XHOUND: Quantifying the Fingerprintability of Browser Extensions

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

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

***Abstract*—In recent years, researchers have shown that unwanted web tracking is on the rise, as advertisers are trying to capitalize on users' online activity, using increasingly intrusive and sophisticated techniques. Among these, browser fingerprinting has received the most attention since it allows trackers to uniquely identify users despite the clearing of cookies and the use of a browser's private mode.**

In this paper, we investigate and quantify the fingerprintability of browser extensions, such as, AdBlock and Ghostery. We show that an extension's organic activity in a page's DOM can be used to infer its presence, and develop XHOUND, the first fully automated system for fingerprinting browser extensions. By applying XHOUND to the 10,000 most popular Google Chrome extensions, we find that a significant fraction of popular browser extensions are fingerprintable and could thus be used to supplement existing fingerprinting methods. Moreover, by surveying the installed extensions of 854 users, we discover that many users tend to install different sets of fingerprintable browser extensions and could thus be uniquely, or near-uniquely identifiable by extension-based fingerprinting. We use XHOUND's results to build a proof-of-concept extension-fingerprinting script and show that trackers can fingerprint tens of extensions in just a few seconds. Finally, we describe why the fingerprinting of extensions is more intrusive than the fingerprinting of other browser and system properties, and sketch two different approaches towards defending against extension-based fingerprinting.

I. INTRODUCTION

Mayer in 2009 [38] and Eckersley in 2010 [19] showed that, contrary to popular belief, web trackers could keep tracking users without the need of any stateful identifiers, such as, third-party HTTP and Flash cookies. The authors showed that a tracker can create a sufficiently stable, per-user identifier (called a *browser fingerprint*) by combining features that are already present in the browsers of users, such as, their list of installed plugins, the list of fonts, and the HTTP headers of their browsers. Specifically, Eckersley discovered that fonts and plugins were the two most discriminating features, allowing him to uniquely identify 94.2% of the surveyed users. Many researchers have conducted follow-up studies validating and extending Eckersley's findings [12], [14], [22], [36], measuring the adoption of browser fingerprinting in the wild [6], [7], [37], [44], showing that smartphone components are fingerprintable [16], [17], [51], quantifying the fingerprintability of mobile browsers [27], [36], and proposing countermeasures against browser fingerprinting [11], [43], [49].

Laperdrix et al. [36], in a recent study validating and extending Eckersley's findings, discovered that even though browser plugins remain one of the most revealing features of desktop and laptop browsers, the entropy that they offer has considerably decreased since Eckersley's study. This is due to the increasing popularity of HTML5, providing powerful functionality to developers, and the decreasing trust in proprietary browser plugins which have, in the past, caused many performance and security issues. For instance, Google Chrome has stopped supporting plugins utilizing the NPAPI architecture [3] and Mozilla Firefox announced a plan to do the same [5], while already limiting the ability of scripts to enumerate plugins [1]. As the popularity and fingerprinting power of plugins is decreasing, we argue that trackers will look for new techniques to fingerprint users. Predicting these new techniques will allow the research community to start developing countermeasures for these future threats.

In this paper, we show that browser extensions, such as, AdBlock and Ghostery, installed via browser add-on markets can serve as powerful discriminating features for fingerprinting and uniquely identifying the browsing environments of users. Note that, in contrast with browser plugins, there are no browser APIs that webpages can use to retrieve the list of installed browser extensions. As such, the only way that browser extensions can be detected is through their side effects on a page's DOM, such as, the addition of new DOM elements and the removal of existing ones. While researchers are already aware that certain extensions are fingerprintable, previous work has always revolved around the manual analysis of a handful of extensions [7], [41], [44] and the subsequent reasoning about their fingerprintable features.

To quantify the fingerprintability of browser extensions at a large scale, we present XHOUND (Extension Hound), the first fully automated system for fingerprinting browser extensions, based on a combination of static and dynamic analysis. XHOUND fingerprints the organic activity of extensions in a page's DOM, such as, the addition of new DOM elements and the removal of existing ones, and is thus robust against incremental patching of extensions. Using XHOUND, we are able to answer, among others, the following four important questions:

- **How many popular extensions introduce on-page changes and are thus fingerprintable?** We examine the top 10,000 Chrome Store extensions, and show that at least 9.2% of extensions introduce detectable changes on any arbitrary URL, and more than 16.6% introduce

detectable changes on popular domains. The numbers increase to more than 13.2% and 23% respectively, if we consider just the top 1,000 extensions. Moreover, we find that popular extensions remain fingerprintable over time, despite updates and rank changes.

- **What kind of on-page changes do browser extensions introduce?** The possibility of extension-based fingerprinting relies on a tracker's ability to distinguish between introduced changes, i.e., which change was introduced by what extension. For instance, many ad-blocking extensions will result in the same absence of an ad on the page, while the additional UI elements of password managers will tend to have unique HTML code structures. Analyzing XHOUND's results, we show that among 1,656 fingerprintable extensions almost 90% are uniquely identified based on the on-page modifications that they cause.
- **How fingerprintable are the extension profiles of real users?** Extension fingerprinting will only work if users have rather unique sets of detectable extensions. To analyze extension profiles of everyday users, we deploy an extension-survey, which anonymously collects the list of installed extensions from a user's browser. We find that, among 854 participants, 14.1% have distinct sets of universally detectable extensions and can thus be identified with 100% accuracy, while an additional 19.4% of users share the same extension-based fingerprint with 2-20 other users.
- **How can a tracking script check for the presence of browser extensions?** As a final part, we implement a proof-of-concept script that can fingerprint tens of popular browser extensions in just a few seconds, using the triggering conditions and on-page side effects extracted by XHOUND. A video demo of our extension-fingerprinting script is available on this URL: `https://vimeo.com/178330178` (password is SP2017).

Our results highlight the danger of extension-based fingerprinting which, in conjunction with existing fingerprinting techniques, can greatly boost the accuracy of stateless, user identification. Moreover, our findings are likely to be applicable to mobile platforms where most browsers have poor or no support for plugins, yet popular browsers, such as, Firefox Mobile and Dolphin Browser for Android, and Chrome for iOS [32], support extensions. To address the threat of extension-based fingerprinting, we first briefly discuss the difficulty of protecting against it, and then sketch two possible countermeasures, based on isolating DOM changes and constructively polluting the DOM namespace.

II. BACKGROUND

In this section, we first provide a brief comparison of browser extensions and browser plugins and then list the threat models that we will use throughout this paper.

A. Plugins versus extensions

Even though many users tend to use the term "plugin" and "extension" interchangeably [15], [23], [31], [39] in reality plugins and extensions refer to very different technologies.

Plugins allow browsers to parse and display content that is not traditional HTML. Webpages that depend on plugins, directly invoke them through the use of appropriately set `<object>` and `<embed>` tags. Plugins became popular because they enabled the delivery of non-traditional HTML content, such as, video, and audio, at a time when browsers could only support basic HTML. Plugins, such as, Adobe's Flash, and Sun's Java, dominated the landscape of plugins until they started falling out of favor, due to their performance impact on mobile devices [29], the large number of security vulnerabilities that were routinely used to compromise the machines of users [34], [45], and the general tendency of moving away from proprietary libraries and towards open standards. HTML5 delivers many technologies, such as, WebRTC [18], Canvas [48], and native audio and video players, that are now used to build games and applications that, in the past, could only be built using proprietary plugins.

In contrast with plugins, browser extensions are meant to extend or modify the default behavior of a browser and are targeted towards end users, rather than application developers. Browser extensions are built using JavaScript, CSS, and HTML, and make use of well-defined APIs provided to them by browsers. Using extensions, users can modify a browser's interface, add new features that are not, by default, supported, and modify webpages according to their preferences. Users utilize extensions to block ads, download videos, capture screenshots, and manage their passwords. Since the target audience of extensions are users, pages cannot ask for extensions to "load" in the way that they could do with browser plugins. Instead, browser extensions register hooks on various browser events and are allowed to view and modify the DOM of all the webpages whitelisted in their manifest files.

Plugins and extensions are also different from a fingerprinting standpoint. Since plugins are meant to be used by developers, webpages can utilize JavaScript to obtain the list of plugins currently installed in the user's browser (by accessing the `navigator.plugins` object). Webpages that fingerprint their users abuse this functionality to retrieve all plugins installed on a user's browser and turn them into features for differentiating users from one another. In contrast, there is no API that webpages can use to retrieve the list of installed extensions. The only generic method that a webpage can use to detect an extension, is to identify a modified DOM and attribute that modification to an installed extension. In fact, many webpages are currently using this method to detect the presence of ad-blockers (expected ad-related elements are not present in the DOM of a page) [8]–[10].

