Extended Tracking Powers: Measuring the Privacy Diffusion Enabled by Browser Extensions

Oleksii Starov
Stony Brook University
ostarov@cs.stonybrook.edu

Nick Nikiforakis
Stony Brook University
nick@cs.stonybrook.edu

ABSTRACT

Users have come to rely on browser extensions to realize features that are not implemented by browser vendors. Extensions offer users the ability to, among others, block ads, de-clutter websites, enrich pages with third-party content, and take screenshots. At the same time, because of their privileged position inside a user's browser, extensions have access to content and functionality that is not available to webpages, such as, the ability to conduct and read cross-origin requests, as well as get access to a browser's history and cookie jar.

In this paper, we report on the first large-scale study of privacy leakage enabled by extensions. By using dynamic analysis and simulated user interactions, we investigate the leaking happening by the 10,000 most popular browser extensions of Google Chrome and find that a non-negligible fraction leaks sensitive information about the user's browsing habits, such as, their browsing history and search-engine queries. We identify common ways that extensions use to obfuscate this leakage and discover that, while some leakage happens on purpose, a large fraction of it is accidental because of the way that extensions attempt to introduce third-party content to a page's DOM. To counter the inference of a user's interests and private information enabled by this leakage, we design, implement, and evaluate BrowsingFog, a browser extension that automatically browses the web in a way that conceals a user's true interests, from a vantage point of history-stealing, third-party trackers.

Keywords

Privacy diffusion; browser extensions; web tracking

1. INTRODUCTION

In recent years, there has been a significant amount of research that aims to investigate the web tracking practices of websites and offer technical countermeasures that users can utilize. Researchers have shown that, among others, websites use third-party cookies, Flash LSOs, and the browser cache, to hide identifiers which can be used to track users across websites [19, 20, 26, 27]. To withstand cookie deletion, websites have also been found to engage in browser fingerprinting where natural variations in a user's browsing environment are converted to near-unique user identifiers [7, 8, 14, 23].

A critical aspect of the modern browsing environment that has, so far, escaped the attention of researchers is that of browser extensions. Modern browser extensions are written using HTML, JavaScript, and CSS, and are therefore as capable of tracking users as websites. Moreover, browser extensions enjoy access to APIs that are out of reach of regular websites (such as those allowing them to perform cross-origin requests) which can make user tracking easier and more intrusive. Despite the great potential of abuse, the privacy practices of browser extensions have not been investigated systematically, with the only research that exists being that of blog posts discussing the issue and manually analyzing a handful of browser extensions [5, 6, 30].

To help better understand the phenomenon of tracking through browser extensions, in this paper, we present the first large-scale study of the privacy diffusion enabled by browser extensions. We utilize the phrase "privacy diffusion", coined by Krishnamurthy and Wills in the context of regular browsing [19], to capture the phenomenon of an ever-increasing number of third-parties becoming privy to a user's browsing habits. Namely, we develop a dynamic-analysis framework which we use to automatically analyze the 10,000 most popular Google Chrome extensions, in search of information leakage, such as, extensions that leak a user's browsing history and search queries, to one or more third parties. Through this process we discover that 6.3% of the evaluated extensions perform some kind of leakage, with the majority of these cases being accidental, i.e., leakage enabled by the way that extensions introduce content in a visited website.

In terms of intentional leakage, we identify hundreds of extensions that share a user's browsing history with third parties, in the context of offering that user a service, such as, sending the currently-visited URL to third-party servers in order to locate shopping coupons or assess its web ranking and safety. While we manually analyze some of these extensions and report our findings, this observation is overall troubling since it complicates the assessment of an extension as suspicious or not. This is because such an assessment would require automated system to determine whether a leakage is because of an extension's stated goals, or is extraneous.

Motivated by this finding, we design and develop BrowsingFog, a countermeasure that allows users to protect their

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052596

privacy, while assuming the presence of one or more browsing-data-leaking extensions. BrowsingFog is a browser extension that constantly evaluates a user's browsing habits and simulates visits to webpages of other random website categories, hiding a user's real interests in a "fog" of irrelevant information.

2. DATA COLLECTION AND ANALYSIS

To quantify the tracking practices of browser extensions, we focus on Google Chrome and we analyze the 10,000 most popular extensions available in the Chrome Store. Note that, while Google Chrome was chosen due to its market share, our system does not make use of any Chrome-specific functionality hence it could be straightforwardly ported to analyze the privacy leakage perpetrated in other extendable browsers.

Detecting privacy leakage by browser extensions is akin to detecting malware, a known hard problem. Since browser extensions are written in JavaScript, extension developers can use heavy obfuscation and dynamic code evaluation to evade any checks done through static analysis. For our approach, we opted for dynamic analysis where an extension is automatically installed in a monitored browser environment where the system provides stimuli and observes the outgoing network requests in search of leaked private information. While this approach bypasses all the known issues of static code analysis, our method, as most dynamic analysis methods, may be prone to false negatives due to reduced code coverage and the potential use of custom encryption methods that will stop our network monitor from matching the outgoing requests to known private information. For the former issue (that of code coverage) we argue that if extension developers seek to opportunistically track a user's online behavior, this tracking should be straightforwardly triggered by the user merely browsing on the web. For the latter issue (that of custom encryption and obfuscation), there exist many prior papers on detecting the leakage of Personally-Identifiable Information which share the same limitations with our work, and yet have provided significant utility to their users [24, 25, 28].

