Extension Vetting: Haven't We Solved This Problem Yet?

Dénes Bán
Brave Software
dban@brave.com

Benjamin Livshits
Brave Software and Imperial College London
ben@brave.com

***Abstract*—Browser extensions are almost omnipresent in today's web ecosystem, but their extended capabilities can and do often lead to abuse. Since manual inspection cannot keep pace, plenty of research is focused on automated extension vetting. These efforts, however, mostly target the detection of features that are heuristics, rather than guaranteed predictors of malicious intent.**

In this work we survey the features of common detection methods along with both malicious and benign examples of their usage to showcase the shortcomings of the state of the art. We also advocate for promising, but — so far — neglected approaches, and discuss possible future directions.

I. Introduction

Browser extensions are almost omnipresent in today's web ecosystem. This is understandable, as they do a good job of adding custom functionality to the base browsing experience. With hundreds of thousands to choose from, and numerous ones with tens of millions of users, browser extensions are likely here to stay. However, for extensions to be able to provide useful functionality, they generally need escalated privileges, which in turn can be abused. Blog posts documenting abusive behavior on the part of extensions appear with increasing frequency [32], [17], [3]. This is disheartening in that the problem is widespread, but also encouraging in that these flaws at least get some attention.

Browser vendors try to defend against malicious activity with tight(er) security models and closely inspected centralized repositories, but sneaking past these checks is apparently not impossible. The most common extension security models are aimed at defending users from "benign but buggy" extensions and have no adequate protections against intentional attacks. The most reliable way for rigorously checking an extension is expert manual inspection, and that requires skill — malicious code is rarely out "in the open" and is often accompanied by legitimate functionality, masking its presence.

The obvious solution involves automation, but that first requires a stricter formalization of what malicious browser extensions look or behave like. This is where we often get stuck. There is plenty of research analyzing extensions from many different perspectives, and, although most of them involve complex inspection, none seem to be able to bridge the gap between being able to automatically detect some features and being able to classify extensions as malicious or benign. Each feature that is extracted — be it information leakage, remote script injection, header tampering, etc. — is closer to a heuristic than to a definitive signal of abuse. There might be cases where the intuition behind these heuristics and detectors is accurate, and these cases might even be in the majority, but there will be inevitably benign extensions that are flagged, and also malicious ones that manage to slip through.

In this paper we concentrate on the current state of the art: what features are generally extracted from extensions, and how each of those can appear in both malicious and benign extensions. Our case studies are artifacts of manually checking the accuracy of a work-in-progress extension vetting methodology; they serve as proof that more and better features — or a significantly different mindset — are needed for correct results.

We also discuss a set of approaches that could nudge the status quo in the favor of the defender, if explored more in depth. These include correlation with natural language descriptions, checking deviations from the average for certain metrics, and focusing more on incremental version-to-version changes rather than examining extensions in isolation. Finally, we propose moving away from thinking in terms of pre-computed extension *blacklists* and more towards educating and involving users in keeping the browser extension ecosystem healthy.

A. Contributions

1) **Comprehensive review** of the current state of the art of malicious extension detection techniques and the features they use.
2) **Indicative case studies** demonstrating the strengths and weaknesses of existing approaches.
3) **Proposals** for future directions in malicious browser extension research.

B. Paper organization

The rest of the paper is organized as follows. Section II provides some background on threat models and current

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2019
24 February 2019, San Diego, CA, USA
ISBN 1-891562-60-6
https://dx.doi.org/10.14722/madweb.2019.23xxx
www.ndss-symposium.org

research efforts related to browser extensions. Section III showcases manual case studies with examples of both malicious and benign use cases for each feature. Section IV summarizes our vision for the future of extension vetting, before we conclude in Section V.

II. Related Work

To understand the implications of the features discussed in Section III — i.e., how and why they are extracted and how they could be misinterpreted — we start with an overview of related research efforts.

A. General Threat Models

Privacy leaks: The most widely-researched threat extensions can pose is the leakage of private, often personally identifiable information (PII) [8], [2], [25], [33], [23], [16], [18]. This category contains the diffusion of visited domains or complete browsing history, form data — including e-mails and passwords — search queries, other installed extensions, or even keyboard and mouse events. These categories of data can be combined with other, more innocent information, like browser agents, installed fonts, etc. to facilitate user tracking and fingerprinting — either by actively performing it [16], or helping other parties perform it [26]. There are extensions, though, where this diffusion is warranted; a topic that will recur throughout this paper.