B. Threat models

For our purposes, an attacker is an entity that wishes to fingerprint the extensions of a user's browser and use this information to uniquely identify the user between browsing

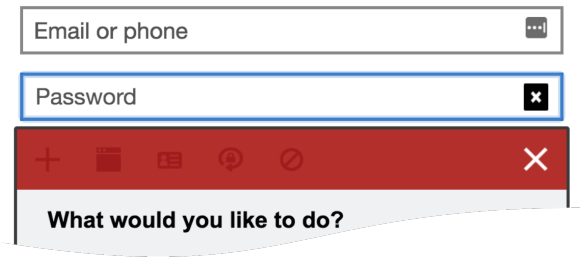

Fig. 1. LastPass extension as an example of content based on-page changes.

Listing 1 DOM changes introduced by LastPass.

```
<input type="password" style="cursor: pointer; background-
    image: url(&quot;
    iVBORw0KGgoAAAANSUhEUgAAABAAAAA...yeBAAAAAElFTkSuQmCC&
    quot;); background-attachment: scroll; background-size:
     16px 18px; background-position: 98% 50%; background-
    repeat: no-repeat;">
```

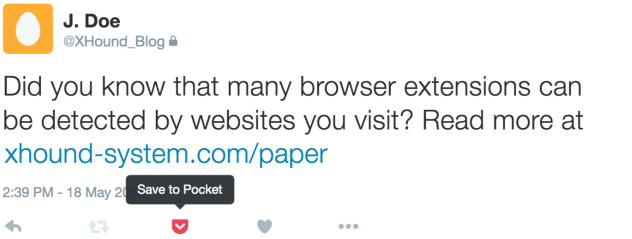

Fig. 2. SaveToPocket extension as an example of URL based on-page changes.

Listing 2 DOM changes introduced by SaveToPocket.

```
<div class="ProfileTweet-action action-pocket-container">
    <a class="js-tooltip" href="#" data-original-title="
        Save to Pocket">
    <span class="icon icon-pocket"></span>
    </a></div>
<div class="ProfileTweet-action...
```

sessions, without relying on cookies or other stateful identifiers that users can delete.

As mentioned in Section II-A, extensions are only allowed to access the DOM of the webpages that are specified in their manifest files. This means that some extensions which add functionality to specific popular web applications, reveal their presence only on specific websites and paths, and consequently, only those websites have the ability to detect them and fingerprint them. For instance, a YouTube video downloader will only reveal its presence when a user is browsing video pages on `youtube.com`. As a consequence, websites that are routinely "selected" by these non-generic extensions have more fingerprinting power than the rest of the web since they can detect and fingerprint both website-specific and generic extensions. Thus, when considering the fingerprintability of extensions, we can distinguish the following two attack scenarios based on the vantage point of the attacker.

Tracking script situated on an arbitrary domain

In this tracking scenario, any arbitrary webpage is able to fingerprint extensions that are installed and enabled in a browser. Such tracking scripts can be loaded inside any custom domain and craft a web page which will trigger extensions to reveal themselves. In this scenario, the tracker will be able to detect only the extensions that introduce changes regardless of the URL of the page. In other words, the detectable functionality of these fingerprintable extensions must be content-dependent instead of URL-dependent. Figure 1 shows an example of content-dependent DOM changes, introduced by the LastPass extension. This extension is a password manager that adds an ellipsis button (**...**) to each input field allowing users to conveniently access their stored credentials. These visible changes are triggered by specific content (a login form) and will work on any domain or URL that includes such a form. Listing 1 shows the corresponding DOM changes. Thus, any tracker situated on any arbitrary domain can create a password input field and check whether an ellipsis button appeared on it, thereby inferring the presence of the LastPass extension in the browser of the current user. Similarly, universal extensions that highlight phone numbers in text, provide additional controls to any videos, or block any ads, can be easily detected in the same fashion.

Tracking script situated on a popular domain

Many popular extensions in the market are targeted towards specific popular websites like Facebook, Twitter, YouTube, and Gmail. Therefore, the changes that these extensions introduce on those websites can be called URL- or domain-specific. Figure 2 shows an example of such DOM changes, introduced by an extension called SaveToPocket. Its functionality includes the ability to save web pages or parts of web pages for later reading. In addition, for specific popular domains, the extension introduces its own buttons on their pages to facilitate users. The current example is based on Twitter, where each tweet receives the additional "save to pocket" button. Listing 2 shows the corresponding DOM changes. Any script loaded from Twitter's domain has the ability to test the presence of this button. This includes both first-party scripts located on Twitter servers, as well as third-party scripts loaded from arbitrary remote domains. In practice, this means that hundreds of third-party script providers, ranging from advertising libraries, to content distribution networks, can capitalize on their privileged position (being included on a popular domain) to fingerprint these URL-dependent extensions and their users. In fact, Nikiforakis et al. [44] found that `skype.com` was including a fingerprinting script from a commercial fingerprinting vendor, and Acar et al. [6], [20] later discovered that many popular websites, including `whitehouse.gov` and `youporn.com` were including a tracking library from `addthis.com` which was using canvas-fingerprinting.

III. XHOUND ARCHITECTURE

A human analyst can straightforwardly reason about the on-page changes made by any given extension (e.g. ad-related iframes removed and new DOM elements added), craft appropriately structured pages that would cause extensions to reveal themselves, and write tests that are used to infer the presence or absence of an extension in the browsers of users. This manual approach, however, cannot scale to the size of popular browser extension markets which are hosts to tens of thousands of extensions. To address the challenge of a large-scale, fingerprintability analysis of extensions, we designed and developed XHOUND (Extension Hound), an extension-discovering framework that automatically extracts the visible and invisible on-page side-effects of any given browser extension. In this section, we report on the architecture of XHOUND and our design choices.

XHOUND uses a two-step approach where in the first step our tool patches the JavaScript source code of extensions in order to place hooks on functions of interest and, in the second step, uses dynamic analysis in an attempt to stimulate the DOM-changing code segments of each evaluated extension. Similar two-step passes have been used to detect malicious extensions that exfiltrate private user data or inject advertisements. Kapravelos et al. introduced Hulk [30], a framework for automatically detecting malicious Chrome extensions. The authors developed the concept of dynamic "honey pages" which are empty pages that have JavaScript code which can dynamically create appropriate DOM elements as those are queried by extensions. Jagpal et al. report on a similar system used internally by Google to identify malicious browser extensions in the Chrome Store [28]. In addition to dynamic honey pages, the authors use static honey pages, i.e., pages that appear to be hosting sensitive content (such as login forms) and could thus trigger the content-dependent functions in extensions. Thomas et al. [47] and Xing et al. [50] developed tools for the dynamic analysis of Chrome extensions, which detect extensions that inject malicious ads or swap ads so that the authors of malicious extensions can benefit from advertising revenue.

Even though XHOUND shares design choices with the aforementioned systems, the goal of our platform is distinctly different. Specifically, in XHOUND, we treat all extensions as benign extensions which have no interest of evading dynamic analysis tools and will modify pages by adding, modifying, and removing DOM elements to achieve their stated goals. XHOUND aims to capture these modifications and use them to build detection code that can infer the presence or absence of any given extension. On the one hand, these relaxed requirements (in contrast with adversarial scenarios) relieve us from implementing complex monitoring mechanisms like traffic analysis. Therefore, in XHOUND, we focus on the ability to analyze the final DOM tree of a web page, which was rendered with the inspected extension being active. On the other hand, we aim to maximize the detection of any functional DOM modifications, and thus we have to trigger as much of an extension's functionality as possible. The ability to stimulate an extension to, for example, create a new benign DOM element is irrelevant for Hulk and the rest of the aforementioned dynamic analysis systems, yet is crucial for our system since that new DOM element can be used as a signature for an extension's presence.

Figure 3 shows the architecture of XHOUND. We elaborate on each component in the following paragraphs.

A. Test preparation

XHOUND first automatically unpacks an extension and patches it with JavaScript code in order to hook into functions of interest. More precisely, inspired by Hulk's honey pages, we developed the OnTheFlyDOM library in JavaScript which, upon inclusion on a web page, intercepts most of the possible queries that an extension may use to locate DOM elements. For instance, XHOUND patches the `document.getElementById` method so that, when an extension uses it to inquire about the presence of an element with a specific identifier, our library will actually create, on-the-fly, such an element, record that it did so, and return that element to the calling script. The end effect is that extensions are made to believe that the queried elements are present on a given page and are thus allowed to continue executing. Our library is included in all honey pages deployed by XHOUND, both static and dynamic, and is injected to all extensions under examination as a first-to-include content script. This gives us the ability to also intercept functions, which are defined in the execution environment of extensions. Those functions can be called from the declared content scripts or injected programmatically, and are mainly used by extensions to query the DOMs of web pages.