Figure 1 shows our developed system for installing and analyzing browser extensions. The core automation is implemented using Selenium's ChromeDriver [2]. Each extension is automatically installed in an instance of the Google Chrome browser before the browser is automatically navigated to hundreds of webpages and made to interact with them. All traffic originating from the monitored browser is channeled through an instance of mitmproxy [3] which captures packet traces on all ports greater than 22, so as to capture standard HTTP/HTTPS requests as well as outgoing traffic using non-standards ports, e.g., enabled by extensions with proxying capabilities.

While it is certainly possible to navigate to popular sites with and without an extension and compare the outgoing requests (in search of new requests carrying private information), this approach brings a whole host of problems related to the dynamic nature of the web. Due to third-party advertisements, featured content, and client-side widget integration, the same URL can be visited multiple times only seconds apart, and yet result in significantly different outgoing requests. In these cases, the attribution of a request to the evaluated extension or the website itself becomes challenging, and heuristic-based methods are bound to be susceptible to false positives and false negatives. In our work, we circumvent this problem by never really visiting remote websites, but only pretending to visit them by resolving them on our local machine and serving a mock page with known, static content. As before, we expect leaking browser extensions to opportunistically send a user's web traffic to third-party servers and hence we see no reason why an extension would try to evaluate whether a website is "real", before leaking its URL to the outside world.

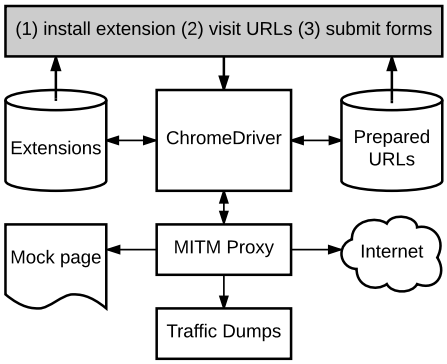

Figure 1: *System for the dynamic analysis of browser extensions for information leakage.*

In the following sections we explain in more detail: (1) what private information we consider to be sensitive; (2) how we define a privacy leakage and third-parties; (3) how we inspect the captured traffic to detect leakage of private information to a third-party.

2.1 Types of private information

In the current study we measure extension-based leakage of the following types of private information:

- **History leakage (HL)**: We detect History Leakage by visiting several hundreds of the web's most popular URLs. Specifically, we focused on the top 50 Alexa domains, where for each domain we retrieved up to 20 popular URLs and subdomains using suggestions from the Bing search engine. In total, each extension is exposed to 780 URLs, which resolve to 308 different subdomains. The resulting number of requests allows us to detect the accidental leakage of browsing history via the HTTP Referer header (e.g. when an extension embeds a remote object on every visited webpage), as well as trigger the intentional tracking behavior of extensions, regardless of method of history exfiltration (e.g. leaking each individual request, or leaking URLs in batches). For the latter type of history leakage, we detect both the leakage of the entire visited URL as well as the main domain of each visited website.

- **Search query leakage (SL)**: We detect Search-Query leakage by automating Google searches and looking for the presence of our queries in outgoing traffic towards unrelated third parties. We chose to decouple search-query leakage from generic history leakage since it is entirely possible that a third-party tracker is only interested in the search queries of users and keeps a low tracking footprint by avoiding the leakage of entire domains and URLs.

- **Form data leakage (FL)**: By submitting a mock HTML form with specific values on several URLs, and later searching for these values being transmitted to a third party in the captured traffic, we can detect the leakage of data that a user explicitly types in HTML forms. To identify leaks even when the information is slightly obfuscated, we utilize a method that we developed in past work [28], where we submit a form three times, two with the same and one with a changed input, and search for the same behavior in obfuscated payloads of outgoing requests.

- **Information about other extensions (EL)**: Since a browser extension can list other extensions installed in the user's browser, we also created scenarios to account for "extension-information leakage". Namely, by installing the under-test extension simultaneously with other extensions (e.g., a sensitive one like Mailvelope for encrypted communication via email), we can later search for the presence of extension identifiers in outgoing traffic.

2.2 Defining privacy leakage

We define privacy leakage to be an event where any of the aforementioned private data (e.g., browsing or search history, information submitted through HTML forms, and lists of installed extensions) is transmitted to a third-party server. If a leakage happens because of a specific browser extension, we say that this extension enables privacy diffusion of a user's sensitive information.

In this paper, we consider as a *third-party* any domain that does not share the TLD+1 part with the first-party visited domain (or any distinct IP address). Hence, if the first-party website is `www.example.com`, a request towards `log.tracker.com` is a third-party request, but a request towards `members.example.com` is not. In practice, this definition may capture benign cases, such as, private domains used by a browser extension for their own backend APIs, as well as true tracking cases, such as, third parties who have no relationship with a given extension. For the former case, an extension that requires the capturing of private information to achieve its stated goals must provide a proper privacy policy which described what data is gathered, and how that data is handled and warehoused. For the latter case, the exfiltrated data was never necessary for the correct operation of the extension, hence each such request further diffuses a user's online privacy.

For the rest of this paper, we try to distinguish between *accidental* and *intentional* privacy leakage, and report our findings accordingly. For example, leakage that happens via the HTTP Referer header is, in most cases, accidental and stems from the way third-party content is introduced in web pages. Contrastingly, a highly obfuscated POST request from an extension's background page can be classified as "intentional", since this request cannot have reasonably occurred by accident. It is worth mentioning that with the term "intentional" we do not mean "malicious" or "intrusive." We merely mean that an extension, whether through its own code or through that of embedded SDKs, explicitly sends a user's private information to a third party. Whether that request is malicious or intrusive, largely depends upon an extension's stated goals.