Another attack vector is social media abuse where access to the user's account is compromised. Many of these cases can also fall under PII leakage if they happen separately, through exfiltrated access tokens or username-password pairs, but there is research about behaviorally detectable cases as well [10].

Integrity violations and tampering: Yet another threat is header tampering, which might make the visited sites be able to perform actions they were not intended to perform [15], [18]. This could include cross-site scripting attacks (XSS) or disregarding content security policies (CSP) set by the server. Additional techniques include privilege abuse [18] — e.g., the extension attempts to prevent the user from uninstalling it — and privilege escalation [6] — multiple extensions working together through some channel to achieve something that none of them would be capable of on their own.

Online advertisement is also a lucrative target for malicious extensions [34], [16], [29], either by injecting new ads into visited sites or just replacing existing ones with fraudulent ones for profit. This causes not only user frustration but also lost revenue for publishers and possibly even malware installations.

Botnets could lead to even more serious consequences than anything discussed so far, and browser extensions provide a prime opportunity for botnet authors to access the privileged execution context they require [24], [21].

Browser-specific attacks: We note that there are a number of earlier studies specifically aimed at Internet Explorer's Browser Helper Objects [20] or Firefox's previous extension architecture [28], [4], [19], [31]. While the technical details of these papers are not directly applicable to subsequent sections, they raise the same kinds of issues and suggest similar approaches.

Additionally, combined discussions and special focus on multiple threat vectors — along with concrete attack implementation and defenses — can be found in [6] and [21].

B. Current Detection Approaches

For this study we are mostly concerned with the Chrome-like web extension architecture, which can arguably be considered the new de facto standard. However, while its security model [5] is accepted by most browsers, it is primarily designed to defend against malicious sites gaining escalated privileges through buggy extensions and not against malicious extensions themselves.

Combined approaches: The most all-encompassing methodology to extension vetting is Google's own WebEval infrastructure [16] put in place to protect the Chrome Web Store (CWS). WebEval involves both static and dynamic analyses for manifest permissions, obfuscation markers, file signatures and directory structures, DOM modifications, Chrome API calls, network traffic, etc., in combination with manual rules and expert opinions. Millions of raw features are used in the daily training of logistic regression models that learn on manually labeled datasets to detect malicious extensions. Nevertheless, the fact that malware regularly makes it to the CWS means there is room to improve either the features or the exercising of the extensions (or both). Aggarwal et al. [2] take a similar route, only with fewer features but more machine learning techniques, and also replacing single Chrome API calls with call sequences.

Hulk [18] is another complex malicious extension detection technique that focuses on dynamic script injection, bad (404 or non-existent domain) network requests, obfuscation through `eval`s, uninstall abuses, header tampering, and information theft. Emphasis is placed on eliciting the malicious behavior by fake "visits" to millions of domains and HoneyPages that try to adapt the DOM to the extension's expectations.

Targeted approaches: A somewhat direct successor to Hulk is the Mystique tool [8] using generally the same eliciting while adding taint tracking to concentrate on data leakage withing the extension's code. If data coming from a sensitive source reaches a taint sink — i.e., a point where it might potentially leave the confines of the user's browser — it is flagged as leaking.

Others take a more external approach to data leakage and analyze the generated traffic patterns instead. Starov and Nikiforakis [25] look for both accidental and intentional leakage of browsing history, search queries, form data, and even other installed extensions by inspecting outgoing payloads with an additional attempt at deobfuscation — repurposing their tooling from [26]. On the other hand, Weissbacher et al. [33] operate under

the assumption that URL leaking can only happen when visiting longer URLs leads to larger traffic payloads. They use the preliminary dataset originating from this check to build supervised models that can predict the same leaking behavior, only through Chrome API traces.

The Expector tool [34] focuses on ad injection, and especially on malvertising (i.e., ads that lead to malware). Their detection method is checking for DOM changes that introduce 3rd-party-referencing elements, and seeing if triggering a click on those elements leads to redirections before reaching a landing page. Another ad injection detector using DOM manipulations combined with manual post-processing of the offending injector libraries — and build on top of the WebEval framework mentioned above — is presented in [29].

Lastly, DeKoven et al. [10] describe an approach more similar to a generic antivirus mentality, i.e., removing infected extensions based on a central repository of definitions. They rely on Facebook's existing infrastructure to detect affected users through negative feedback, content spikes, and DOM-based indicators, and then scan those user's extensions for blacklisted URLs, domains, hashes, e-mail addresses, etc. using static analysis. Malware information propagation then happens with the help of ThreatExchange and VirusTotal.