In comparison to Hulk's pages, we attempt to increase coverage of multi-step queries of DOM elements. First, when an extension script receives a queried element, it may additionally check it for proper attributes. Since we have no way of predicting these attributes and we cannot hook into their value checks, we populated XHOUND's honey pages with static web elements bearing different parameters, which include parts of popular HTML structures, like the ones found on the results of popular search engines, or YouTube videos. Second, an extension's logic may search for a container element first and launch subsequent queries from it on its subtree, e.g., `container.getElementsByClassName("child")`. This is a rather common JavaScript practice, aimed towards increasing the performance of queries. To address this, XHOUND recursively patches all DOM elements created on the fly with intercepted functions for queries (including the container element from the earlier example), as well as all initial static elements present on the honey page. If such multi-step queries are used to search for a child element, the element will be created and appended to the proper parent node.

Our aforementioned JavaScript library allows XHOUND to fingerprint extensions that expect a particular DOM struc-

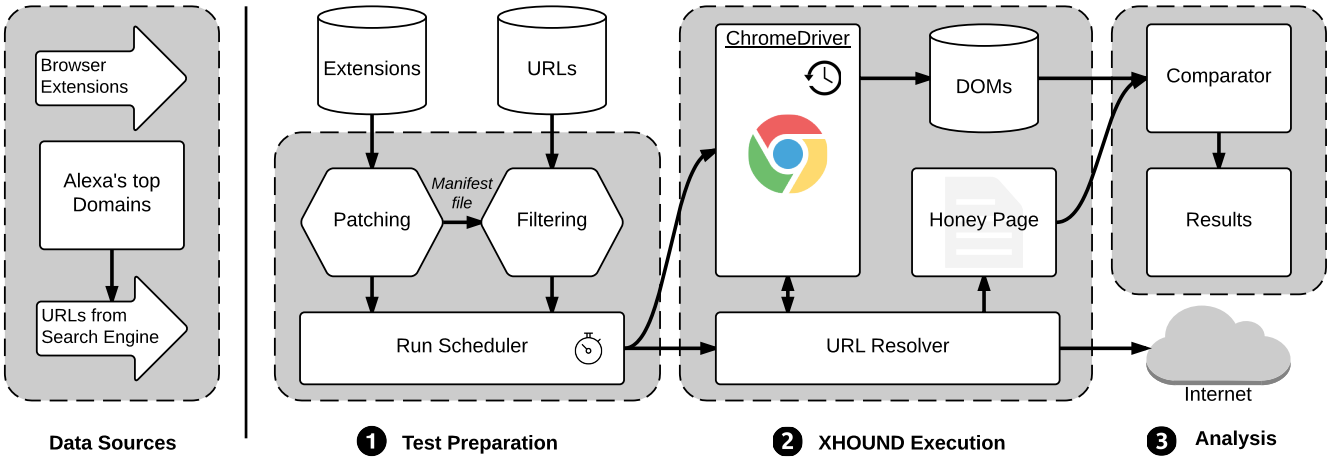

Fig. 3. XHOUND's data sources and architecture: patching extensions and filtering list of target URLs (1), running tests with Chrome Driver (2), and analyzing introduced on-page changes (3).

ture before manifesting themselves. At the same time, many extensions will only inspect the DOM and modify content on specific URLs. A video-downloader extension for YouTube and the SaveToPocket extension described in Section II-B, will only append extra controls for pages hosted on `youtube.com` and `twitter.com`, respectively. While it is certainly possible to navigate to popular sites with and without an extension and compare their DOMs, 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 DOMs. In these cases, attribution of a DOM change to either 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 XHOUND, we tackle this challenge by only *pretending* to visit popular domains, while in fact always visiting static pages which host the aforementioned OnTheFlyDOM JS library. To achieve this we make use of a local DNS stub resolver which resolves the Alexa top 50 sites to `localhost` and an appropriate Apache module that rewrites URLs so as to always serve our honey pages. We address the issues that arise due to HTTPS by installing our own root certificate in the browser used by XHOUND, and accepting all HTTPS errors. Moreover, instead of just pretending to visit the root page of the top 50 Alexa domains, we use a search engine to identify popular URLs for each Alexa domain and pretend to visit up to 20 URLs for each popular domain. In total, each extension is exposed to 780 URLs spanning 308 subdomains. These additional visits are utilized in order to trigger extensions that may be activated only on certain pages or subdomains of popular websites. The combination of these two techniques (local DNS resolutions and on-the-fly DOM population) allows us to both convince extensions that they are on the "right" page and present to them a DOM that is "as expected." Note that we follow this approach because we cannot rely on the URLs whitelisted in an extension's manifest since most extensions request permission to run on every page that a user visits. Even the extensions that request to run on specific domains can include URL checks in their actual code and only execute actions when a user is on a specific page or subdomain of the whitelisted domain. Specifically, from our collected 10,000 Chrome extensions that we analyze in the next sections, more than 67% of them request permissions to all possible URLs, or at least to all non-HTTPS pages.

B. XHOUND Execution

XHOUND's core is based on the Selenium ChromeDriver, which provides the APIs that can be used to launch the Chrome browser with a particular extension installed and navigate it to a particular URL. To support honey pages, XHOUND instructs the domain resolver component to resolve main URLs that the browser visits to our local specially-crafted pages, omitting other calls, which can be internal requests initiated by the extension. That is, we differentiate between the loading of a page that is meant to be locally served, and the request for a remote resource requested by an extension, e.g., an extension fetching a copy of the jQuery library from a content-distribution network. After loading a web page, XHOUND waits for five seconds and takes a snapshot of the current state of the page's DOM tree. Through empirical testing, we chose a five second delay since each extension is taken to a local honey page, which loads fast. However, longer timeouts may potentially reveal even more functional on-page changes from particular extensions, at the expense of increased total analysis time. Moreover, during pilot runs with XHOUND, we noticed that some extensions spent some time after installation to load configuration files or other additional components (e.g. update blacklists or check for newer versions). Failing to account for the time necessary to complete these checks, caused some extensions to either crash, or not be active. As such, we decided to introduce another five-second wait right before instructing Chrome to visit the first target URL. In

its current instantiation, each extension requires appropriately one-hour of dynamic analysis, on a single core and less than 1GB of RAM. Naturally, multiple multicore machines can be used to parallelize the analysis of extensions.

The dynamic honey pages that were described in the previous section (where the DOM is populated on-the-fly, driven by the methods for searching for elements) are not sufficient for revealing all possible cases of on-page changes. Namely, a number of extensions are purely content-based, examining all content on a page and modifying the content that matches their internal logic. For instance, an extension could be searching a webpage's text for phone numbers and making them clickable so as to launch the user's favorite VoIP client. Some adblockers also fall into this category since they try to match elements and outgoing requests to their internal blacklists and remove/block those that match.

To account for these extensions, we designed static honey pages with many types of content, including: tracking scripts, audio and video tags, images, custom tables, phone numbers, suspicious URLs, and login forms. While our static content is by no means exhaustive, it was guided by a manual analysis of a sample of 100 popular extensions so as to ensure that all the fingerprintable extensions that we identified through manual analysis as fingerprintable, would be stimulated by the content of our static honey pages.

C. Analyzing results

All initial DOM elements on the honey pages are indexed to facilitate further comparison of the potentially modified DOM with the one of the original page. Each queried element, which is created on the fly, is also recorded, so we can later check whether it was modified by an extension. This enables us to straightforwardly identify any on-page changes as shown in Figure 3.

D. Limitations

XHOUND currently supports Google Chrome and Mozilla Firefox extensions. We chose to focus on these browsers due to their large market share and wealth of available browser extensions. At the same time, we do not make use of any platform-specific functionality hence our system can be straightforwardly ported to other browsers.

The OnTheFlyDOM library intercepts many popular DOM-querying methods, including:

- `getElementById`
- `getElementsByName`
- `getElementsByTagName`
- `getElementsByTagNameNS`
- `getElementsByClassName`

There are, however, possible ways of accessing the DOM that are currently not supported by XHOUND, such as, discovering elements through global lists, like, `document.forms`.

