Age of Extensions: Modern Browser Helpers & User Privacy

Shubham Agarwal
Saarland University, Saarbrücken
Email: s8shagar@stud.uni-saarland.de

***Abstract*—The research community has continually studied and reported the abuse of browser extensions across different vendors as a security threat leveraged by malicious developers to invade the security and privacy of user data over time. One of the key exploitation reasons observed among these abuse vectors is the discrepancy in the initial vetting process employed by the respective extension hosting platforms to accurately identify malicious extensions and the potential security risks associated with them.**

This paper takes into account the work done by the Google security researchers on Chrome Extensions as lead study and further compares and contrasts the related studies done so far by other researchers in this domain. We discuss the evolution of the extension ecosystem, the detection mechanism for malicious extensions in different browser settings, and their implications. We also compare and contrast the selected features studied by researchers such as extension architecture, research methodologies, attack surfaces, and the potential security threats posed to the user to draw a general understanding of current research status.

We observe a significant difference between the threat models, their detection methodology, and defensive tools proposed for the analysis of extensions and further outline the precision, accuracy, and performance of the framework, and challenges faced by each of them in a real-time environment.

1. Introduction

Browser Extensions, also known as Add-Ons or Browser Helper Objects in different operational settings, have become an integral part of modern browsing architecture. All major browser vendors like Chrome, Firefox, Microsoft Edge and Safari have widely adopted the use of extension within their system. Extensions provide additional functionality to enhance the browsing experience of the user such as improving the appearance of web pages, integration to popular services, adding extra functionality to browsers, and privacy preservation.

Extensions significantly differ from the native web applications running on the browser based on their capability to perform certain operations. Unlike web applications, Same-Origin Policy (SOP) does not apply to them and thus, it can perform cross-origin operations with no intervention. It can also fetch resources from third-party websites, inject data into visited web pages or just extract information from them. Similarly, it can intercept all network requests as well as issue fresh ones to arbitrary domains with no restriction. The developer of the extension defines these capabilities as part of the permissions model. Some extensions intend to provide functionalities with respect to certain hosts or visited pages, while others intend to provide services on all of them.

A central repository hosts all the extensions submitted by the developers for distribution usually controlled by the browser vendor themselves. For instance, Chrome Web Store hosts all the extensions that could be installed on Chrome browser while Firefox Add-ons repository all the extensions for Firefox. Extensions often mediate security-sensitive user information by storing or sending over the network as per its defined capabilities. Previous large-scale research studies on extensions and the Web Stores have indicated a significant number of extensions to pose a potential threat to user privacy and security [1], [14], [15]. Spying on user browsing history, injecting advertisements, illegitimate modifications to web page content and script injections, hijacking social media accounts and tampering with security headers in network responses are some of the widely observed malicious behavior reported by researchers so far.

We consider the large-scale analysis on the Chrome Extensions submitted to Chrome Web Store from 2012 – 2015 studied by Jagpal et al. in [3] as our lead study. We further identify a few related follow-up studies across different browsers and other necessary background studies [2], [4], [9]. We discuss the approaches implemented in these works to analyze, detect and defend against malicious browser extensions.

To summarize, the key research contributions to this paper are as follows:

- We discuss the difference in extension architecture between Firefox & Chrome and briefly outline a class of attack specific to the architecture – Extension Reuse Vulnerability.
- We discuss the research methodologies adopted by different researchers over time to detect malicious browser extensions targeting a broad range as well as specific threats.

- We then focus on the dataset selected for study and the outcomes of their analysis framework by comparing their precision, accuracy, and performance.

- We outline a general trend of how the extension abuse ecosystem has evolved based on findings in the research works conducted over time and summarize them for future work.

2. Extension Architecture

Developers commonly implement browser extensions in JavaScript, CSS & HTML, similar to generic web applications. However, the overall file and directory structure usually differs based on the browser. In general, an extension contains three files needed for its functionality – the manifest file, the content script, and the event pages. It may contain background pages optionally.

The manifest file holds extension specific details such as permissions, hosts and developer details. The content script contains the core logic of the intended functionality. This content script interacts with the visited web pages and performs desired operations. Event pages, optionally required, act as a graphical user interface for the extension. Background scripts aid core functionality by operating in the background for the entire browser session irrespective of the visited hosts.