The Missing Link: The overall problem from an end-to-end perspective is that all of the research mentioned above focuses on clearly delineated, objectively inspectable features that are not *necessarily* connected to maliciousness. Even if the research explicitly aims for just detecting a given feature and nothing more, the presence in itself is not that useful unless it definitely implies something "bad" — or it is shared for further decision making, as we will discuss in Section IV. Nearly every observed behavior has both malicious and benign use cases, which leads to false classifications: we could ban good extensions and — even worse — allow bad ones. We conclude that no one approach has enough features yet for a purely model-driven, automatic method to be able to take intentional limitations or contextual assumptions out of the picture.

III. Case Studies

In this section we present a set of features commonly used in extension analysis and argue that none of them are definitive signals of malicious intent by themselves. We do this by providing both malicious and benign examples for each item we encountered during our extensive manual case studies.

Referenced URLs and domains: It is definitely important what new connections to remote domains an extension aims to establish. A relatively easy — but noisy — method for collecting these domains is by examining the extension's manifest and source code through static analysis. Metrics derived from this check can be the number of distinct eTLD+1 domains, or where they are considered "bad" for some definition of bad. As an initial effort, for example, we used blocked [7] or blacklisted [11] domains, while domain-scoring through `alexa.com`, Web of Trust, `checkpagerank.net`, and similar ranking lists, or using anti-malware checks are also often utilized.

True positives for this feature include cases where the higher number of filtered domains mean potentially unwanted affiliate relationships as encountered in the SwagButton[1] extension. In contrast, false positives could include instances like DuckDuckGo Privacy Essentials[2] which just try to block these sites themselves.

There are also outliers in overall domain references which could be suspicious, but the ones we looked at had reasonable explanations for this upon closer inspection. E.g., Bloomberg[3] enumerates information (including URLs) for thousands of companies, Send to Kindle (by Klip.me)[4] hardcodes integration with hundreds of sites, and FullScreen for GoogleMaps[5] declares content script injection into all possible `google.*` eTLD+1 domains.

Third-party domain accesses: This category goes one step further than the above section and instead of just the implication, contact is actually established to a new third party introduced by the extension. Dynamic analysis approaches often look for differences in network traffic between default browsers, and browsers with the evaluated extension installed. This still does not necessarily mean outgoing data (discussed below), but incoming data, commonly the dynamic loading of remote JSON responses or scripts.

There is a big difference regarding the quality of the response, however, between deciding to query a trading API for actual quotes on shares like E*TRADE Browser Trading[6] and what Flash Video Downloader[7] (since removed from the CWS) tried to do by loading a script full of tracking, search-overriding and advertisement-related functionality.

Additionally, even if we could *perfectly* distinguish between malicious and benign remote scripts, we would still have to dynamically recheck extensions exhibiting this feature continuously since time or remote state-based triggers could change the extension's behavior without any apparent change to its codebase.

Data leakage: If we turn our attention to outgoing data, traffic length, content, and even in-code taint tracking is used to see if private data leaves the user's computer. Additionally, adhering to the aforementioned remote trigger warning, simply storing personal information for later access is often treated as potential leakage. Malicious examples discovered from this category [1] are probably the ones treated most seriously and are highly visible due to their security risks being easily understandable even to non-technical users.

However, there are many instances where "leaking" is necessary for the extension's purpose, e.g., shopping

[1]SwagButton: gngocbkfmikdgphklgmmehbjjlfgdemm
[2]DuckDuckGo: bkdgflcldnnnapblkhphbgpggdiikppg
[3]Bloomberg: llgiblikeclfoebojkplbcmnicgcabhg
[4]Send to Kindle: ipkfnchcgalnafehpglfbommidgmalan
[5]FullScreen: kapgobifldgnkpcgoejmkfoemkajilcj
[6]E*TRADE: adgjomjdnhlppcidahijhehhfgneaolh
[7]Flash Video Downloader: babnbebdlabbfcmekadnndbppmkhflog

assistants like Avast SafePrice[8] that need at least some information about the visited page to be able to present deals and coupons to the user. Sadly, confirming the presence of a leak with increasing precision does not inevitably lead to the detection of malicious intent.