For the `querySelector` and `querySelectorAll` methods, our library supports the parsing of CSS selectors and attempts to recreate a proper hierarchy of DOM elements exactly as requested by the utilized query. Additionally, we tested our approach in scenarios where an extension makes use of the jQuery library to discover DOM elements. Modern implementations of jQuery will leverage powerful `querySelector` methods for most of the cases of complex selectors, and thus we do not need to separately support differences of jQuery syntax since we instrument the `querySelector` method.

XHOUND is currently limited in that it searches for modifications in a page's DOM but not in the browser's BOM (Browser Object Module). As such, our tool will not be able to detect certain niche extensions, such as, user-agent spoofers which spoof attributes from the `navigator` object in an effort to spoof the identity of the browser [44]. Finally, it is worth noting that XHOUND does not attempt to configure an extension, once that is installed in the utilized browser. As such, if an extension requires a user to configure it by clicking through various dialogues the first time that extension is installed, XHOUND may not be able to detect its presence.

IV. ANALYSIS OF RESULTS

In this section, we first describe the results of applying XHOUND to popular browser extensions and then analyze the findings of multiple user surveys which we conducted, in an effort to estimate the presence and popularity of extensions in the browsing environments of everyday users.

A. Fingerprintability of Popular Extensions

To estimate the overall fraction of detectable extensions in the market we applied XHOUND to the 10,000 most popular extensions in the Chrome Store. Popularity is measured in terms of downloads, with the most popular extensions like AdBlock, AdBlock Plus, Avast SafePrice, and Avira Browser Safety having more than 10 million active users, and the least popular one (ranked 10,000th) having 450 active users. XHOUND's results show that at least 9.2% of extensions introduce detectable DOM changes on any arbitrary domain. This means that any webpage with an appropriately structured DOM could infer approximately 10% of the extensions available in the Google Chrome store. Moreover, more than 16.6% are fingerprintable on at least one popular URL of the Alexa top 50 websites. If, instead of looking at all 10K extensions, we limit ourselves to the top 1K, the fraction of detectable extensions increases to 13.2% for arbitrary domains and 23% for popular URLs. Figure 4 illustrates the distribution of fingerprintable extensions across popularity ranks. Note that all statistics presented in this section are lower-bounds since XHOUND cannot always detect the side-effects and appropriate stimulation parameters that a human analyst could, through manual analysis, discover.

Fingerprintability as a function of ranking
As seen in Figure 4, the overall trend is that the fraction of detectable extensions decreases when we consider less popular Chrome extensions. Interestingly, the ratio of the number of extensions fingerprintable on any arbitrary domain, to the total number of fingerprintable extensions amounts to more than 0.5 and is stable across ranks. The fact that

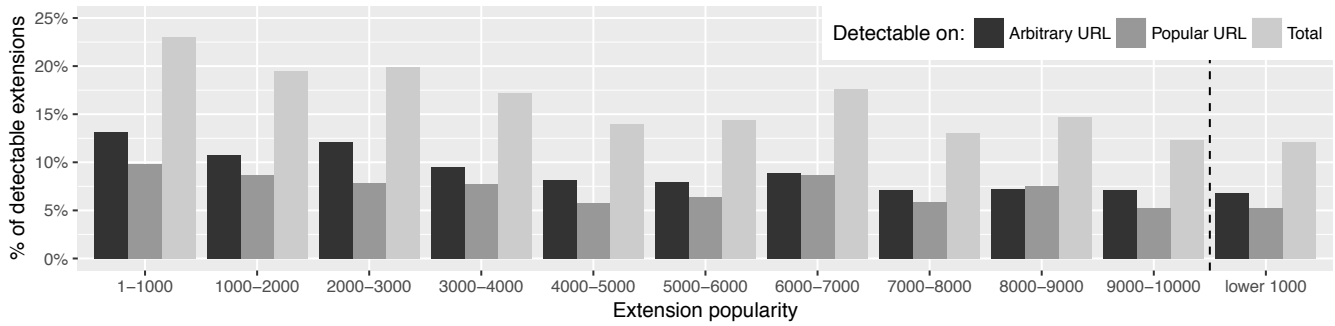

Fig. 4. Results of applying XHOUND to the top 10,000 most popular extensions and to a random sample of 1,000 other extensions. Higher-ranked extensions tend to be more fingerprintable, presumably because of their higher utility, compared to lower-ranked extensions.

the percentage of detectable extensions is higher among the more popular extensions indicates that the threat of web tracking via browser extensions is a realistic threat since most extension-utilizing users are likely to be fingerprintable. At the same time, in addition to using fingerprintable extensions, different users must be using different sets of fingerprintable extensions, if they are to be differentiated from each other by a web tracker. We demonstrate that this is, in fact, the case, in Section IV-B. In addition to the top 10K extensions, we also randomly sampled 1,000 extensions from lower ranks and discovered, as shown in the rightmost part of Figure 4, that they are as fingerprintable as the browser extensions between the 9,000 and 10,000 rank.

Fingerprintability as a function of category

Next to differences according to popularity, the fingerprintability of extensions differs among categories. As Table IV-A illustrates, many shopping extensions are detectable on at least one URL, presumably, being designed for a specific set of online stores; as well as social extensions, which support specific popular social websites. At the same time, some universal shopping extensions and those belonging to categories like accessibility, productivity and photos, are more content-dependent and therefore, more of them can be detected on any arbitrary URL. According to the Pearson's Chi-squared test, in both of the cases URL dependent and independent on-page changes, fingerprintability is significantly dependent on categories, as p-values for the null hypothesis of their independence are 3.812e-15 and 2.2e-16 respectively.

Uniqueness and types of DOM modifications

By analyzing the DOM changes incurred by the 1,656 detectable extensions, we discovered that almost 90% of them perform uniquely identifiable combinations of changes, and more than 86% have at least one completely distinct on-page side-effect that cannot be attributed to any other extension or group of extensions. In other words, the vast majority of fingerprintable extensions perform at least one DOM change (or combination of changes) that is unique to each one of them. Therefore, for the majority of extensions, a web tracker can create signatures based on their DOM changes and precisely attribute the changes to the underlying browser extension. The main type of extension which was not uniquely identifiable is that of ad-blockers. Our current static honey pages utilize tracking scripts from well-known web trackers which appear to be in the blacklists of most of the evaluated extensions. However, as recent research has shown [40], the blacklists of different ad-blockers are not identical, hence, an interested party could potentially analyze the blacklist of each extension and pinpoint entries that are unique to each one. This task requires a significant upfront cost of getting and maintaining a large list of ad URLs and exposing all ad-blockers to these URLs in order to identify their "blind spots." We consider this procedure as out-of-scope for this paper.

TABLE I
DETECTABLE EXTENSIONS PER CATEGORY

Category	*#Evaluated Extensions*	*Detectable On Some URLs*	*Detectable On Any URL*
Productivity	3,438	14.95%	10.01%
Social & Commun.	1,397	27.06%	9.81%
Fun	1,300	12.92%	6.31%
Accessibility	952	17.02%	11.87%
Developer Tools	936	9.29%	8.23%
Search Tools	595	13.28%	5.71%
Shopping	444	34.68%	17.57%
News & Weather	336	4.76%	3.87%
Photos	208	19.71%	11.54%
Blogging	144	14.58%	5.56%
Unknown	129	23.26%	4.65%
Sports	121	4.96%	4.13%

Table II shows statistics of the four types of modifications performed by the 1,656 fingerprintable extensions. Specifically, whenever an extension modifies the DOM it can i) add a new DOM element, ii) delete an existing DOM element, iii) set/change a tag's attribute, and iv) change the text on the page. As the data shows, the most popular action among fingerprintable extensions is to introduce new elements on a page. These new elements are typically used to provide extension-driven UIs to the user, such as, additional controls, overlays, and menus. The LastPass and SaveToPocket extensions described in Section II-B, both fall into this category. A consequence of

TABLE II
TYPES OF DOM CHANGES FROM EXTENSIONS

Type	*Extensions*	*In on-the-fly content*
New DOM node	78.7%	20.3%
Changed attribute	41.6%	84.4%
Removed DOM node	15.8%	59.9%
Changed text	4.7%	61.5%

this finding is that, extension authors who wish to protect the privacy of their users, may have a hard time implementing their desired functionality without introducing changes in a page's DOM. We further elaborate on this problem and possible solutions in Section VI.

The low percentage of discovered textual changes may be due to the limited textual content of XHOUND's static honey pages. Our honey pages can be straightforwardly extended to include a larger and more varied text corpus, including more specific words, names, and headings.

