensure objectivity, the following operational definitions and procedures were used:

1. **Defining "Essential" and "Functioning":** A permission or configuration was considered *essential* only if it was indispensable for delivering the extension's core advertised functionality. To test this, we employed a comparative analysis approach. For a given extension, we identified other popular extensions within the same functional category (e.g., ad blockers, note-taking apps). If a significant number of market-leading alternatives could provide the same core function *without* a specific permission, that permission was flagged as potentially *non-essential*. An extension was considered "functioning" if it successfully delivered its primary promised benefit.
2. **Gauging "Benefit" and "Proportionality":** The *benefit* was defined as the core functionality offered to the user. The *proportionality* of a permission was then evaluated by weighing this benefit against the privacy risk posed by the permission (as classified in Section 3.2). For example, a high-risk permission like

`history` (access to Browse history) was considered disproportionate for an extension whose primary benefit was merely changing a website's color scheme, as the risk to user privacy far outweighed the offered functionality.

3. **Defining "Insecure Setting" and "Safer Approach":** An *insecure setting* refers directly to the configurations identified as high or medium risk in Section 3.2. This includes, but is not limited to, the use of Manifest V2, a Content Security Policy (CSP) containing directives like `unsafe-inline` or `unsafe-eval`, or overly broad host permissions. A *safer approach* is the corresponding modern best practice, such as migrating to Manifest V3 or refactoring code to eliminate the need for insecure directives. A "functional impairment" was considered to have occurred only if adopting the safer alternative demonstrably broke the extension's core functionality, which was verified during our comparative analysis.

This manual analysis provided the qualitative context necessary to interpret the results of the large-scale automated analysis that followed, ensuring that claims of "inappropriate use" were grounded in both functional and security-based evidence.

3.4. Automated Static Permission Analysis

To identify excessive permissions, we implemented a static analysis pipeline through *scripts* in Python comprising four main components:

1. **Download and Extraction** (*script*:):
   - Obtaining the `.crx` files of the extensions selected in the filtering step, using their unique IDs.
   - Extraction of JavaScript files for analysis, preserving the original project structure.

2. **Pattern Detection**: We have developed three complementary strategies to identify permission usage:
   - **Canonical Standards**: Regular expressions based on the Chrome API, such as direct access via `chrome.<permission>` and bracket syntax like `chrome['permission']`
     .
   - **Dynamic Calls**: Indirect access detection via:
     ```
     API = 'tabs';
     chrome[API].query(...);
     ```
   - **Script Injection**: Identification of indirect uses such as `activeTab` through: `chrome.scripting.executeScript(...)`

3. **Empirical Validation**:
   To ensure the accuracy of the static analysis, we manually sampled 20% of the extensions (57 out of 287) with a dual purpose: to calibrate the initial patterns and to identify extreme cases (false negatives/positives). From these cases, we iteratively refined the search patterns in the static code analysis.

4. **Consolidation**: Generate structured JSON reports and integrate with manifest data for cross-analysis.

3.5. Automated Statistics Collections

In this last stage of the methodology, some *scripts* were created in Python, with the objective of automatically collecting various information about the use of configurations of the 287 extensions. The main objective was to verify which permissions were being requested from the user without being used in the extension code, that is, permissions that were being requested without being used by the extension. This reveals a lack of care in handling user privacy, in addition to the fact that future updates of the extensions could make some subsequent use of these permissions.

4. Results

4.1. Data Visualization

Table 1 shows the number of occurrences of inappropriate configurations by category.

Table 1. Number of settings classified as inappropriate by setting category

Configuration/Permission	Inappropriate	Total
content_scripts permissive	14	67
cookies	39	50
content_security_policy permissive	28	28
manifest_version	17	17
identity	9	14
identity.email	4	5
history	1	3
externally_connectable permissive	1	1

Table 2 shows the number of extensions by risk category, considering the 287 extensions analyzed. This categorization was based on the risk classification performed in the second stage of the methodology of this work. The difference is that only the configurations selected in the second stage of the methodology will be considered to perform the risk classification.