Also, there is a grey area where extra data leakage might not be that closely related to what the extension is doing, but the fact is clearly stated in official privacy policies and users are presented with an option to opt out, like in the case of Pinterest Button[9]. How a fully automated system could make a decision about this remains an open question.

Access to sensitive data: Another similar feature is trying to catch sensitive data as it is first accessed instead of following its path or discovering it in traffic. This could be done using DOM-instrumentation and issuing warnings on access or modification for fields deemed private, like `<input type="password">`. Malicious cases that would trigger this check are applicable from the previous section [1] but benign usages provide an opportunity to mention password managers like LastPass[10] where it's essential to the extension's user-serving functionality, and is backed by a reputable company with verified privacy practices.

Trackers and fingerprinting: Tracking the user across the internet is yet another kind of data leakage where the targets are the user's aggregate interests and behavioral model. Checking for this is usually performed through observing the usage of known tracking libraries during runtime. A clearly offending example of this is LINER - Web/PDF Highlighter[11] that injects analytics scripts into every visited page regardless of usage. However, the situation can often be much more nuanced, as evidenced by Wikibuy[12] and Panda[13].

If we were to just look at the presence of tracking scripts, both would classify as "bad." If we were to refine the inspection by where those tracking scripts are used, we could see that neither inject it directly into the page's contents, only using it within their background pages — which is no guarantee, but one step better. Looking at how these tracking activities get triggered paints a different picture again: Panda uses its tracking to see how its users interact with its interface (single page usage statistics), while Wikibuy logs shopping site visits (possible multi-page tracking) with additional low-level, fingerprint-enabling data that should be avoidable.

Obfuscation: Unlike script minification, which is not only still allowed, but even encouraged by Google [30], detecting obfuscation is much harder. The most common heuristic for obfuscated code is detecting the use of `eval`, possibly constrained by input length filters or expanded by also detecting other, "implied" ways of executing text as code. Searching for base64 strings, or structural recognition through machine learning could make separating obfuscation "types" more precise, but would still remain a heuristic.

Although obfuscation is often used by malicious extensions to hide their functionality [22], our preliminary study on `eval`s showed that there are many benign uses. For example, outliers like Remarq[14] relied on a specific webpack bundling configuration that put each item inside an `eval` call. Spot checks in the middle range showed either unnecessary but ultimately unharmful uses like in the case of FlyView[15] or deliberate uses to run scripts in specific contexts like Virtru Email Protection[16]. Lastly, a few `eval` usages could appear by simply including jQuery, InboxSDK, or other similarly popular libraries in the extension bundle, which clearly does not indicate malicious intent in isolation.

Ad injection: Existing work mostly identifies ad injecting extensions by comparing a pristine DOM tree to one where the extension has had a chance to manifest its behavior. This is often combined with filtering for introduced 3rd party links, link following, checking for redirects, or manually identifying ad-injecting libraries and checking for their presence. These features reveal malicious examples like Web Developer 0.4.9 described in [17], for example, but also lead to misclassifications. One interesting extension we came across was SEOquake[17], which injects a Search Engine Optimization (SEO) header bar with third-party linking into pages. Clicking a field causes non-ad-network related redirects before reaching `www.semrush.com`, so it is falsely considered as adware by this specific detection method.

Header tampering: Existing work [18] treats any kind of manipulation of a request's response headers — detected by comparing the original response with what actually reaches the site requesting it through the `webRequest` API — as malicious. However, during our manual checks we encountered Black Menu for Google[18], which indeed modifies headers, but only to be able to embed original Google content in the `iframe`s of the extension's popup, without any apparent malicious intent. We also note that extensions like Requestly[19], whose explicit goal is header tampering for testing purposes, are possible false positives that need further differentiation.

Search or new tab overrides: Detecting an extension's intent to override the default search engine or the deafult new tab page is straightforward using its manifest. The non-trivial part comes when trying to decide whether those overrides are abused in any way. To focus on search engines, the whole practice would be unreasonable to denounce as there are valid uses like — to repeat a benign example — DuckDuckGo Privacy Essentials[20] that pro-