Overall, 42.9% of the fingerprintable extensions make changes to the content they query on web pages, i.e. to the DOM hierarchies or particular elements that are created on-the-fly by our honey pages. Table II shows how such modifications in queried contents contribute to the amount of detected extensions per each type of DOM changes. This proves the value of dynamic honey pages in the detection of fingerprintable extensions. Moreover, almost 90% of 1,656 detected extensions issue at least one query to the DOM, which will often serve as a check that an extension is on the right page, before it proceeds to modify the available content.

Longitudinal analysis of fingerprintability

To understand whether the fingerprintable DOM changes are temporal artifacts of extension development, or necessary modifications that persist through time, we performed a small-scale, longitudinal study of fingerprintable extensions. Specifically, we waited for four months since the original collection of Chrome extensions and examined the updated versions of a sample of 2,000 extensions. 1,000 of these extensions were originally fingerprintable by XHOUND, and 1,000 were originally invisible, i.e., XHOUND could not identify any DOM changes for any of these extensions. Table III shows the results of our analysis. First, we find that more than 90% of the extensions are still available in the Chrome store. Among the extensions that were originally fingerprintable, approximately 38% were updated and, overall, 88% of them remained fingerprintable. An example of an extension that stopped being fingerprintable is the Hola extension [26] that provides free VPN access to its users. Specifically, the extension used to set a "hola_ext_inject" attribute to the root HTML node of each visited page, but now only sets a similar attribute on URLs owned by `hola.org`. It is likely that this change was a reaction due to a recent crackdown by streaming services, such as, Netflix, on users that utilize VPNs and proxies to fake their location and stream content that is not available to their home country [33]. At the same time, nine extensions that were previously invisible to XHOUND, became fingerprintable. An example extension is Imagine Easy Scholar [25], which started injecting additional style sheets on more recent versions of the extension.

TABLE III
TRENDS IN FINGERPRINTABILITY OF EXTENSIONS (RE-RUNNING EVALUATION AFTER 4 MONTHS)

Sample	*Available*	*Updated*	*Detectable*
Detectable	91.0%	37.9%	88.0%
Invisible	95.3%	25.9%	3.73%

Finally, we took advantage of the elapsed time of our previous experiment (four months), to assess whether the "new" top 1,000 extensions were as fingerprintable as the "old" top 1,000 extensions. We found that the intersection of these two sets of top 1,000 extensions was 79.8% out of which 54.6% had updated their versions. By applying XHOUND on the new top 1,000 extension set, we discovered that 12.2% of the extensions were fingerprintable on any arbitrary URL, while 21.6% were fingerprintable on at least one popular URL, compared to our previous 13.2% and 23%. As such, we can conclude that the fraction of fingerprintable extensions appears to be a stable property of the extension ecosystem and is therefore an issue that will not be resolved by itself.

Fingerprintability of extensions for other browsers

A browser extension's modifications to web pages that are taken advantage by XHOUND to fingerprint it, are part of that extension's organic activity, rather than a specific vulnerability of a particular browser. Therefore, other popular web browsers that support extensions are likely to "allow" extension fingerprinting. To test this assumption, we modify XHOUND and use it on the most popular extensions of Mozilla Firefox. At the time of this writing, the Firefox browser supports several technologies for developing add-ons, though developers are officially advised to use the newest WebExtensions APIs, or its predecessor, the Add-on SDK. Since XHOUND is already compatible with the WebExtensions API (since this is the one used by Google Chrome), to support Add-on SDK extensions, we enhanced our patching methods to include the ability to intercept DOM queries from programmatically injected content scripts (as each such script runs in its own execution context).

Among the most popular 1,000 Firefox extensions implemented with either WebExtensions or Add-on SDK, we found that 16% are fingerprintable on at least one URL, and 7.3% on any domain. The fingerprintable extensions include add-ons that have equally fingerprintable Chrome versions like Ghostery, Grammarly, Turn Off the Lights and, Privacy Badger, as well as extensions which are only detected on Firefox, such as, ZenMate Security & Privacy VPN. 88% of the fingerprintable Firefox extensions introduce distinct on-page changes which can thus be used for precise attribution of changes back to their originating extensions. Similar to

the analyzed Chrome extensions, the most popular types of changes are the addition of new DOM elements (67%), the changing of particular attributes (37%) and the deleting of parts of content (27%). Overall, these results correlate with our findings from examining Chrome extensions, showing that extensions developed for either browser are equally likely to be fingerprintable.

B. Fingerprintability of Regular Users based on their Extensions

In Section IV-A we showed that a significant fraction of popular browser extensions are fingerprintable, i.e., a website could infer the extensions installed by preparing an appropriately-structured DOM and allowing the installed extensions to modify it. At the same time, even though knowing that an extension is installed could be used to infer a user's preferences (we discuss this issue in Section VI), that, in itself, does not allow websites to uniquely identify the user. In order for users to be uniquely identifiable, they must not only utilize fingerprintable extensions, but each user must, to a certain extent, utilize a *different set* of fingerprintable extensions.

Collecting extension-usage data from real users

To understand the sets of extensions that everyday users of the web install, and to what extent these extensions could be used for uniquely identifying users, we deployed several surveys where we collected the list of installed extensions from each volunteering user.

In prior work, when researchers where collecting data to assess the fingerprintability of browsers, they would merely ask users to visit a website which would utilize JavaScript to collect various attributes of the users' browsing environments [19], [36]. In our case, since browsers do not have APIs for collecting the list of installed extensions, the only way of collecting installed extensions through the mere visiting of a website, would be to use XHOUND's results and create webpages that would fingerprint extensions through their DOM changes. While, as we later show in Section V, it is certainly possible to fingerprint the extensions that have been analyzed by XHOUND, we would have no way of gauging the fingerprintability of extensions that were not in the set of extensions analyzed by our system, e.g., less popular, or custom extensions. Therefore, we opted to design and implement a browser extension which, upon installation, retrieves the list of other extensions installed on the user's browser, and send this list to our monitoring server, in an anonymous (no PII collected) and secure (using HTTPS for communication) fashion. In addition to the list of extensions, our extension also calculates and sends the size of the browsing history and the number of cookies in the browser's cookie jar (just the size, not the actual history or cookies). As we discuss later in this section, we use these quantities to isolate entries of users who are not active users of Google Chrome, e.g., users who could have just installed the browser in order to participate in our surveys and get compensated.

TABLE IV
CONDUCTED SURVEYS FOR EXTENSION PROFILES

Survey Audience	*Participants*	*#Unique Extensions*
Friends and Colleagues	51	148
US MTurk Workers	313	482
Non-US MTurk Workers	196	312
Students	294	385
Total	854	941

TABLE V
ACTIVE USERS OF EXTENSIONS PER SURVEY AUDIENCE

Survey Audience	*Mean*	*Median*
Friends and Colleagues	1,466,066	283,909
US MTurk Workers	835,046	118,696
Non-US MTurk Workers	1,196,496	190,454
Students	1,049,267	217,896

Since our surveys involved the installation of software and the extraction of data from users' machines, we applied and obtained permission from our institute's IRB that allowed us to conduct our surveys. Initially, we distributed our survey among colleagues and friends from US and European institutions. We then targeted a larger audience by making use of the Amazon Mechanical Turk platform and by sending advertisements for volunteers to our campus mailing list. Mechanical Turk users were compensated directly after completing our survey, whereas students from our university participated in a raffle to win multiple gift cards.

Statistics of extension usage

Overall, 854 users participated in our surveys who had a total of 941 unique browser extensions installed and enabled. Table IV presents exact numbers for each set of users who participate in our surveys. On average, surveyed users had 4.81 active extensions in their browsers. Interestingly, we discovered that some users keep extensions that are no longer available in the market, or are side-loaded from third-party websites, regardless of Google's disapproval of this practice. As such, even though the participating users were using a total of 941 unique browser extensions, we were able to download and analyze only 856 of these extensions from the Chrome Store. As such, the statistics reported in the rest of this section are obtained using these 856 browser extensions.

By deploying different surveys we targeted participants that represent different professions, technical backgrounds and geographical locations. First, as Figure 5 illustrates, Non-US Mechanical Turk users, most of whom originate from India, along with surveyed students, tend to have less extensions installed. Second, Table V shows that US Mechanical Turk workers tend to use less popular Chrome extensions compared to the other surveyed audiences. Specifically, Table V shows that the average extension installed by the "US MTurk Workers" group is installed by approximately 835K users, compared to 1.4 million installations of the average extension of "Friends and Colleagues".