Chrome enforces Lockdown policy on all the extensions which restricts the browser to install only those extensions hosted on Chrome Web Store [1]. Other browser vendors have implemented similar policies as well. Each extension submitted to the central repository is subject to an initial vetting process where the system scans them to detect any known malware signatures or abnormal behavior based on its implementation. The initial vetting process is implemented by all major browser vendors that allow extensions. The extensions are only available to users when they have passed the vetting process successfully.

2.1. Chrome Extension Architecture

Prior to the work of Barth et al. in 2010, the separation between the processing boundaries of extensions installed on the browser ceased to exist. No central authority existed to scrutinize the submitted extensions and identify over-privileged extensions. However, Chrome updated its extension architecture based on the design proposed by Barth et al. in [1]. The current architecture of Chrome enforces an extensive permission system as per the Principle of Least Privilege over all the extensions. Moreover, the browser allows each extension to operate in an isolated process and no two extensions could communicate to each other directly, similar to an application running in a virtual environment. The system also ensures that an extension installed on the browser does not use privileges given to other installed extensions in the browser.

The extension architecture deployed by other browsers such as Safari, as mentioned in [9], resembles the Chrome ecosystem.

2.2. Firefox Extension Architecture

As described by previous studies on Firefox extensions in [4], [11], the extension architecture is typically conventional as compared to the current architecture in Chrome. It employs an XPCOM-based extension system that exposes feature-rich yet security-sensitive APIs to the extension also used by the browser itself, such as access to file systems and command execution in the browser's context. Furthermore, the extension ecosystem is such that all the extensions installed on the system share common global JavaScript namespace and run in the same process the browser runs in. This implies that unlike Chrome, the Firefox ecosystem does not isolate the execution of extensions and is devoid of other security principles such as minimal permission models. The shared JavaScript namespace leads to an unwanted modification of the values stored in the variables or the behavior of global functions since objects declared with the same name in more than one extension would result in conflict, an age-old recognized problem known as a namespace pollution problem.

Firefox hardened the security boundary by implementing security policies like minimal permission system in the architecture by introducing *Jetpack*, a new extension framework for the development of extension. It contained a manifest file to store the metadata of an extension to display it to the user. However, this framework remained unpopular among developers due to the lower number of APIs available to them. A further hardened framework for extension development – *WebExtensionsAPI* allowed even less number of security-sensitive APIs available for development.

2.3. Extension-Reuse Vulnerability

This vulnerability is unique to browser extensions in Firefox where the attacker tries to utilize the permissions of other benign extensions installed in the browser to initiate malicious actions. The key factor to this exploit is the fact that all the extensions installed on the browser share the same global JavaScript namespace as mentioned earlier. In this case, it is comparatively easy for a malicious developer to bypass the vetting procedure by developing a seemingly benign extension with minimum permissions requested to pass the process successfully. Further, this attacker can analyze and identify all the global variables, functions and objects defined in the JavaScript file available on the repository for any extensions available for users.

This capability leak allows an attacker to launch a classic confused-deputy attack in which the malicious extension alters JavaScript objects or variables in the global namespace illegitimately, which other benign extensions access to carry

security-critical functionalities ahead. In this way, malicious extensions without requiring any additional permission leverage the use of permissions assigned to other benign extensions to carry out malicious activities. Phishing, remote code execution, issuing illegitimate network requests and drive-by-download are a few potential threats caused by this class of vulnerability.

3. Research Methodologies

In the lead study performed by Jagpal et al. in [3], the authors employed an amalgam of static and dynamic analysis mechanism to generate a comprehensive view of the intended as well as actual interactions between the extensions and other entities such as the browser, web pages, and network. The static analytical method also tried to detect the correlation between obfuscated and non-obfuscated code and draw conclusions on the basis of distance for known malicious strains. Dynamic analysis, performed along with specially crafted behavioral suites, recorded the activities of extensions on the basis of certain common events. The developer details provided additional insights into the relationship between a malicious extension and its developer. The ML-based regression model used for the scoring of extensions provided an automated platform to determine the behavior of extensions based on the generated report, independent of the human expert verification process required. The tool, known as WebEval, aimed to aid the existing vetting process of the Chrome Web Store so that malicious extensions can be detected proactively before its release to the public. We observe that Thomas et al. utilized WebEval's capabilities with no modifications to detect ad-injection at client-side by inspecting DOM elements [6].