[8] Avast SafePrice: eofcbnmajmjmplflapaojjnihcjkigck
[9] Pinterest Button: gpdjojdkbbmdfjfahjcgigfpmkopogic
[10] LastPass: hdokiejnpimakedhajhdlcegeplioahd
[11] LINER: bmhcbmnbenmcecpmpepghooflbehcack
[12] Wikibuy: nenlahapcbofgnanklpelkaejcehkggg
[13] Panda: haafibkemckmbknhfkiiniobjpgkebko
[14] Remarq: gbjoeicgnnclannmimohheehbbegpfec
[15] FlyView: blmfdjjomajmejdkdbcabhgfonkhfaam
[16] Virtru: nemmanchfojaehgkbgcfmdiidbopakpp
[17] SEOquake: akdgnmcogleenhbclghghlkkdndkjdjc
[18] Black Menu: eignhdfgaldabilaaegmdfbajngjmoke
[19] Requestly: mdnleldcmiljblolnjhpnblkcekpdkpa
[20] DuckDuckGo: bkdgflcldnnnapblkhphbgpggdiikppg

vides its own search next to other privacy related features. On the other hand, blindly allowing such modifications enables extensions like MusicNet Home[21] to hijack all search queries. While its CWS description references `yahoo.com` as the new search engine, it first takes a detour to `eanswers.com` (which then doesn't even show up in the history due to the nature of the redirection) before finally landing on an actual Yahoo results page. This is a clear example of no added value with a possibly malicious side effect emphasizing, again, that deeper inspection is needed to separate similar cases.

Available metadata: Some proposals consider the metadata available on the Chrome Web Store itself when making classification decisions (e.g., author, user count, average rating). However, this metadata clearly cannot be sufficient to determine malicious intent — a low rating could mean the extensions does something "bad", but could also mean it's just not doing its stated purpose well enough. Additionally, statistical analysis from [2] suggests that none of user base, rating count, average rating, or developer statistics differ significantly between manually identified spying and non-spying extensions. Even comment text didn't contain references to users being aware of any unwanted behavior, although that keyword-based search approach is susceptible to the same issues raised for conceptual filters below.

Ownership changes: Changes in who owns or maintains an extension is also not a reliable indicator. Malicious examples include Particle for YouTube [9] where threat actors purchased access to the extension and almost immediately followed it up with adware modifications to infect the existing user base. On the other end of the spectrum are more straightforward cases, e.g., Invite All Friends on Facebook[22] that changed its displayed author from "Mohammed N. El-Madhoun" to "Jack Cooper" in early October of 2018 but hasn't had a significant (or malicious) modification since.

Conceptual filters: In an initial effort to filter adblocking extensions, we tried — among other — the phrase `ad block`. It resulted in numerous valid matches, but it missed, e.g., Tunnello VPN[23] where part of the description was phrased "...prevent the tracking and targeted advertising...". It also, comically, introduced Crazy Rider[24], a motorcycle riding game where the description contained "Causion: road block, stairs etc". Admittedly, this is less about actual malicious/benign comparison and more an example of simple false positives and false negatives, but we think it is demonstrative of the kinds of mishaps that could happen during binary approaches without sufficient context, even for much more complex detection methods.

Other inherently mistakable signals: While other signals such as permissions requested, DOM modifications, or Chrome API calls — either in isolation or as groups/sequences — are heavily utilized, they inherently lack the potential for identifying definite malicious intent. Each permission and API endpoint was put in place to represent a valid, benign use case and pairing common usage patterns to a higher chance of malware still leaves scenarios where those patterns appear in "goodware" just by accident.

While none of the features mentioned in Section III might be decisive in itself, we are certainly not advocating against them. They might still be very useful in combined models — as they have already been, and as they will undoubtedly be in the future. However, these models should consider including (or relying more heavily on) additional features, some of which are discussed in Section IV.

A note on more "unequivocal" signals: There are signals that are harder to misinterpret than others, like accessing a known malicious domain, downloading a file whose hash is already classified as malware, or preventing access to the extension management page to prevent its own uninstall. Being able to automatically detect these is great because they are more black-and-white (mostly black) than most other features and there is practically no chance at misclassification. Nevertheless, these represent only a subset of available extensions, and filtering these does not address the bigger issue of keeping the whole extensions ecosystem healthy.

IV. Where Do We Go From Here?

This section is devoted to a few perspectives that lack sufficient attention. These ideas and techniques are not new, by any measure; they just have not been explored as thoroughly yet in relation to extension vetting as we think they would deserve. We acknowledge that some of them might suffer from ambiguity similarly to the features we have covered so far but they would still help in reaching a more complete understanding. Additionally, there are some that advocate for a different mindset altogether, starting with user involvement.