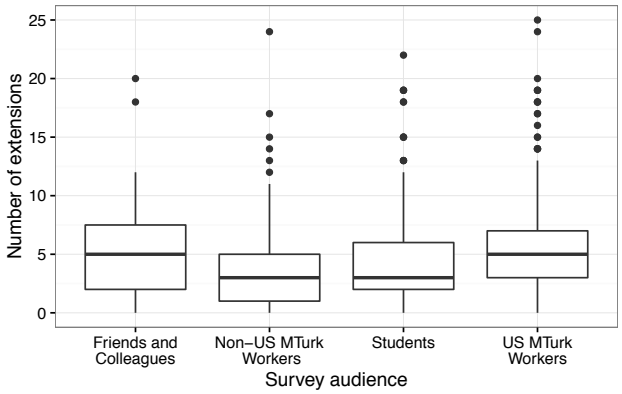

Fig. 5. Distribution of numbers of extensions for different survey audiences.

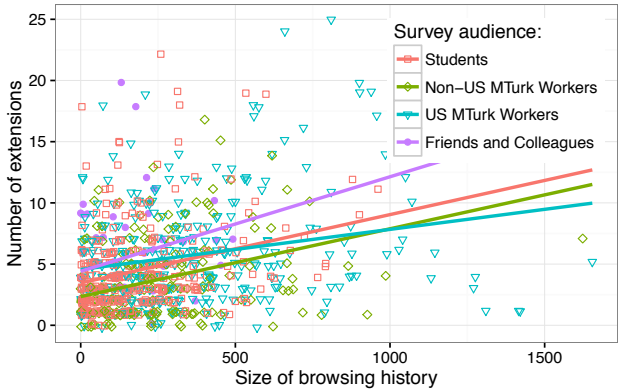

Fig. 6. Relation between number of installed extensions and size of browsing history.

As mentioned earlier, our survey extension also collects the size of browsing history and size of cookie jars to help us identify entries of inactive users. As Figure 6 illustrates, the size of browsing history and the number of extensions in a user's browser are positively correlated. The number of extensions also depends on the particular group of users. One may notice that MTurk users tend to have a different ratio between the number of extensions and the size of browsing history, e.g., even users with large browsing histories have less installed extensions, compared to the rest of the surveyed audiences. Even though, as shown in Figure 6, the fitted regression lines are less steep for MTurk workers, the analysis of covariance showed no significant interaction between the size of browsing history and different user groups. We take advantage of this fact to combine an overall set of users, as well as to extract a new subset of users across all surveyed audiences, whose history size is above the first quartile of their respective distribution. We label these 641 (75.1%) users as "Frequent Chrome Users" and report separate statistics for them.

Fingerprintability of collected extensions

Overall, from the 856 unique extensions obtained from surveys, 174 (20.3%) were fingerprintable, introducing 81

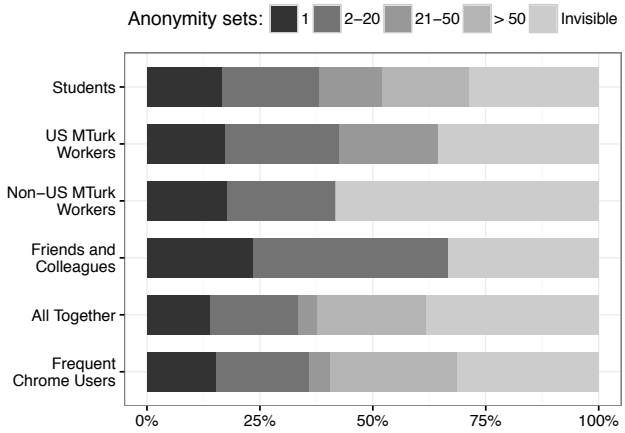

Fig. 7. Distribution of anonymity set sizes based on different survey audiences.

unique, on-page DOM changes. Of these 174 extensions, 93 were fingerprintable on any arbitrary URL. To understand the fingerprintability of users utilizing these extensions, we use the notion of anonymity sets as described by Eckersley [19] and used by Laperdrix et al. in [36]. Each anonymity set represents a group of users with the same extension-based fingerprint. The smaller the size of an anonymity set, the more trackable, users of this set are. In its extreme, an anonymity set with a size of one, means that a user is uniquely identifiable, i.e., no other user has the same browser extensions installed. Figure 7 shows the distribution of the anonymity sizes of our surveyed audiences. One can see that, for all groups of users, with the exception of Non-US MTurk workers, approximately 70% of users had at least one fingerprintable extension. In addition, 14.1% of all users in all groups are uniquely identifiable (belong to an anonymity set of size equal to one). These results show that extension-based fingerprinting is a real threat to online privacy and could be straightforwardly used to supplement existing fingerprinting techniques.

Discriminatory power of extension-fingerprinting

To quantitatively compare extensions with other popular attributes for fingerprinting, we calculated the normalized values of Shannon's entropy, following the approach of Laperdrix et al. [36] and Cao et al. [14]. This gives us ability to compare entropies of fingerprinting attributes investigated in previous studies, despite the different sizes of testing sets. At the same time, because Laperdrix et al.'s dataset is significantly larger than ours, we limit our comparison to the work of Cao et al. [14] who collected 3,615 fingerprints from 1,903 users.

Table VI compares the entropy values, which we calculate for each surveyed audience of our study, to the entropy of other attributes of desktop and mobile browsers as calculated by Cao et al. [14]. As one can observe, the normalized entropy provided by extension-based fingerprinting can reach the values of other popular attributes for fingerprinting desktop browsers. Moreover, for users from a CS background

TABLE VI
NORMALIZED ENTROPY OF EXTENSIONS COMPARED TO OTHER ATTRIBUTES FROM CROSS-BROWSER FINGERPRINTING STUDY [14]

Extension Fingerprinting		
Attribute	*Actual*	*Normalized*
Extensions (Friends and Colleagues)	3.24	0.571
Extensions (US MTurk Workers)	4.20	0.507
Extensions (Students)	4.03	0.491
Extensions (Frequent Chrome Users)	4.40	0.472
Extensions (All Together)	4.25	0.437
Extensions (Non-US MTurk Workers)	3.02	0.397
Attributes from Tables I and IV of [14]		
Attribute	*Actual*	*Normalized*
User Agent	6.71	0.612
List of Plugins	5.77	0.526
Timezone	3.72	0.340
Screen Resolution	Not listed	0.285
List of Fonts (Flash)	2.40	0.219
Cookie Enabled	0.00	0.001

who tend to use more extensions (the ones in our "Friends and Colleagues" audience), extensions may even eventually substitute one of the most discriminating attributes according to previous studies, namely installed plugins. However, we again acknowledge the limitation of our study in terms of the significantly smaller set of collected fingerprints compared to other studies and how this could be biasing our normalized results. For this reason, Table VI also shows the actual, non-normalized, entropy values. We hope to be able to, in the future, repeat our survey against a larger set of users in order to better estimate the discriminatory power of extension-based fingerprinting.

V. IMPLEMENTATION & PERFORMANCE

Having shown that browser extensions are fingerprintable and that different users tend to use different sets of fingerprintable extensions, the only question that remains is how our findings could be put into practice by trackers. In this section, we describe a proof-of-concept, extension-fingerprinting script and measure its performance overhead.

According to our threat models, we assume that an attacker, i.e., a tracker, keeps a database of detectable extensions with the on-page changes that each extension introduces. To keep such a database up-to-date the attacker may periodically run a system similar to XHOUND on a market's most popular extensions. Given these inputs, we identify two different ways to architect an extension-fingerprinting script. First, the script may deliver a page with all the content necessary for triggering fingerprintable extensions of interest, extract the resulting DOM, and send the entire DOM to the tracking server, for further offline analysis. Later, the comparator module of XHOUND can match the modified DOM to specific browser extensions and combine them into a fingerprint. Second, in addition to including the extension-triggering content, the tracking script can also include logic that immediately analyzes the page searching for extension-specific changes.

In both scenarios, the attacker needs a way to generate DOM content which triggers the detectable functionality of fingerprintable browser extensions. We argue that, to a large extent, this can be done automatically using the information already provided by XHOUND. Depending on each specific case, an extension may need as little as an ad or a web form to reveal itself (XHOUND discovers these types of extensions through its static honey pages) or as much as a specifically-named DOM element or a series of elements in parent-child relationships. These latter changes are recovered by XHOUND through the use of dynamic honeypages and on-the-fly DOM generation.