The authors in [2], performed a pioneering study on large-scale analysis of Chrome extensions to detect malicious behavior based on dynamic analysis by developing a tool called Hulk. They used HoneyPages and Event-Handing Fuzzers to trigger and log activities by hooking additional loggers based on the presence of certain DOM elements, API calls, network activity or event occurrences to understand their functionality. However, we observe that the framework contained only a subset of features and signals considered for detection, as compared to the lead study, thus limiting the functional space of operation. The manual labeling of the extensions analyzed by Hulk, not guided by an automated scoring system, required constant human experts. Hulk depended entirely on the network interactions observed during the analysis phase and is thus susceptible to the cloaking strategies and noise created by malicious extensions and there is no mechanism for Hulk to adapt to new threats.

Buyukkayhan et al, analyzed Firefox extensions to detect Extension-Reuse Vulnerability – a specific threat which bypasses the vetting procedure in the Firefox ecosystem [4]. The authors proposed CrossFire, a static analysis tool developed to detect this specific issue by identifying the global variables and functions and the data flow to the security-critical APIs as source and sinks. It analyzed each extension statically by building a function call graph and determining data flow to the sinks. To confirm this threat, the tool generated concept exploit or exploit templates used to determine the existence of the vulnerability in extension. This work focused on a single class of attack instead of studying the broad attack surfaces. This approach lacked a clear distinction between benign and malicious extensions using identified sources and sinks. The concept of exploit generation for decision-making helped draw conclusive evidence for decision-making. This work suggests that a seemingly benign extension with no security-critical permission requested could also pose a security threat to the user and bypass the initial vetting procedure successfully, similar to the suggestions provided by authors in [2] and [3].

The authors in [5], developed a tool, called OriginTracer, to detect ad-injecting extensions through the provenance-based flow tracking mechanism. It worked by annotating the web page contents – both static and dynamic, with labels indicating the parent of that resource. Based on these labels, the tool compared the content of the web page on load with the one after extension finishes its operation and identify any changes in the provenance of the content. Instead of making a decision by themselves or automating it, the authors proposed to allow users to make decisions based on the displayed information on the screen about the injected content. The authors also highlighted the tool's capacity as an in-browser detector and further extension to an ad-blocking system. This work outlines the significance of provenance-based tracking to detect content manipulation in web pages. However, it also posed a question about the security and usability of the approach that the user is given to decide on the fate of extensions.

Another threat specific study performed by Aggarwal et al. focused on malicious detection spying on user sensitive information in the browser [7]. In this study, the authors first tried to identify spying behavior among extensions using the ML-based regression model deployed on WebEval and reported its shortcomings [3]. They further employed the Recurrent Neural Network (RNN) to identify the difference in access-pattern of security-sensitive APIs among benign and malicious extensions. This new RNN based model proved to provide significantly accurate results when compared to the outcomes of the previous model. We observe that while ML-based models provide significant results when studying a broad range of attack surfaces, other models such as RNN may incur better insights into specific threat models. We also infer that threat-specific studies provide better results than a broad range of studies.

Weissbacher et al. worked on the detection of history-leaking browser extensions by relying on dynamic analysis mechanisms in [9]. The proposed tool, Ex-Ray, intercepted

and analyzed network calls that were initiated by browser extensions to leak browsing and other sensitive data from the user to the attacker or third-party websites. It comprised a complex combination of unsupervised, supervised and triage-based learning models where the supervised and unsupervised model together analyzed each extension. They manually vetted the findings of the unsupervised model further and used as training data for supervised learning. We observe that this approach significantly helps to adapt to new threats in the wild and requires minimal human intervention after a certain interval when the supervised model is optimally trained. However, the signals detected for malicious behavior in the dynamic analysis are still susceptible to logic and time bombs.