Listing 1 Obfuscation used by known tracking libraries

```
// upalytics.com and wips.com
base64(base64(payload))

// fairsharelabs.com
btoa(pako.deflate(root.dca_compressor.utf16to8(
JSON.stringify(requests)), {to: "string"}));

// prestadb.net
utf8Str = ANALYTICS.utils.utf16to8(
JSON.stringify(batchPool));
uint8Array = compressor.deflate(utf8Str);
encodedData = utils.encode(bin2String(uint8Array));
ANALYTICS.xhr.sendPost(dataUrl, {}, encodedData);
```

Algorithm 1 Pseudocode of the traffic analysis to detect obfuscated information leakage

```
for extension in extensions do
    for request in getTrafficDump(extension) do
        if isThirdParty(request) then
            ▷ From query string, body, headers and cookies
            queue ← extractValues(request)
            iteration ← 0
            while !queue.empty() & iteration < MAX do
                iteration ← iteration + 1
                value ← dequeue(queue)
                if len(value) < 5 then
                    continue
                checkForLeakage(value)
                ▷ From URL encoding, JSON, base64, deflate,
                ▷ or key=value&key=value...
                newValues ← tryDecode(value)
                if newValues then
                    enqueue(queue, newValues)
                    continue
```

2.3 Traffic analysis

As mentioned in Section 2.1, our framework installs an extension, provides opportunities for privacy leakage, and then analyzes the captured outgoing traffic for instances of that leakage. In the case of plain-text leakage, such as, accidental leakage through the HTTP Referer header, discovering that leakage is a straightforward string search of known private information in outgoing requests, either in their original or URL-encoded form. Note that leakage done over HTTPS is captured by this method since our framework uses a mitmproxy-provided certificate which allows requests and responses to be decrypted before stored.

At the same time, extensions that are involved in intentional tracking are free to use various types of obfuscations especially when they seek to avoid detection. According to prior reports on the ability of extensions to track users [5,6,30] as well as our manual analysis of known tracking extensions, the most popular encoding/obfuscating techniques used by the current generation of tracking extensions are the following: URL encoding, base64, repeated base64, gzip/deflate, and JSON-packing. Examples of such obfuscation are shown in Listing 1. If an extension does not leak information in plain text and does not utilize one or combination of the aforementioned obfuscation schemes, our current methodology will unfortunately not be able to detect the exfiltration of private information.

Algorithm 1 illustrates our method for identifying user-data-leaking requests using different combinations of known encodings. We extract individual parameters from third-party requests and iteratively attempt to de-obfuscate them separately. A challenge that we need to overcome is that it is

Table 1: *Top 20 TLD+1 domains called by extensions when browsing popular websites*

Domain	# Extensions
google-analytics.com	1,700
google.com	936
googleapis.com	828
gstatic.com	823
doubleclick.net	818
facebook.com	530
facebook.net	467
fbcdn.net	274
twitter.com	266
googlesyndication.com	252
100pages.top	194
adnxs.com	179
cloudflare.com	178
yahoo.com	174
cloudfront.net	172
2mdn.net	163
amazonaws.com	158
youtube.com	149
addthis.com	141
advertising.com	138

not always possible to automatically distinguish whether a string was successfully and meaningfully decoded or not (e.g. in case of base64), and thus we have to keep performing de-obfuscation attempts, either until the parameters degenerate to very short strings, or until we reach a predefined threshold. From our pilot tests and analysis of random extension samples, we discovered that a limit of 1,000 iterations is sufficient to uncover leakage while bounding the time spent on the analysis of each individual extension. Even though an attacker can clearly utilize more than 1,000 layers of obfuscation to evade our current architecture, our framework can be straightforwardly extended to count the JavaScript-level calls to obfuscation-related APIs that each extension makes (e.g. through wrapping these APIs or modifying the underlying browser code). Extensions that make heavy use of these APIs can by flagged during dynamic analysis and further investigated.

3. ANALYSIS OF RESULTS

In this section, we first report the statistics of the detected cases of browser extensions leaking private information to third parties, and then describe the different types of accidental and intentional leakage that we discovered and the third parties contacted.

From the top 10K extensions, our framework was able to successfully download and install 9,839 extensions. 38% of these extensions make new requests towards third parties and, on average, each extension contacts six different third-party hosts. Overall, the installed extensions contacted a total of 3,526 unique third-party domains (TLD + 1) with the most popular third-party domains shown in Table 1.

One can see that many browser extensions use popular tracking solutions like `google-analytics.com`, issue requests to social APIs like `facebook.com` and `twitter.com`, and call their own APIs hosted on `amazonaws.com` and those protected by `cloudflare.com`. Listing 2 shows a benign example where an extension is using Google Analytics to track its usage, i.e., track the installation and update events. We argue that while this extension causes new third-party requests, these requests do not leak any private information towards `google-analytics.com` and therefore do not cause unwanted privacy diffusion.

Listing 2 Popular usage of Google Analytics on an extension's background page

```
_gaq.push(['_trackEvent', 'Install', chrome.app.
    getDetails().version]);
...
_gaq.push(['_trackEvent', 'Update', chrome.app.
    getDetails().version]);
```

Once we move past the most popular third-party domains, it is hard to reason, at a domain level, whether a third-party was legitimately contacted by an extension or it was contacted solely for the purpose of exfiltrating user data. The fact that 66% of the 3,526 unique third-parties were only contacted by a single extension further complicates the request-attribution process. Moreover, according to Web-Of-Trust statistics, only nine of the contacted third parties are blacklisted and 61 have a very low trust score. For this reason, in the rest of this section, we focus on domains and extensions where our monitoring tool was able to detect explicit information leakage.