Listing 3 Examples of DOM contents retrieved by XHOUND, which trigger on-page changes from extensions (underlined)

```
// HoverZoom inserts special class attribute
<a data-expanded-url="nstagram.com/p/"
     class="hoverZoomLink"></a>
<img src="http://gravatar.com/avatar/"
     class="hoverZoomLink">

// Skype removes the following script
<script id="skype_script" src="chrome-extension://
     lifbcibllhkdhoafpjfnlhfpfgnpldfl/menu_handler.js">
</script>

// Google Calendar adds a link
<div class="vevent">
    <div class="summary">
    <a href="https://calendar.google.com/calendar/...">
    <img src="chrome-extension://.../calendar_add_38.png
    alt="add to google calendar"></a>
    <div class="description"></div>
    <div class="dtstart"></div>
    <div class="published"></div>
    <div class="dtend"></div>
    <div class="url"></div>
        <adr>
            <div class="locality"></div>
            <div class="region"></div>
        </adr>
    <div class="location"></div>
    </div>
</div>
```

For instance, while analyzing the popular HoverZoom extension, XHOUND recorded HoverZoom's queries for finding all elements like "`a[data-expanded-url*="nstagram.com/p/"]`", "`img[src*="gravatar.com/avatar/"]`" and created mock elements that satisfied the requirements. Listing 3 shows the created mock content and introduced changes after HoverZoom finds it (new "hoverZoomLink" class name). In most of the cases, this mock content can immediately serve as extension-triggering content. Similarly, XHOUND identified and created a mock script that was queried by the Skype extension in order to remove it, and created the necessary complicated hierarchy so that the Google Calendar extension inserts a link into it, as shown in the Listing 3. Given such triggering content, the actual in-page detection consist of simple DOM queries and if-checks, such as the ones illustrated in the Listing 4.

Finally, it is worth noting that, for most extensions, an attacker can use a single fingerprinting script to identify extensions since, according to our results, more than 86% of extensions are uniquely identifiable regardless of possible overlapping on-page changes. If one would still want to isolate

Listing 4 Examples of DOM queries that are sufficient to detect triggered extensions

```
if (document.querySelector(".hoverZoomLink") !== null) {
    // HoverZoom is present
}
if (document.getElementById('skype_script') === null) {
    // Skype is present
}
if (document.querySelector('a[href*="https://calendar.
     google.com/calendar/event"]') !== null || document.
     querySelector('img[src*="chrome-extension://
     gmbgaklkmjakoegficnlkhebmhkjfich"]') !== null) {
    // Google Calendar is present
}
```

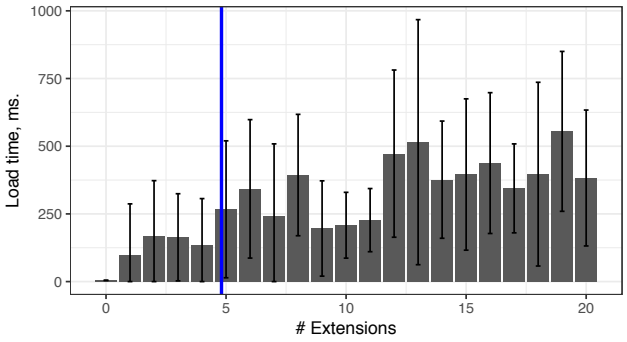

Fig. 8. Load time of extensions before on-page changes appear. The blue vertical line signifies the number of extension that an average user installs, as discovered by our Mechanical Turk experiment.

the effects of extensions, one strategy would be to use iframes where each extension would manifest itself in a separate DOM.

To analyze the performance of our proof-of-concept extension-fingerprinting script, we automatically retrieved the XHOUND-detected triggering content for 30 universally-fingerprintable extensions and implemented the corresponding checks. These checks could, in principle, be automatically generated by parsing XHOUND's output but we leave this automation for future work. Our sample of extensions includes popular fingerprintable extensions based on the number of active users, which we found in the market and during our user study, as well as randomly selected lower-ranked extensions. Our non-minimized script together with the triggering content has a size of less than 16 Kb. For testing, we run our script on random subsets of up to 20 extensions, taking the average of ten runs per subset. The testbed was automated with Selenium's ChromeDriver on a MacBook Air laptop (1.7 GHZ Intel Core i5, 4 GB RAM, other applications open). All fingerprintable extensions were correctly identified by our script. Apart from the time that we need to wait so that extensions manifest themselves (discussed in the next paragraph), the actual checks take less than 5 ms.

The part that "delays" the fingerprinting process is the fact that, unlike traditional fingerprinting which, for the most part, just reads out existing properties, such as screen size or a list of plugins, most extensions introduce their changes *after* a web page is loaded (`window.onload` event fires). An extension-fingerprinting script must therefore wait, or keep polling the DOM, until a page has loaded and until each extension has had a chance to introduce its changes. To quantify this delay, we run our extension-fingerprinting script 21 times in increments of one extension, starting with a browser with no extensions and ending with a browser with 20 installed extensions. If our script could not find the appropriate number of extensions, it restarted itself after a few tens of milliseconds. Figure 8 present the necessary load time (after the `window.onload` event fires) as a function of the number of installed extensions (whiskers represent 95% confidence intervals). One can see a wave-shaped growth of the extension load time which increases as the number of installed extensions increase. We suspect that these changes are because of multi-threading and non-overlapping processing of particular on-page changes. We attribute the presence of spikes to our system load, as well as recurrent additional delays from the use of ChromeDriver. Overall, despite these "delays", our results show that even for as many as twenty installed extensions, the entire fingerprinting process takes less than one second. Note that this delay is not affected by checking for extensions that are not present. Therefore, a tracker can try to trigger multiple hundreds of extensions and still wait for less than a second, under the assumption that the vast majority of them will not be present and will therefore not delay the manifestation of the ones that are present (our results from Section IV-B indicate that the average user utilizes five extensions).

The reader can view a video demonstration of our extension-fingerprinting script by visiting this URL: `https://vimeo.com/178330178` (password is SP2017). The script waits for a few seconds after the `window.onload` event fires and proceeds to fingerprint the page's DOM. As one can notice, a few extensions reveal the fact that they performed an action (coaxed by the XHOUND-extracted triggering conditions) by adding visual elements in a page's DOM (such as Avira which informs the user that it blocked a suspicious resource). Since these visual elements are part of a page's DOM, the tracking script can immediately delete them and hence, even for these "verbose" extensions, hide the fact that the user's browser is being fingerprinted.

VI. Discussion and Future Work

In this section we first discuss the implications of our findings with regard to user privacy and then sketch two possible countermeasures against tracking facilitated by browser extensions.

A. Privacy Implications

As described in Section IV-A, XHOUND was able to automatically fingerprint 9.2% to 23% of all evaluated extensions, with the exact percentage depending on the popularity of the extension, and the considered threat model. Moreover, by collecting extension profiles from 854 real users (Section IV-B),

we discovered that not only do most users utilize fingerprintable extensions, but that many of them use different sets of fingerprintable extensions, allowing trackers to use extensions as a way of uniquely, or near-uniquely identifying them. It is worth pointing out that, given the current architecture of browser-extension frameworks, we do not assign blame to the developers of the extensions that XHOUND can fingerprint. The functionality of the majority of extensions that modify the DOM depends on the same modifications that make them fingerprintable. To quantify this dependence, we manually examined a random sample of 100 fingerprintable extensions and discovered that 75% of them were modifying the DOM in accordance with the stated purpose of the extension.

Our findings demonstrate that extension-based fingerprinting is a real and credible threat that further complicates online browsing. By utilizing an XHOUND-like system, advertising companies and online trackers can, in bulk, discover the side-effects of browser extensions and incorporate extension-detection code in their existing tracking scripts. As we showed in Section IV-B, the amount of entropy that browser-extensions provide is higher than many fingerprintable attributes, such as, canvas, that are already adopted by fingerprinting scripts [6]. Moreover, since the extensions of Google Chrome users can be synced between different machines, extension-based fingerprinting can be used for cross-device tracking.

Interestingly, our results are also likely applicable to mobile browsers. Most mobile browsers do not support plugins (such as Flash or Java) and hence are less susceptible to standard fingerprinting practices, than desktop and laptop browsers. At the same time, many popular mobile browsers, like Firefox Mobile and Dolphin Browser for Android, and Google Chrome for iOS [32], are extendable, i.e., they allow users to install browser extensions, much like their desktop counterparts. Therefore, the ability to fingerprint browser extensions can, in principle, allow trackers to extract entropy from a platform that has been long-considered, from a tracking point of view, "problematic."