From the above discussion, we observe that there exists some discrepancy with each of the methodology adopted in these works discussed. For instance, an attacker could leverage the dynamic analysis framework as padding oracle to further strengthen its evasive techniques without affecting its malicious functionality. Whereas, static analysis alone does not elicit an environment to observe the activities based on events and certain contents present in the visited web page. However, when adopted in combination, they provide significantly better results. Human experts tend to be an integral part of giving a final verdict on the behavior of extensions. Generic issues such as obfuscation in static analysis and coverage and cloaking in dynamic analysis still remain a challenge faced by the researchers.

4. Evaluating Results

The lead study considered in this work performed an extensive study on approximately 100K extensions submitted to the Chrome Web Store from January 2012 to 2015. The underlying detection tool, WebEval, analyzed all of the above extensions with a median detection latency of 25 minutes from the time of submission to the store. It flagged 9.4% of all the analyzed extensions as malicious based on different features used to identify malicious behavior. It precisely determined the extension as malicious 73.7% of the time, which further improved to 98% in the final phase of the study. It also showed a high recall of 93%, as advocated by the authors, that they preferred recall over precision to safeguard against false negatives and recognition of new threats. The signals observed in the dataset, such as code distance to that of known malicious extensions, antivirus scan results, developer reputation, requested permissions, API calls, DOM manipulations, and other known malicious logic influenced the decision-making process majorly. The top 10 permission requests identified by WebEval during analysis are listed in Table 1 [3]. We observe that the top permissions discovered by WebEval have strong similarities with the findings of Hulk in [2]. `tabs` is the most commonly requested permission, followed by `webRequest`. Other security-sensitive permissions include `storage`, `cookies`, etc.

Requested Permission	Precision	Recall
tabs	12%	84%
webRequest	23%	39%
webRequestBlocking	22%	27%
notifications	14%	27%
contextMenus	15%	26%
storage	9%	25%
webNavigation	21%	19%
cookies	10%	14%
unlimitedStorage	14%	13%
idle	27%	10%

TABLE 1. TOP 10 PERMISSIONS REQUESTED IN EXTENSION MANIFEST

Human experts spent an average of 2.75 minutes analyzing over 10K extensions. The obtained results provided an overview of the extension ecosystem at that time and outline the threats associated with it, as discussed in the next section. The author emphasized the fact that even a small number of false negatives may affect a large user base and the role of human experts in the system to avoid false positives.

Hulk, a dynamic analysis framework, analyzed approximately 48K Chrome extensions available in the Chrome Web Store and a handful of them obtained from a third-party repository, Anubis [2]. It identified 130 extensions of them as malicious, while 4,712 as suspicious based on the signals obtained. Features such as extension management capabilities, remote content inclusion, API calls, and network events influenced the decision-making process. The permissions failed to prove as a contributing factor to arrive at a certain decision due to the similarity in usage among both benign as well as malicious entities. The lead study observed and reported similar remarks on permissions.

The top permissions and API calls observed in this study compared to the lead study showed significant similarity and usage shifts over time in extensions. It also provided insights into potential inadvertent inconsistencies in the use of wildcards while declaring the hosts. Due to the limited nature of feature detection such as multi-step querying and lack of data flow analysis in the runtime environment, this study provided a lower bound of all the existing malicious extensions in the wild. The author also highlighted the inadequacy of this approach against robust cloaking strategies and network errors caused.

The security framework of CrossFire, on the other hand, analyzed top 10 Firefox extensions and a random sample of 323 obtained from top 2,000 extensions, according to the standard theory of confidence interval, available in Firefox ecosystem for Extension-Reuse vulnerability [4]. It specifically looked for global JavaScript entities by constructing the Abstract Syntax Tree (AST) of global objects and further

analysis of the data flow to security-critical APIs to identify potential security risks among extensions in a two-stage process that took an average time of 10 minutes. The results indicated that 9 out of the top 10 extensions were vulnerable to the class of attack, while the other 323 extensions reported a total of 351 vulnerabilities among them. An additional scan of the top 2000 extensions reported a total of 4,462 potential vulnerabilities. The author, however, emphasized the high false positive rate of 27.3%, affecting the overall efficiency of the system. The difference in false positive extensions and vulnerabilities is significant and we observe that the identified sources and sinks are widely used by the developers. The key exploits reported by the authors in this study are represented in Figure 1 below, taken from [4]. We observe that around two-thirds of extensions include network access vulnerability, followed by file I/O, event listeners, preference access and code execution.