User involvement: Considering the points and shortcomings raised in previous sections, we strongly believe that the solution lies with not *only* better detection strategies, but with a more fine-grained user interaction and educating the user to be a more emphatic part in guarding against malicious extensions. Sadly, this view is somewhat conflicting with both how things are done now, and with the direction current research efforts aim for.

On one hand, most browsers defer to user decision upon install, but the information presented is minimal, based mostly on manifest permission declarations only. So while the act of involving the user to at least some degree is commendable, even an expert would not be able to make an informed decision in this situation and would have to resort to further analysis and checks. On the other hand, there is a wealth of research concerned with these additional checks which, despite the drawbacks discussed, could provide users with much more to go by. Here, however, the information is used to construct blacklists (or whitelists depending on the feature under examination) and is generally confined to the researchers.

[21] MusicNet Home: ilnanmefnphbfjefajmciblllkdecepd
[22] Invite All Friends: inmmhkeajgflmokoaaoadgkhhmibjbpj
[23] Tunnello VPN: hoapmlpnmpaehilehggglehfdlnoegck
[24] Crazy Rider: lfgcmpnnailedfapmafbigfifabfamcl

Our opinion is that these two, seemingly separate worlds should be fused together: we should try and inform the user better using all the data we have, and let her decide for herself. Our first priority to address this issue is the design of more informative and understandable extension install dialogs, while later plans include better reporting infrastructure and opt-in, privacy-preserving usage data collection.

Privacy policy: Privacy policies are thankfully getting more widespread in the CWS, containing important clues to what the extensions should and should not do, any why. The main problem is that few humans can (or take the time to) review these documents, much less cross-reference them with the extension's permissions and behavior. This is unfortunate, since cross-referencing could significantly help in making the previously discussed features much more definitely identifiable as malicious or benign. Efforts that aim to make privacy policies more understandable [14] could be useful additions to existing detection methodologies. Even if natural language processing (NLP) does not facilitate full automation yet, it could still aid decision making for extension vetting experts and users alike. We also note that NLP methods could be similarly useful for analyzing descriptions and reviews to get a better intuition of what the extension is supposed to do and compare that to what it is actually doing to detect anomalies.

Deviation from the "pack": Statistics, especially clustering and outlier detection should be more heavily utilized as well. For example, data leakage, third-party contacts, or similar characteristics could be interesting and informative not just as absolute values on their own, but also as relative distances from corresponding metrics of similar extensions. The same idea is behind previous research [27] that applies it to websites from the viewpoints of tracking, fingerprinting, 3rd party content, and leaky forms. We have ongoing experiments with adapting the principles of Gorla et al. [13] to browser extensions. Initial results dealing with permission declarations show that deviations are mostly due to common developer practices of gross over-privileging, but we remain convinced that the new perspective this technique provides can be beneficial to other features as well.

Historical context: Another approach missing focus is trying to classify *changes* in extensions instead of looking at versions in isolation. Blog posts referenced so far show that the majority of malicious extensions start out as benign and turn bad after a specific modification. This suggests it would be worthwhile to try and catch hijackings not just through the infected version of the extension, but the change that triggers the extension to turn malicious — possibly in the context of more, previous changes. History is already utilized in Chrome for added permission requests upon updating, but that is not sufficient: making every potential feature history-aware would paint a much clearer picture. Historical context should be viewed not as a new feature, but a new dimension for all features, similarly to the statistical deviations discussed above. Additionally, preventing automatic updates for a change in any feature until central (or user) approval could significantly diminish the number of extensions *turning* bad.

Coverage and symbolic execution: Existing approaches rarely measure how much of the extensions they managed to exercise during analysis, although multiple papers mention it as intended future work. Coverage would indeed provide a good baseline, as one of our goals should be raising it as high as we can to be sure that the features we extract represent the full functionality of the extension. Current work focuses on the number and the addresses of visited sites, and creating an evolving HoneyPage [18], [26] trying to satisfy the extension's expectations. Symbolic execution, on the other hand, could attempt to generate inputs — be they DOM structures, URLs, storage contents, or user action sequences — that manage to exercise the whole source code. Significantly more resources and effort would be needed per extension, but it would be a more complete and robust version of this "behavior eliciting" mindset.

A better security model: We note that we focus this much only on better features and user involvement under the assumption that we have to make the best of the current extension standards and APIs. Were we to relax that assumption and look further into the future, we would also heavily advocate for finer-grained permission declarations in the manifest, separated permissions for both background and content scripts (or even configurable on a per-script basis), more precise and content sensitivity aware DOM accesses, and a more widespread use of (possibly negotiable) content security policy headers [12], [21], [6], [15].