3.1 Detected leakage

Overall, our system allowed us to locate 618 extensions (6.3%) that introduce privacy leaks of browsing and search history, whether accidentally (245), intentionally (188), or both (185). In addition to these 618 extensions, we found 33 extensions leaking information about other installed extensions, and 4 leaking data entered in web forms. Note that, when compared to other types of unwanted extension behavior, privacy-leaking extensions appear to be a much more widespread phenomenon. Specifically, Kapravelos et al. analyzed 48K extensions and found only 130 (0.27%) malicious extensions [18], while Xing et al. analyzed 18K extensions and discovered 348 (1.9%) extensions related to advertising fraud [31]. Even though one could argue that many extensions need to "leak" sensitive information as part of their stated goals (e.g. the Web-Of-Trust extension needs to know each webpage that a user visits so-as-to provide a security score for that page), it is worrisome that only 10% of these 618 extensions included the phrase "privacy policy" in their Chrome Store descriptions.

The distribution of privacy-leaking browser extensions is not uniform across rankings. Figure 2 illustrates that, in general, more popular extensions tend to leak more often than less popular ones, with the largest fraction of leaking extensions concentrated in the top 1,000-3,000 range. Moreover, for the top 3K extensions, we can see that accidental leakage happens significantly more than intentional leakage. One reasonable explanation is that these popular extensions attempt to provide more utility to their users and end-up including many more third-party libraries in a way that causes accidental leakage, e.g., through the Referer header of third-party HTTP requests. We provide more insights into this accidental leakage, later in this section.

By comparing the percentage of leakage across extension categories, it becomes clear that certain types of extensions are more prone to accidental and intentional leakage. As Table 2 illustrates, the "shopping" category of extensions has the largest percentage of accidental and intentional leak-

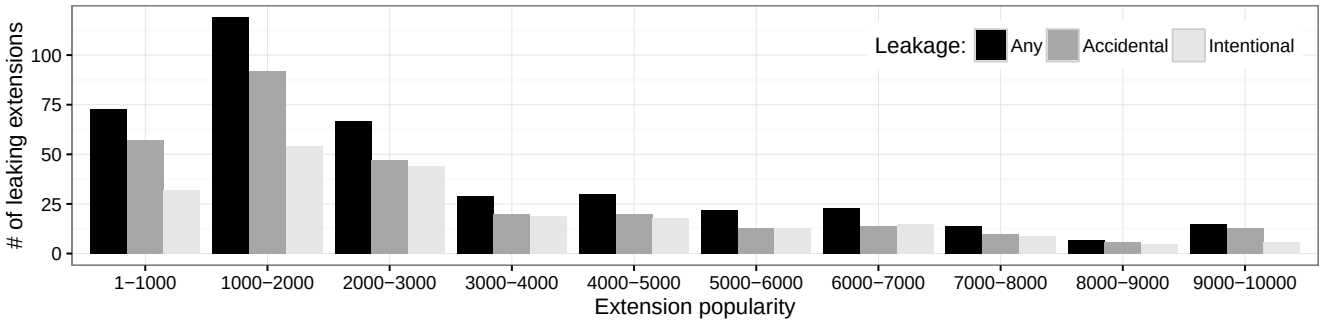

Figure 2: *Distribution of the detected privacy leaking extensions from the less to the most popular. Higher-ranked extensions tend to leak more with the larges fraction on the range of the top 1000-3000.*

Table 2: *Fraction of leaking extensions by category*

Category	# Ext.	Accidental	Intentional
Productivity	3528	2.5%	2.4%
Fun	1268	3.5%	2.0%
Social & Commun.	1226	8.3%	5.0%
Developer Tools	965	1.3%	2.8%
Accessibility	871	3.3%	2.4%
Search Tools	631	4.1%	5.9%
Shopping	443	20.7%	21.0%
Unknown	340	6.1%	5.0%
News & Weather	314	2.2%	1.2%
Photos	189	1.0%	0.5%
Blogging	121	4.1%	0.8%
Sports	104	0.9%	0.9%

age. Extensions belonging to this category usually intentionally request different third-party APIs that reveal a user's browsing history (e.g., to get relevant offers for the currently viewed product, or search for similar items in other stores). Furthermore, any useful information found is usually presented by injecting highly noticeable widgets in the main page's DOM, which can cause further third-party requests for images, JavaScript and CSS, leading to accidental leakage via the Referer header. Next, we see that social extensions tend to leak private information accidentally, which again can happen because of injected widgets, panels, and styles.

Accidental leakage via the HTTP Referer header

Our extension-analysis framework flagged 430 extensions as ones leak browsing history to, on average, two third-party domains per leaking extension. These extensions include the ones that solely leak information via the Referer header, as well as those that leak information via the Referer header as well as through other intentional methods. Table 3 shows the 20 domains that receive the most accidentally leaked private information, both in terms of history leakage (HL) as well as search-query leakage (SL). By manually analyzing these extensions we witnessed that, in most cases, the accidental leakage happens because of additional elements (e.g., images, JavaScript, CSS, custom DOM items) being injected in the visited webpages via an extension's content scripts. These injected resources, upon loading, leak the browsing history to third parties via the HTTP Referer header. Even though these extensions leak a user's history in a piecemeal fashion (URL after URL), the third-parties receiving these requests can accurately reconstruct a user's entire browsing history and use it to infer sensitive information about a user.

An unexpected discovery was that a number of extensions introduce custom toolbars to the visited webpages (examples shown in Figure 3). It is worth noting that, Google Chrome, was one of the first browsers that consciously moved away from allowing extensions to create toolbars in the browser's UI. Therefore, the extensions that still want to show users a toolbar, now need to construct that toolbar out of HTML, CSS, and JavaScript and inject it in every page that a user visits. Every time that these toolbars are injected, they trigger third-party requests, leaking each and every URL to the third-parties providing resources associated with the creation of each toolbar.