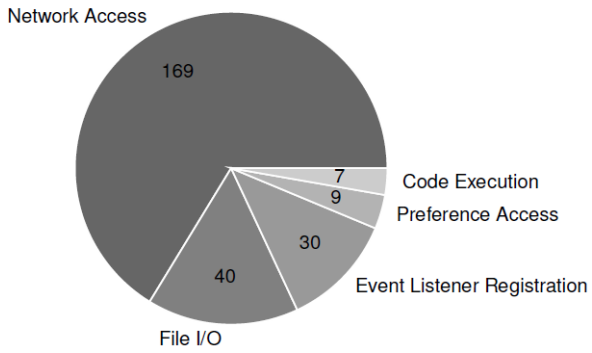

Figure 1. Breakdown of true positive vulnerabilities discovered by CROSSFIRE by category.

Based on the nature of the attack, it is impossible to scan against each extension against all the hosted extensions, which makes the precise detection of malicious extensions difficult. Furthermore, the authors developed an extension intended to perform this attack in a real-time environment and submitted it for vetting. We infer from the fact that it passed the initial scan without any warnings, the vetting process is inconsistent across all browsers as suggested by the previously discussed in [2] and [3].

Arshad et al., in their work related to the detection of ad-injecting extensions, developed a web provenance information tracking framework, OriginTracer. The outcome of this study is based on the user study performed with six known ad-injecting extensions. They observed user interaction and response to websites in two stages, first noting the behavior in the unmodified environment and then with the modified browser configured with the developed detection framework. To evaluate the performance overhead of this approach for in-browser detection, the authors ran a crawler over top 1K Alexa websites and noted the difference in elapsed time to load the websites by the unmodified and modified browser which confirmed a total of 10.5% of overhead.

The results obtained from this study showed that users responded to visually provenance indicators and acted on the basis of information provided to them with respect to the content injected while they observed almost no such behavior in the absence of such indicators. We infer from this study that there exist benign cases where the content injection is an intended functionality and cases where the advertisements by the extensions fetched from the publisher source itself. These challenges could be incorporated into future studies to detect this vulnerability.

The security framework developed by Aggarwal et al. in [7] to detect spying extensions based on Recurrent Neural Network analyzed extensions in multiple stages for accurate training and measurements. The authors selected over 43K extensions spread across 12 categories from the Chrome Web Store as ground truth for their detection model. The initial evaluation phase yielded 374 extensions to be potentially suspicious, expanding to over 1,200 extensions by reiterating the process over the trained model. The overall study confirmed the presence of 218 malicious extensions present in the Web Store spying on user data. This study reported a high precision and recall of 90.02% and 93.31%, respectively. We observe that this framework results in comparatively few numbers of false positives and false negatives discussed so far. But even a single extension that remains undetected may lead to significant repercussions, as suggested by Jagpal et al. in [3].

We observe certain limitations mentioned in this study which hinders its in-browser integration. The large computational overhead that it requires at the client-side may pose performance issues in the browser. Moreover, this study only detected information theft based on detection of Chrome API calls, while Web API calls remained undetected. This approach relied on the specific sequential pattern observed in spying extensions. However, we observe that it doesn't apply to all other malicious usages reported so far. The basic requirements of the model such as the minimum length of the sequence and the intended noise generated by the malicious developer in the sequence of API calls are some of the challenges and may be considered in the future scope of the study.

In the above-discussed studies, the results suggested that the malicious extensions are able to evade the Web Store security policies with ease and thus, are made available to users. Further, we observe that some of the extensions detected to elicit malicious behavior hold the user base in the figure of millions. Thus, even a single malicious extension with seemingly benign functionality could target a larger audience and compromise their security. We agree with the authors' recommendations in the lead study that proactive steps are needed in this domain so that an extension is not made available to users without a comprehensive security scan. The vetting procedure can be made efficient by incorporating the security design of the works discussed in his paper. However, this also means that it may also affect the agreement between the developers and the browser vendors to avail their extensions in the market.