V. Conclusions

In this short overview of current extension vetting methodologies, we look at frequently extracted features of the related literature, and provide examples and scenarios where they could be misleading. We also list a number of insufficiently explored feature avenues that might lead to more robust and accurate detection approaches. We conclude that the binary nature of blocking or allowing is too black-and-white for the many shades of "grey" extensions that exist in online stores today, especially when the evidence used to make that decision is not always correct from the user's perspective. Therefore, instead of solely pursuing the one true mapping between automatically detectable properties and "malicious intent", we should also aim to make the user a part of the process through more comprehensive warnings and by utilizing usage statistics. Our final suggestion for the future of extension vetting research, then, is to progress more towards better features and deeper user involvement while planning (and advocating) for a tighter extension security model.

Acknowledgment

The authors would like to thank Sid Stamm for his contributions to the extension vetting experiments that prompted this study.

References

[1] L. Abrams. (2018) MEGA Chrome Extension Hacked To Steal Login Credentials and CryptoCurrency. [Online]. Available: https://www.bleepingcomputer.com/news/security/mega-chrome-extension-hacked-to-steal-login-credentials-and-cryptocurrency/

[2] A. Aggarwal, S. Kumar, A. Shah, B. Viswanath, L. Zhang, and P. Kumaraguru, "I Spy with My Little Eye: Analysis and Detection of Spying Browser Extensions," in *3rd IEEE European Symposium on Security and Privacy (EUROS&P 2018)*, London, 2018.

[3] P. Arntz. (2018) New Chrome and Firefox extensions block their removal to hijack browsers. [Online]. Available: https://blog.malwarebytes.com/threat-analysis/2018/01/new-chrome-and-firefox-extensions-block-their-removal-to-hijack-browsers/

[4] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett, "VEX : Vetting Browser Extensions For Security Vulnerabilities," in *USENIX Security Symposium*, 2010, pp. 339–354.

[5] A. Barth, C. Jackson, C. Reis, T. Team *et al.*, "The security architecture of the chromium browser," *Technical report*, 2008.

[6] L. Bauer, S. Cai, L. Jia, T. Passaro, and Y. Tian, "Analyzing the dangers posed by Chrome extensions," *2014 IEEE Conference on Communications and Network Security (CNS 2014)*, pp. 184–192, 2014.

[7] Brave Software, "Brave Ad Block," https://github.com/brave/ad-block, 2018.

[8] Q. Chen and A. Kapravelos, "Mystique: Uncovering Information Leakage from Browser Extensions," in *ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, 2018, p. 14.

[9] C. Cimpanu. (2017) "Particle" Chrome Extension Sold to New Dev Who Immediately Turns It Into Adware. [Online]. Available: https://www.bleepingcomputer.com/news/security/-particle-chrome-extension-sold-to-new-dev-who-immediately-turns-it-into-adware/

[10] L. F. DeKoven, S. Savage, G. M. Voelker, and N. Leontiadis, "Malicious Browser Extensions at Scale: Bridging the Observability Gap between Web Site and Browser," in *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET'17)*, Vancouver, BC, 2017.

[11] Disconnect, "Disconnect blacklist," https://github.com/mozilla-services/shavar-prod-lists/blob/master/disconnect-blacklist.json, 2018.

[12] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *Proceedings of the 2Nd USENIX Conference on Web Application Development*, ser. WebApps'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 7–7.

[13] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, pp. 1025–1035, 2014.

[14] H. Harkous, K. Fawaz, R. Lebret, F. Schaub, K. G. Shin, and K. Aberer, "Polisis: Automated analysis and presentation of privacy policies using deep learning," *CoRR*, vol. abs/1802.02561, 2018.

[15] D. Hausknecht, J. Magazinius, and A. Sabelfeld, "May I?-content security policy endorsement for browser extensions," in *12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2015)*, Milan, Italy, 2015, pp. 261–281.

[16] N. Jagpal, E. Dingle, J.-P. Gravel, P. Mavrommatis, N. Provos, M. A. Rajab, and T. Kurt, "Trends and Lessons from Three Years Fighting Malicious Extensions," in *24th USENIX Security Symposium (SEC'15)*, Washington, D.C., 2015, pp. 579–593.