Table 4 shows the most popular privacy leaking extensions. There we see many popular shopping extensions (including the security-oriented Avast SafePrice) and extensions for social platforms and news-curating websites. Many of these extensions have millions of users who are likely not aware that one or more third parties get to reconstruct their entire browsing history because of a specific extension that they downloaded.

At the same time, among the 430 extensions we also found several suspicious cases. For instance, the Ask Wiki extension, which had almost 1,000 users, sent all visits to the suspicious `analyticssgoogle.com` domain via the Referer header (likely trying to pose as `google-analytics.com`). Similarly, we discovered two extensions that were outliers in terms of their leakage (leaking to more than 20 third parties) named "YouTube™ WindowChrome Extensions Archive" and "Facebook Room". All three extensions disappeared from the extension store, approximately one month after we first downloaded them.

Intentional non-Referer-related leakage

Analyzing the top 10K browser extensions, our framework discovered 373 extensions that perform intentional privacy leakage of browsing history and search queries, leaking to an average of 1.5 third parties per extension. By intentional leakage we mean that private information is sent to third parties in GET or POST parameters, body, cookies or other custom headers, excluding the HTTP Referer header. As we mentioned earlier, intentional leakage does not necessarily mean leakage with malicious/suspicious intents. It merely means that an extension developer consciously sends a user's visited URLs to one or more third-parties in a way that could not have happened by accident (such as the case of leakage via the Referer header).

Table 5 shows the most popular third-party domains that receive the leaked information. One of the first things that one will notice is that some extensions indeed use Google Analytics for intentional tracking, e.g., the "Flatbook" extension for Facebook (with almost 700K active users) in-

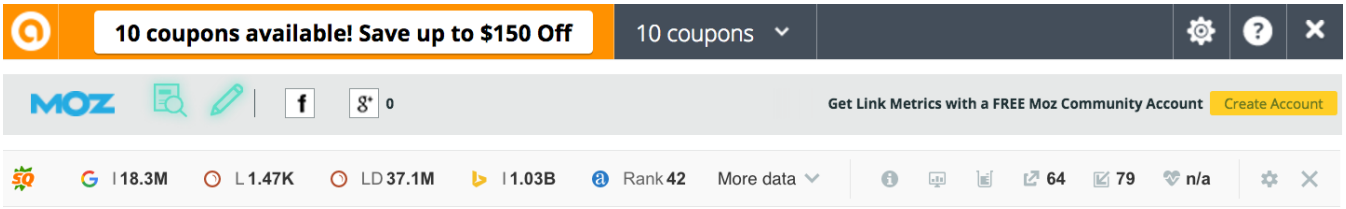

Figure 3: *Examples of extension-injected content that increases privacy diffusion (from the top): Avast SafePrice (10,000,000+ users, adds 'fonts.googleapis.com'), MozBar (360,977 users, adds 4 third-parties), SEOquake (323,683 users, adds 8 third-parties).*

Table 3: *Top 20 third parties that accidentally receive browsing history and search queries*

Domain	# HL	# SL
google-analytics.com	38	5
googleapis.com	38	5
amazonaws.com	29	3
hwcdn.net	15	2
google.com	14	0
okstiker.ru	14	0
purplestats.com	14	13
doubleclick.net	12	0
cloudfront.net	11	5
urlvalidation.com	11	11
facebook.com	10	3
akamaihd.net	9	7
ssl-services.com	8	0
aferon.com	8	8
jfduv7.com	8	0
superyt.ru	8	0
srvtrck.com	8	0
similardeals.net	7	2
cloudflare.com	7	1
apymob.com	6	0

Table 4: *Top 20 extensions that increase privacy diffusion via HTTP Referer*

Extension	#Users	#Parties
Avast SafePrice	10M+	1
Avast Online Security	10M+	1
ZenMate VPN	3,7M	1
Reddit Enhancement Suite	2,3M	1
Honey	1,7M	1
MySmartPrice	965K	11
ShopAtHome.com	938K	4
MusicSig vkontakte	837K	2
SocialLife for Google Chrome	800K	1
SearchLock	765K	1
Flatbook	697K	1
Ebates Cash Back	682K	2
Hootsuite Hootlet	491K	2
SwagButton	451K	2
Keepa - Amazon Price Tracker	410K	1
New XKit	366K	1
MozBar	356K	4
SEOquake	312K	8
Facebook Video Downloader	289K	7
惠惠购物助手	283K	3

jects analytic scripts when users visit Facebook pages to receive statistics about their browsing sessions. Second, among other popular third-parties we again see "big players", such as, Google and Facebook. Our manual analysis of leaks towards these third parties, revealed that these cases occur because of extensions which access Facebook APIs for social-network-related statistics on the visited pages, and Google APIs for ranking information and for storing URLs in Google bookmarks.

From our analysis of the extensions marked as intentionally leaking, we argue that while there is indeed some activity associated with shady trackers that inject similar tracking code across multiple seemingly unrelated extensions (as first observed by the authors of the blog posts that inspired this work [5,30]), most extensions appear to be leaking to different domains that we cannot reasonably cluster together. We base this claim on the following findings: i) 93.3% of the extensions that intentionally leak a user's private details do not utilize complicated obfuscation techniques sending the data either in the clear, or using simple URL encoding and ii) the information-receiving third-party domains resolve to 353 distinct IP addresses which, when grouped based on domain-name and IP-address similarity, result in 205 unique clusters.