5. Trends in Evolution of Malicious Extension

We have so far discussed the large scale study and their detection framework and the results obtained from their implementation to find malicious extensions hosted in the central repositories of browsers. We now discuss over a few of the most widely observed threats reported by the researchers in the order of their findings and how they are leveraged by these malicious entities in the browser.

We start our discussion with the threats observed by Kapravelos et al. based on the findings of Hulk [2]. The authors highlighted four major threats that existed at the time of the study – Ad Manipulation, Affiliate Fraud, Information Theft, and Online Social Network Abuse. Malicious developers used extensions as a common abuse vector to reap monetary benefits by means of the above-mentioned attacks. The authors observed ad replacements on the visited web page as common abuse prevalent at the time of the study. They found affiliate fraud as another means of generating income. The extensions tampered with the URL of the e-commerce domains and replaced identifiers used to distribute the commission to reviewers. Thus, an attacker could generate money without submitting any reviews. Extensions also showed their presence in user information theft, such as stealing passwords, session tokens, authentication cookies, and form data by means of keyloggers, reading web pages or intercepting network data.

The authors also reported that these extensions leveraged the authenticated session tokens of social media platforms stored in the browser and thus posted attacker-desired posts, photos, links, etc. to the account illegitimately. They suggested a few generic solutions to these issues such as restricting extensions to modify security-related headers like CSP in HTTP Responses. Furthermore, an extension should not be allowed to include remotely hosted content on the visited page as it violates the SOP policy. Other critical functionalities such as keylogging APIs should not be available for access to the extensions.

The lead study also emphasized understanding the intent of the malicious extensions detected by their security framework. They reported that the intent of the detected extensions strikingly differed from the traditional malware findings, such as the Banking Trojans and RATs. They mentioned that exploitation through browser extensions is driven by fast-paced monetary benefits sought by malicious developers, in contrast to the traditional malware. We observe that the findings on the nature of the threat are similar to the work done by Kapravelos et al. in [2]. They reported Facebook Hijacking and Ad injection as the most commonly observed threats caused by extensions at that time, among all the threats reported so far. The authors also mentioned that the extensions were involved in crypto jacking and tampering with bitcoin wallets. Search history leakage and user tracking also showed evident existence in their study. We observe that these entities target the social networks and advertisement domain primarily since it is easier to generate monetary benefits in a short time.

We observe from the threat-specific studies on ad-injection and spying extensions performed by researchers. As discussed in Section 3 and 4, these threats have consistently grown over time, along with an increase in the number of malicious extensions exploiting specific attack surfaces. The research community reported that extensions have evolved as the most commonly abused vector for user fingerprinting and history leakage over time. We have already discussed an attack specific to Firefox – Extension – Reuse Vulnerability. Though the JavaScript Namespace Pollution is a long-recognized issue, the extensions still contain a large number of global objects and thus, susceptible to such threats.

Perotta & Hao studied the abuse of the extension ecosystem as botnets who communicate to their central server via C&C channel. On the basis of privileges defined to the extension, the authors defined a theoretical view of all potential attacks possible. They provided a comprehensive list of attacks based on capabilities as well as resources. It also presented a novel class of attack reported never before – Certificate Exception Attack. In Firefox, a user can define custom exceptions in case of certificate errors by writing in *cert_override.txt* and storing it in the browser directory. This extension leverages this feature by overwriting the file and thus, aiming to downgrade the HTTPS connection and compromise the security. Remote Code Execution and Password Manager Attack are other such reported issues in this work.

Pan & Mao worked to understand and detect the browser extension which provides an attack surface for attackers to exploit DOM-based Client Side Scripting on a web page visited by users in [11]. They reported that since the SOP does not restrict the functionality of extensions and it can include remotely hosted scripts and execute in the context of the visited web page, these extensions may pose a threat to the security of the user. Another recent study by Somè D. F on the interaction between the web application and the extensions with respect to the cross-platform extension system – WebExtensions API [12]. Although not exactly related, this study highlights the capabilities of an extension ecosystem that can be leveraged by other web applications to leak browser data sensitive to user security.