[17] Kafeine. (2017) Threat actor goes on a Chrome extension hijacking spree. [Online]. Available: https://www.proofpoint.com/us/threat-insight/post/threat-actor-goes-chrome-extension-hijacking-spree

[18] A. Kapravelos, C. Grier, and N. Chachra, "Hulk: Eliciting Malicious Behavior in Browser Extensions," in *23rd USENIX Security Symposium*, San Diego, CA, 2014, pp. 641–654.

[19] R. Karim, M. Dhawan, V. Ganapathy, and C.-c. Shan, "An Analysis of the Mozilla Jetpack Extension Framework," in *26th European Conference on Object-Oriented Programming (ECOOP 2012)*, 2012, pp. 333–355.

[20] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. A. Kemmerer, "Behavior-based spyware detection," in *15th conference on USENIX Security Symposium - Volume 15*, 2006, pp. 273–288.

[21] L. Liu, X. Zhang, G. Yan, and S. Chen, "Chrome extensions: Threat analysis and countermeasures," in *19th Annual Network & Distributed System Security Symposium (NDSS '12)*, San Diego, California, 2012.

[22] R. Marinho. (2017) Second Google Chrome Extension Banker Malware in Two Weeks. [Online]. Available: https://morphuslabs.com/second-google-chrome-extension-banker-malware-in-two-weeks-6a7c863bc078

[23] R. Pardo, P. Picazo-Sanchez, G. Schneider, and J. Tapiador, "A Runtime Monitoring System to Secure Browser Extensions," in *5th Workshop on Hot Issues in Security Principles and Trust Hotspot 2017*, 2017, pp. 1–5.

[24] R. Perrotta and F. Hao, "Botnet in the Browser: Understanding Threats Caused by Malicious Browser Extensions," *eprint arXiv:1709.09577*, no. 306994, pp. 1–11, 2017.

[25] O. Starov and N. Nikiforakis, "Extended Tracking Powers: Measuring the Privacy Diffusion Enabled by Browser Extensions," in *26th International Conference on World Wide Web (WWW '17)*, Geneva, Switzerland, 2017, pp. 1481–1490.

[26] ——, "XHOUND: Quantifying the Fingerprintability of Browser Extensions," in *38th IEEE Symposium on Security and Privacy (IEEE S&P)*, Oakland, 2017, pp. 1–19.

[27] ——, "PrivacyMeter : Designing and Developing a Privacy-Preserving Browser Extension," in *10th International Symposium on Engineering Secure Software and Systems (ESSoS 2018)*, 2018, pp. 77–95.

[28] M. Ter Louw, J. S. Lim, and V. N. Venkatakrishnan, "Enhancing web browser security against malware extensions," *Journal in Computer Virology*, vol. 4, no. 3, pp. 179–195, 2008.

[29] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. Mccoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. A. Rajab, "Ad Injection at Scale: Assessing Deceptive Advertisement Modifications," in *2015 IEEE Symposium on Security and Privacy*, vol. 2015-July. IEEE, may 2015, pp. 151–167.

[30] J. Wagner. (2018) Trustworthy Chrome Extensions, by Default. [Online]. Available: https://security.googleblog.com/2018/10/trustworthy-chrome-extensions-by-default.html

[31] J. Wang, X. Li, X. Liu, X. Dong, J. Wang, Z. Liang, and Z. Feng, "An empirical study of dangerous behaviors in firefox extensions," in *15th International Conference on Information Security (ISC 2012)*, Passau, Germany, 2012, pp. 188–203.

[32] M. Weissbacher. (2016) These Chrome extensions spy on 8 million users. [Online]. Available: http://mweissbacher.com/blog/2016/03/31/these-chrome-extensions-spy-on-8-million-users/

[33] M. Weissbacher, E. Mariconti, G. Suarez-Tangil, G. Stringhini, W. Robertson, and E. Kirda, "Ex-Ray: Detection of history-leaking browser extensions," in *33rd Annual Computer Security Applications Conference (ACSAC 2017)*, vol. Part F1325, 2017, pp. 590–602.

[34] X. Xing, B. Lee, R. Perdisci, W. Lee, W. Meng, B. Lee, U. Weinsberg, A. Sheth, R. Perdisci, and W. Lee, "Understanding Malvertising Through Ad-Injecting Browser Extensions," in *24th International Conference on World Wide Web (WWW '15)*, 2015, pp. 1286–1295.