Nevertheless, we investigate the most likely cases of intrusive trackers. Out of the clustered third parties, only 77 receive private data from two or more distinct extensions and therefore are likely to belong to shared libraries or services. They affect 250 extensions overall, out of which more than 20% send the data solely to Google Analytics. In Figure 4 we abstract away from specific examples of these third parties and show the relationship between the number of third-party requests and their size, for each analyzed extension. Third-party requests within each extension-testing session are aggregated by the subdomain and path, in order to understand activity of different third-party APIs. There, we see that domains which we identified as tracking tend to cluster together. This could help analysts identify other tracking extensions which, because of higher-levels of obfuscation, our monitoring framework was not able to flag them as information-leaking ones.

In terms of suspicious activity, we were able to group 30 seemingly unrelated domain names as those belonging to a single entity with names, such as, the following: `imp.searchpat.com`, `imp.searchffr.com`, `imp.searchflm.com`, `imp.myemailxp.com` and `imp.myradioxp.com`. Among the extensions that intentionally leaked information about other browser extensions to these domains, was an extension titled "LoginFaster" that has already disappeared from the store previously having 446K active users. Other suspicious leaking extensions are: My Email XP (360K users), My Maps XP (76K), The Package Tracker (56K) and Find Free Recipes (47K) with privacy policies hosted on `searchpat.com` and `searchffr.com` correspondingly. We discovered that those extensions share the same code template to send impressions about the usage of their extensions, but they

Table 5: *Top TLD+1 domains that receive intentionally-leaked private information.*

Domain	# Extensions
History and Search	
google-analytics.com	71
google.com	22
shortem.com	17
facebook.com	16
mixpanel.com	10
apollocdn.com	10
similardeals.net	8
freestikers.top	8
srvtrck.com	7
akamaihd.net	7
HL and SL obfuscated only	
shortem.com	17
mixpanel.com	10
cmptch.com	3
prestadb.net	2
pmddby.com	2
simmersprofession.com	2
odorousructions.com	1
adap.tv	1
newsprompt.co	1
musicsig.com	1
Extension Leakage	
search*.com	18
my*xp.com	12
findizer.fr	1
theweathercenter.co	1
lsmdm.com	1
Form Leakage	
pubsub.googleapis.com	2
gingersoftware.com	1
16mb.com	1

also capture such events like "enabled" or "uninstall" for all installed extensions. Finally, our framework flagged four extensions as collecting information from HTML forms and leaking them to third-parties.

3.2 Leakage over time

In this section we analyze the longitudinal trend on the fraction of privacy leaking extensions in order to understand whether the privacy diffusion that we measured just happened to occur in our snapshot of collected extensions or whether it is a stable property of the browser extension ecosystem. To that extent, we collect a second set of top 10K browser extensions (E_{new}) four months after our first one (E_{old}), use our framework to uncover accidental and intentional leakage, and compare the results.

By comparing the two sets of extensions, we find that more than 13% of extension were substituted ($E_{old} \cap E_{new} \approx 87\%$) (some previously popular extensions are no longer part of the top 10K) and another 22% updated their versions. Table 6 presents the number of detected privacy-leaking extensions for both samples. We see that the number of extensions leaking private information via HTTP Referer remains stable, with 430 versus 431 extensions detected in the history-leakage and search-query leakage categories. This finding reveals that accidental leakage happens because of popular

Table 6: *Comparison of longitudinal results: history leakage (HL), search leakage (SL), form leakage (FL), extension leakage (EL).*

	HL & SL	FL	EL
Accidental (via Referer)			
New sample	430	NA	NA
Old sample	431	NA	NA
Intersection	298	NA	NA
Intentional (non-Referer)			
New sample	373	4	33
Old sample	399	5	17
Intersection	250	4	14

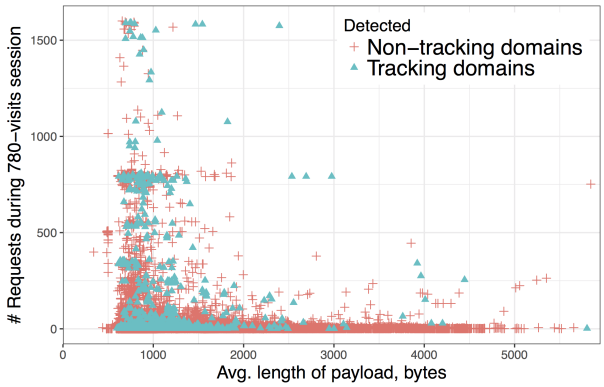

Figure 4: *Scatterplot showing the relationship of number of similar requests to third-party domains and the average size of their payload. Intentionally-leaking requests to tracking domains have two main clusters: (1) one-time setup calls on the install; (2) small-sized requests after each visit to a website. Alongside, we capture repeated leaking calls and larger-sized batched requests.*

functional requirements, e.g., the need to embed a remote JavaScript library, which are likely to remain stable.

Contrastingly, the number of intentionally-leaking extensions decreased by 6.5% for history leakage and search-engine leakage, with only 63% of the extensions that were intentionally leaking in E_{old}, still leaking in E_{new}. At the same time, for the extensions that did continue leaking, we observed a "rotation" in terms of the third-party hosts that were on the receiving side of intentional leakage where we recorded traffic going towards new 74 third parties which we had not observed when analyzing E_{old}.

Overall, through our longitudinal study we find that, while accidental leakage is a stable property of the current extension ecosystem, intentional leakage and suspicious leakage tends to vary. There may be many reasons for this observed variance but one reasonable explanation is that when an extension receives significant negative publicity (such as the ones discussed in the aforementioned blog posts [5,30]) they are either removed from the market, or they remove the intrusive tracking functionality. Presumably, the companies that benefit from this information leakage can always find new extension developers that would be willing to incorporate their code for monetary benefits [16] or transfer ownership of their extensions to them [22].