6. Conclusion

In this review work, we discussed numerous studies done over time to detect and defend against malicious browser extensions across different browsers, on both broad as well as threat specific attacker models. We outlined the results obtained in previous work. We discussed the role of execution environment and privileges given by the browsers to the extensions with respect to the architecture in Firefox and Chrome Ecosystem and assessed its impact. We compared and analyzed the precision, accuracy, and

performance of different methods to detect the malicious behavior in extensions as per their threat model with respect to the lead study. Further, we discussed the evolutionary trends of malicious behavior in extensions and the benefits obtained from these exploits by the attacker.

We understand that certain factors like over-privileges, access to security-critical APIs and content tampering/injection pose a direct threat to the privacy of user data. Browser-specific attacks like Extension-Reuse Vulnerability may affect numerous users and may remain undetected without a specific analytical mechanism. The current vetting procedure employed by browsers does not guarantee full-proof detection and thus, it is not efficient enough to distinguish between benign and malicious use-cases of permissions requested by them. The key takeaway from this work is that no detection framework is robust against large-scale detection of malicious entities in the wild, but can significantly give better results when combined together. We conclude with a view that extensions should be approached with proactive scanning rather than reactive takedown in case of a potential threat to the security of users.

References

[1] Barth, A., Felt, A. P., Saxena, P., & Boodman, A. Protecting Browsers from Extension Vulnerabilities. In *Proceedings of the Network and Distributed System Security (NDSS)*, 2010.

[2] Kapravelos, A., Grier, C., Chachra, N., Kruegel, C., Vigna, G., & Paxson, V. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *USENIX Security Symposium* (pp. 641-654), 2014.

[3] Jagpal, N., Dingle, E., Gravel, J. P., Mavrommatis, P., Provos, N., Rajab, M. A., & Thomas, K. Trends and Lessons from Three Years Fighting Malicious Extensions. In *USENIX Security Symposium* (pp. 579-593), 2015.

[4] Buyukkayhan, A. S., Onarlioglu, K., Robertson, W. K., & Kirda, E. CrossFire: An Analysis of Firefox Extension-Reuse Vulnerabilities. In *Proceedings of the Network and Distributed System Security (NDSS)*, 2016.

[5] Arshad, S., Kharraz, A., & Robertson, W. Identifying extension-based ad injection via fine-grained web content provenance. In *International Symposium on Research in Attacks, Intrusions, and Defenses* (pp. 415-436). Springer, Cham, 2016.

[6] Thomas, K., et al. Ad injection at scale: Assessing deceptive advertisement modifications. In *Security and Privacy (SP), 2015 IEEE Symposium* (pp. 151-167). IEEE, 2015.

[7] Aggarwal, A., Viswanath, B., Zhang, L., Kumar, S., Shah, A., & Kumaraguru, P. I spy with my little eye: Analysis and detection of spying browser extensions. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)* (pp. 47-61). IEEE, 2018.

[8] 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)*. USENIX Association, Vancouver, BC, 2017.

[9] Weissbacher, M., Mariconti, E., Suarez-Tangil, G., Stringhini, G., Robertson, W., & Kirda, E. Ex-ray: Detection of history-leaking browser extensions. In *Proceedings of the 33rd Annual Computer Security Applications Conference* (pp. 590-602). ACM, 2017.

[10] Perrotta, R., & Hao, F. Botnet in the Browser: Understanding Threats Caused by Malicious Browser Extensions. In *IEEE Security and Privacy, 16(4)*, (pp. 66-81), IEEE, 2018.

[11] Pan, J., & Mao, X. Detecting DOM-Sourced Cross-Site Scripting in Browser Extensions. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference* (pp. 24-34). IEEE, 2017.

[12] Somè D.F. EmPoWeb: Empowering Web Applications with Browser Extensions. In *2019 IEEE Symposium on Security and Privacy (S&P)*, San Fransisco, CA, US, (pp. 917-935), IEEE, 2019.

[13] Bandhakavi, S., King, S. T., Madhusudan, P., & Winslett, M. VEX: Vetting Browser Extensions for Security Vulnerabilities. In *USENIX Security Symposium* (Vol. 10, pp. 339-354), 2010.

[14] Wang, J., Li, X., Liu, X., Dong, X., Wang, J., Liang, Z., & Feng, Z. An empirical study of dangerous behaviors in firefox extensions. In *International Conference on Information Security* (pp. 188-203). Springer, Berlin, Heidelberg, 2012.