Table 2. Number of extensions by risk category (limited)

Risk level	Number of extensions
High	64
Medium	17
Low	206

Table 3 reveals a ranking of the permissions requested in the 287 extensions analyzed, while Table 4 shows the same ranking considering only the extensions that contain some inappropriate configuration.

The results indicate that even among the most relevant extensions, there are problems with inappropriate configurations. Of the 287 extensions analyzed in step 3 of the methodology, which considers only some specific configurations previously mentioned, 87 ($\sim 30.31\%$ of the total) presented inappropriate configurations. This result indicates that a significant portion of the extensions may be collecting unnecessary data from users

Table 3. Ranking of the 10 most requested permissions

Permission	Number of requests
storage	246
scripting	130
tabs	114
activeTab	83
contextMenus	78
alarms	76
webRequest	76
unlimitedStorage	59
declarativeNetRequest	54
cookies	50

Table 4. Ranking of the 10 most requested permissions in extensions with inadequate configuration

Permission	Number of requests
storage	78
cookies	41
tabs	41
scripting	38
alarms	35
webRequest	31
activeTab	28
contextMenus	28
unlimitedStorage	25
declarativeNetRequest	19

or exposing them to security risks, increasing vulnerability to possible cyberattacks, or generating privacy issues.

Table 2 uses the risk classification algorithm created in step 2 of the methodology, with the limitation of considering only the configurations selected in step 2 itself (which were also used in step 3). This result revealed that problematic extensions are generally high risk, most often due to the unjustified use of cookies, being present in approximately 44.82% of extensions with inadequate configuration.The undue interest in extensions in cookies is probably motivated by the commercial value that the information contained in cookies has [Somé 2019][Malgieri and Custers 2017]. The analyses revealed that permissions that can be used to collect user data, and consequently generate a financial return, were very present. For example, it was observed that the permissions **cookies**, **identity**, **identity.email** and **history**, which have commercial value [Somé 2019][Malgieri and Custers 2017] as they allow access to user information, were problematically present in 50 ($\sim 57.47\%$ of the total) of the 87 extensions with inappropriate configuration. Furthermore, it is possible to identify a relationship between the use of the **cookies** permission and the probability of an extension containing some inappropriate configuration, for example, when comparing the tables 3 and 4, it is possible to see that the **cookies** permission becomes much more present if we consider only the exten-

sions with inappropriate configuration, when compared to the total number of extensions analyzed, while the remaining configurations remain in similar positions in the ranking of both tables.

In Table 1, it is possible to verify that there are some configurations that were classified as inappropriate in all uses, some of them had this result because the occasions of real need were very rare [Perrotta and Hao 2017][Google Chrome Developers 2023b], while others, such as the use of outdated versions of the manifest, completely lack justification in any case. These findings highlight the urgent need to implement automatic filtering mechanisms, both in the Chrome Web Store and at the user's browser level, to block extensions with certain types of configurations, since some of these configurations are never recommended [Google Chrome Developers 2023a][Pantelaios and Kapravelos 2024].

Another observation arising from the results is that 36.84% of the 287 extensions analyzed request at least one permission that is not used by the extension, indicating a practice of excessively requesting permissions without functional justification. This problem is also evident when we observe that 29 requested permissions were not used in any of the extensions studied, and that the chance of an extension requesting a permission and not using it is 31.13%. These findings demonstrate the lack of strict control over permission requests, which can pose significant risks. Extensions can be updated later, and previously granted permissions, but not initially used, could be activated for malicious purposes without requiring new user authorization.