4. DISCUSSION

One could argue that, once an extension is discovered to be involved in accidental or intentional leakage, users should stop utilizing that extension and, for the more severe cases,

the extension should be removed from the extension store. Note, however, that even using automated privacy-leakage-discovery systems such as the one designed and evaluated in this paper, extensions can amass tens of thousands of downloads before they are detected and removed. As such, we reason that, in addition to reactionary countermeasures, we must provide users with tools to protect themselves for the remaining window of time. In this section, we briefly describe possible solutions on how to protect a user's privacy, for the cases where extension developers are willing to collaborate in reducing accidental leakage, as well as for non-cooperative, intentional-leakage scenarios.

Combating accidental leakage
The main culprit for accidental privacy leakage is the HTTP Referer header that is included with every third-party request that an extension requires. In general, extension developers may protect their users by overwriting such calls and requesting internal pages of extensions instead, which implement the actual HTTP requests to third parties. Similarly, a Referer-hiding "proxy" could also be implemented by using Chrome's messaging APIs, redirecting calls to the background page. In this case, requests towards the outside world would just reveal the extension's identifier through the HTTP Origin header. To facilitate such privacy-preserving architectures, the browser may implement specific APIs for HTTP requests, either for proxying or merely hiding the Referer header of extension-related requests. Similarly, browser developers could investigate architectures that allow extensions to create toolbars in a way that does not clutter the browser's UI, but also keeps these toolbars outside of a page's DOM.

Combating intentional leakage
Assuming that an intrusive extension is not willing to collaborate to reduce privacy diffusion and users cannot tell, a priori, whether an extension is intrusive or not, one could reduce the value of the extracted data by making it less accurate. By "poisoning" the collected truth about a user, the value of the collected data is reduced and could therefore force the extension developers to monetize their software in different ways. In our case, one can poison the collected truth by automatically visiting websites that are unrelated to a user's true interests so that a history-stealing third party cannot differentiate real from simulated interests.

To that extent, we developed BrowsingFog, a Chrome extension which deceives third-party trackers with decoy history of opened tabs and websites. BrowsingFog follows the principles set by other protection-through-deception tools, such as, TrackMeNot [4] (for decoy search queries) or AdNauseam [1] (for decoy clicks on advertisements). BrowsingFog generates visits to 15 categories of websites according to Alexa, attempting to hide the real interests of a user in a "fog" of irrelevant pages. BrowsingFog, in its default setting, opens a new tab every minute, amounting to four page visits per Alexa category during one hour. Naturally, these settings can be adjusted by a user to approximately match the amount of traffic that the user organically produces per hour. Since a tracking extension may capture repeated visits to the same websites, BrowsingFog continuously visits a randomly chosen website for each Alexa category while skipping categories that the user actually visits on a daily basis (e.g., email, social and, sports news).

Our extension opens a "pinned" tab which allows it to reduce its footprint in the browser's UI and allow users to continue browsing as per usual. Our performance evaluation shows that, on average, BrowsingFog requires 50 Mb of RAM to keep the background page, and one Chrome tab with random websites uses approximately 100 Mb at any given point in time.

Currently, BrowsingFog is a prototype and only protects from the leakage of browsing history. One could, however, straightforwardly add more features to BrowsingFog, such as, fuzzing of social network pages, search queries, and the usage of specific and evolving browsing profiles, e.g., simulating a price-sensitive user versus one interested in luxury goods. To test the concept of BrowsingFog, we resolved a domain associated with an intentionally-leaking extension to our server, and installed that extension in a browser running BrowsingFog. The video recording of BrowsingFog's functionality is available on the following link (password: WWW2017): `https://vimeo.com/188468225`

5. RELATED WORK

Our motivation for this work came from three blog posts of researchers who brought attention to the problem of unwanted tracking via browser extensions and manually analyzed a handful of browser extensions [5,6,30]. Interestingly, privacy concerns about the capabilities of browser extensions can be traced back to the year 2000, when Internet Explorer started supporting toolbars and Active-X objects [21].

Even though a manual, in-depth analysis is crucial for understanding the issue, any manual analysis cannot scale to the size and dynamic nature of modern browser-extension markets. To the best of our knowledge, this study is the first study that uses a dynamic-analysis system that can, without human supervision, analyze thousands of extensions and deterministically detect ones leaking private user data. Moreover, because our system searches for the leakage of private information in outgoing traffic, we were able to discover that accidental, Referer-header-related leakage happens significantly more often than the type of shady intentional leakage described in the aforementioned blog posts.

While our work is the first one that investigates the leakage of sensitive user information through browser extensions, browser extensions have attracted a lot of prior work, primarily from a security perspective. In terms of malicious extensions, prior research has proposed systems that can automatically analyze browser extensions and flag those that perform malicious actions, such as, stealing a user's credentials [9, 17, 18] and performing advertisement fraud [29, 31]. In addition to malicious extensions, researchers have also investigated the security architecture and vulnerabilities of browser extensions [13] and proposed countermeasures to reduce the impact of vulnerable browser extensions in a user's browser [10, 11].

On the privacy front, recent years have attracted a significant amount of research in the privacy practices of websites and the techniques that intrusive websites can utilize to track users with and without stateful identifiers [7, 12, 14, 15, 19, 20, 26–28]. Understanding the overlap between traditional web-page-originating tracking and tracking enabled by browser extensions, is crucial for enabling researchers to develop countermeasures that can withstand both.

6. CONCLUSION

In this paper, we investigated the privacy diffusion enabled by browser extensions. By designing and implementing a framework that can automatically install an extension and simulate user browsing while observing the network, we identified that 6.3% of popular Google Chrome extensions leak privacy-sensitive information to at least one third party. Contrary to the findings of previous work, we observed that most leakage occurs in an accidental fashion where the extension developers do not explicitly desire to leak a user's sensitive information. Even for intentional leakage, we explain why it is not always straightforward to gauge the maliciousness of a leaking extension and proposed BROWSINGFOG, a browser extension that hides a user's true interests in a fog of irrelevant browsing.

Acknowledgments: We thank the reviewers for their valuable feedback. This work was supported by the National Science Foundation (NSF) under grants CNS-1617593 and CNS-1527086, as well as the Data Transparency Lab.

7. REFERENCES

[1] AdNauseam. `http://adnauseam.io/`.

[2] ChromeDriver - WebDriver for Chrome. `https://sites.google.com/a/chromium.org/chromedriver/`.

[3] mitmproxy. `https://mitmproxy.org/`.

[4] TrackMeNot. `https://cs.nyu.edu/trackmenot/`.

[5] Detectify labs. Chrome Extensions - AKA Total Absence of Privacy. `https://labs.detectify.com/2015/11/19/chrome-extensions-aka-total-absence-of-privacy/`, 2015.

[6] How-To Geek. Warning: Your Browser Extensions Are Spying On You. `http://www.howtogeek.com/180175/warning-your-browser-extensions-are-spying-on-you/`, 2015.

[7] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[8] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: Dusting the Web for fingerprinters. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, 2013.

[9] S. V. Acker, N. Nikiforakis, L. Desmet, F. Piessens, and W. Joosen. Monkey-in-the-browser: Malware and vulnerabilities in augmented browsing script markets. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2014.

[10] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett. Vex: Vetting browser extensions for security vulnerabilities. In *USENIX Security Symposium*, volume 10, pages 339–354, 2010.

[11] A. Barth, A. P. Felt, P. Saxena, and A. Boodman. Protecting browsers from extension vulnerabilities. In *Annual Network and Distributed System Security Symposium, NDSS*, 2010.

[12] K. Boda, A. M. Földes, G. G. Gulyás, and S. Imre. User tracking on the web via cross-browser fingerprinting. In *Proceedings of the Nordic Conference on Information Security Technology for Applications (NordSec)*, 2012.

[13] A. S. Buyukkayhan, K. Onarlioglu, W. Robertson, and E. Kirda. Crossfire: An analysis of firefox extension-reuse vulnerabilities. In *23rd Annual Network and Distributed System Security Symposium, NDSS*, 2016.

[14] P. Eckersley. How Unique Is Your Browser? In *Proceedings of the 10th Privacy Enhancing Technologies Symposium (PETS)*, pages 1–18, 2010.

[15] S. Englehardt and A. Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2016.

[16] Fairshare: User-friendly, cross-browser monetization solutions for extensions, websites, games and more. `https://www.fairsharelabs.com/`.

[17] N. Jagpal, E. Dingle, J.-P. Gravel, P. Mavrommatis, N. Provos, M. A. Rajab, and K. Thomas. Trends and lessons from three years fighting malicious extensions. In *24th USENIX Security Symposium*, 2015.

[18] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 641–654, San Diego, CA, Aug. 2014. USENIX Association.

[19] B. Krishnamurthy and C. Wills. Privacy diffusion on the web: a longitudinal perspective. In *Proceedings of the 18th international conference on World wide web*, pages 541–550. ACM, 2009.

[20] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner. Internet Jones and the Raiders of the Lost Trackers: An Archaeological Study of Web Tracking from 1996 to 2016. In *Proceedings of the USENIX Security Symposium*, 2016.

[21] D. M. Martin Jr, R. M. Smith, M. Brittain, I. Fetch, and H. Wu. The privacy practices of web browser extensions. *Communications of the ACM*, 44(2):45–50, 2001.

[22] L. Mathews. Shady developers are buying Chrome extensions and turning them into malware. `http://www.geek.com/apps/shady-developers-are-buying-chrome-extensions-and-turning-them-into-malware-1582416/`.

[23] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 541–555, Washington, DC, USA, 2013. IEEE Computer Society.

[24] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, and V. Paxson. Haystack: in situ mobile traffic analysis in user space. *arXiv preprint arXiv:1510.01419*, 2015.

[25] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile*

Systems, Applications, and Services (New York, NY, USA, 2016), MobiSys, volume 16, 2016.

[26] F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 12–12. USENIX Association, 2012.

[27] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle. Flash cookies and privacy. In *AAAI spring symposium: intelligent information privacy management*, volume 2010, pages 158–163, 2010.

[28] O. Starov, P. Gill, and N. Nikiforakis. Are You Sure You Want to Contact Us? Quantifying the Leakage of PII via Website Contact Forms. In *Proceedings of the 16th Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[29] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. McCoy, A. Nappa, V. Paxson, P. Pearce, et al. Ad injection at scale: Assessing deceptive advertisement modifications. In *IEEE Symposium on Security and Privacy (SP)*, 2015.

[30] M. Weissbacher. These Chrome extensions spy on 8 million users. `http://mweissbacher.com/blog/2016/03/31/these-chrome-extensions-spy-on-8-million-users/`, 2016.

[31] X. Xing, W. Meng, B. Lee, U. Weinsberg, A. Sheth, R. Perdisci, and W. Lee. Understanding malvertising through ad-injecting browser extensions. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1286–1295, 2015.