5. Conclusion

This article addressed the issue of security risks and vulnerabilities in web browser extensions. In it, a methodology was developed for analyzing browser extensions and applied considering the extensions available in the Chrome Web Store in order to identify the use of configurations that may compromise user security and/or privacy. The results revealed that approximately 30% of the analyzed extensions had at least one inappropriate configuration, which could cause harm to user privacy and/or security. These results demonstrate that, even in widely used extensions, negligent development practices or lack of oversight in official stores can expose users to significant risks. The results highlight the need for two main actions: prioritizing security in the development cycle and rigorous review by official extension stores. In addition to providing support for security professionals and developers, this work contributes to the literature by mapping a problem that is still little explored. This lack of specific research on insecure configurations in extensions reinforces the relevance of this analysis and serves as a basis for future investigations, such as the creation of automated auditing tools or the expansion of the study to other stores, such as Firefox Add-ons. A future research direction is the investigation of cyberattacks that can exploit the insecure configurations identified in this study and the analysis of the performance of these attacks through the configurations.

Acknowledgment

This study was funded by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Code 001 and by the UFPE/Propesqi via Edital Nº 06/2024.

References

Allauddin, M. S. and Lokhande, P. S. (2024). Analyzing security risks in browser extension search tools: A literature review. *SSRN Electronic Journal*.

Carlini, N., Felt, A. P., and Wagner, D. (2012). An evaluation of the google chrome extension security architecture. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 97–111.

Eriksson, B., Picazo-Sanchez, P., and Sabelfeld, A. (2022). Hardening the security analysis of browser extensions. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, page 1694–1703, New York, NY, USA. Association for Computing Machinery.

Google Chrome Developers (2023a). Manifest v3 migration guide. Acesso em: 10 jul. 2024.

Google Chrome Developers (2023b). Match patterns - chrome developers. Acesso em: 10 jul. 2024.

Hsu, S., Tran, M., and Fass, A. (2024). What is in the chrome web store? In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '24, page 785–798, New York, NY, USA. Association for Computing Machinery.

Johnson, C. A., Paramiswaran, S., and Mailewa, A. B. Discovering vulnerabilities in web browser extensions contained by google chrome.

Kim, Y. M. and Lee, B. (2023). Extending a hand to attackers: browser privilege escalation attacks via extensions. In *32nd usenix security symposium (usenix security 23)*, pages 7055–7071.

Malgieri, G. and Custers, B. (2017). Pricing privacy – the right to know the value of your personal data. *Leiden Law School Legal Studies Research Paper Series*.

Nguyen, C. O. Y. Z. R. (2018). Analysis of vulnerabilities of web browser extensions. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 116–119. IEEE.

Pantelaios, N. and Kapravelos, A. (2024). Manifest v3 unveiled: Navigating the new era of browser extensions. *ArXiv*, abs/2404.08310.

Perrotta, R. and Hao, F. (2017). Botnet in the browser: Understanding threats caused by malicious browser extensions. *IEEE Security Privacy*, 16:66–81.

Picazo-Sanchez, P., Eriksson, B., and Sabelfeld, A. (2022). No signal left to chance: Driving browser extension analysis by download patterns. In *Proceedings of the 38th Annual Computer Security Applications Conference*, ACSAC '22, page 896–910, New York, NY, USA. Association for Computing Machinery.

Sam, J. and Jenifer, J. A. (2023). Mitigating the security risks of browser extensions. In *2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, pages 1460–1465. IEEE.

Senapati, B., Naeem, A. B., Khan, T. A., Golder, S. S., Das, S., Mondal, S., Mishra, L. N., and Patra, S. (2024). A study on web user's attitude and knowledge towards data security and privacy issues of web browser extensions. In *2024 4th International*

Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), pages 1–8. IEEE.

Somé, D. F. (2019). Empoweb: Empowering web applications with browser extensions. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 227–245.

Vila, E., Novakova, G., and Todorova, D. (2017). Automation testing framework for web applications with selenium webdriver: Opportunities and threats. In *Proceedings of the International Conference on Advances in Image Processing*, pages 144–150.