A more subtle implication of fingerprinting browser extensions is that extensions, unlike plugins and other existing fingerprintable features, capture, to a certain extent, the interests of users. That is, in addition to offering bits of entropy that can be used to uniquely identify them, browser extensions can give away the income-level of a user (e.g., extensions that automatically search and apply coupons to various online shops), the fact that they are not located where their browser claims to be (e.g., VPN and geolocation-bypass extensions), whether they are tech-savvy or not (e.g. extensions that block ads and trackers, or those that show the security status of a website), and even their political inclinations (e.g. extensions that automatically remove or replace text and links that contain keywords associated with specific political figures). All of this information can be extracted and made part of a user's profile, allowing further deanonymization and targeting.

B. Countermeasures

It is important to note that since the fingerprinting of extensions is done through benign-looking DOM queries, "easy" solutions, such as, limiting access to the `navigator.plugins` object [1], will not be of help for combatting extension-based fingerprinting. In this section, we briefly sketch two possible countermeasures that could be used to combat extension-based fingerprinting based on *encapsulation* and *namespace pollution*.

Encapsulation

The idea of enhancing only the appearance of web pages is close to the concept of Shadow DOM, which gives the ability to web developers to encapsulate presentational widgets from other JavaScript and CSS on the page [2]. With Shadow DOM, a subtree of DOM elements can be rendered as a part of a document while not being a part of the main document's DOM tree. For instance, to change the appearance of a web form, a shadow subtree can be created on the top of the form with new CSS styles, additional labels and graphics, and also with special content elements that project original input fields to the rendered DOM. These shadow elements are invisible to queries from the main DOM. Indeed, there exist examples of Chrome extensions that plan or already adopt such techniques, e.g., Adblock Plus and AdBlock to hide non-blocking ads without breaking a page's layout [4]. However, since Shadow DOM is designed with the aim of mainly separating presentation from content, it is not possible to implement all on-page changes from extensions in the same fashion. Although adding additional control elements can be effectively achieved with shadow elements, particular extensions would still require altering the main DOM tree in order to perform the desired changes, which would require synchronization between shadow and main elements:

- to set or update functional attributes on the original nodes, which change the logic flow of original scripts on a page (e.g., the global `document.title` property)
- to change parts of the internal text, which is further processed by a web page as an input, or to integrate additional control elements inside the text
- to actually block or delete existing DOM nodes, such as, ads or tracking scripts

Therefore, even though the Shadow DOM is a step in the right direction for achieving an architecture offering undetectable on-page changes, there are multiple challenges that need to be overcome before this mechanism can be effective. We plan to research these directions along with alternative designs for encapsulation and isolation in future work.

Namespace Pollution

Even though the results of a system like XHOUND could be used to identify browser extensions and therefore to reduce a user's online privacy, they could also be used in a constructive fashion. Namely, given a list of extensions and their DOM

side-effects, a user could *pretend* to have a certain number of extensions installed by populating these side effects in a page, without actually installing any extensions. In addition, a dedicated anti-fingerprinting extension could, for every page load, select a random sample of DOM modifications and apply them to the newly loaded page. This will significantly complicate the task of simple extension-based fingerprinting which will now have to distinguish between extensions that are really installed, from extensions whose side-effects are merely mimicked. A tracking script that fails to account for these mimicries, will be computing a different fingerprint for every page load and will thus be unable to associate different page visits with the same user. Despite its seeming simplicity, this technique is also likely to have its own unique set of challenges, e.g., preserving the functionality of a webpage when arbitrary nodes are being modified, which we will investigate in future work.

VII. Related Work

To the best of our knowledge, this paper is the first one that proposes a fully automated system for fingerprinting browser extensions based on their side-effects on a page's DOM, and quantifies the fingerprintability of popular extensions installed on the browsers of real users. In this section, we discuss the related work dealing with the fingerprinting of browser extensions, separating it into fingerprinting made possible by manual analysis of browser extensions, and automated fingerprinting.

Manual Analysis

Mowery et al. described the process of inferring the user-customized rules of the NoScript browser extension by trying to load multiple JavaScript scripts from various domains and observing which ones succeed and which ones fail [41]. The same technique has been used both by researchers [36], as well as advertising companies [8]–[10], in order to detect the presence of ad-blockers.

Nikiforakis et al., as part of their study of commercial fingerprinting providers, manually analyzed eleven user-agent spoofing extensions and showed that the inconsistencies of the claimed browser identity and JavaScript-accessible objects could be abused to detect the presence of the evaluated browser extensions [44]. In a subsequent study, Acar et al. analyzed an anti-fingerprinting browser extension [11], [12] and showed that it was, in fact, also fingerprintable [7].

Automated Detection

In 2012, Kotowicz presented a technique, reminiscent of timing attacks [13], [21], for detecting the browser extensions installed by Chrome users [35]. Using JavaScript, he attempted to load the manifest files of thousands of extensions by addressing them through the `chrome-extension://` URL scheme and their unique extension identifier. Through the appropriate setting of `onload` and `onerror` event handlers, Kotowicz could differentiate between the presence and the absence of the tested browser extensions. This attack no longer works since Google Chrome changed its extension architecture so that all extension resources are hidden from the public web, with the exception of the resources that an extension developer has explicitly marked as "web accessible" [24]. At the same time, Golubovic [42] and Sjösten et al. [46] found that many popular extensions, do in fact make use of web-accessible resources and are thus discoverable.

In XHound, we chose to focus on the DOM-level side-effects of the presence of browser extensions and thus did not consider web-accessible resources. While this technique can be straightforwardly incorporated in XHound, we argue that our discovery method is significantly more robust than the ones based on web-accessible resources. At any given point in time, the developers of browser extensions can disable the use of web-accessible resources, making their extensions invisible to the aforementioned extension-fingerprinting technique. Contrastingly, our fingerprinting techniques are based on an extension's organic activity in a page's DOM. To remove this DOM-level activity, if at all possible, requires significant re-engineering of the entire extension's codebase. At the same time, from a practical point of view, the techniques that XHound uses and web-accessible resources are fully orthogonal and thus a tracker can incorporate both techniques in their tracking scripts. To quantify this complementary nature, we analyzed the manifest files of the 1,656 extensions that XHound was able to fingerprint and discovered that more than 40% of them do not make use of web-accessible resources and thus would not be detectable by them.

VIII. Conclusion

Recent years have seen the web browser becoming an all-encompassing platform, offering to web applications features that were traditionally only available to installed native applications. One crucial feature of these modern browsers is their ability to be extended to meet individual user requirements.

In this paper, we investigated the fingerprintability of browser extensions and sought to quantify it, in terms of the fraction of popular extensions that are fingerprintable and the sets of extensions that different users install. To this end, we designed and implemented the first fully-automated, robust system, XHound, which uses a combination of static and dynamic analysis to identify an extension's organic, fingerprintable activity in a webpage's DOM. By applying XHound to the 10,000 most popular Google Chrome browser extensions, we discovered that 9.2% to 23% of these extensions are fingerprintable and their presence could be inferred by webpages. We then surveyed 854 real users and discovered that most users utilize fingerprintable extensions, and a significant fraction of them use different sets of fingerprintable extensions, allowing trackers to uniquely or near-uniquely identify them. We described the process of developing an extension-fingerprinting script using XHound's results, and showed that a tracker can fingerprint a large number of extensions in just a few seconds.

On the defensive side, we explained why extension-based fingerprinting is more intrusive than traditional fingerprinting

and why, in the current architectures of browsers, eliminating the risk of extension-based fingerprinting is far from trivial. Finally, we sketched two possible countermeasures based on the ideas of encapsulation and namespace pollution that aim to either hide the presence of extensions or confuse trackers about which extensions are really installed in a user's browser.

We hope that our work will be of use to browser vendors, extension developers, and end users. Browser vendors can investigate alternative architectures for supporting extensions that make it harder for webpages to infer the presence of installed extensions, while extension developers can reconsider the designs of their extensions, adopting encapsulation techniques where possible and ensuring that all DOM changes are absolutely necessary. Finally, end users can become aware of the privacy implications of installing browser extensions and consider uninstalling or disabling the ones that they do not absolutely need.

Acknowledgments: We thank our shepherd Nikita Borisov and the reviewers for their valuable feedback. This work was supported by the National Science Foundation (NSF) under grants CNS-1527086, CNS-1617902, and CNS-1617593 with additional support from the Data Transparency Lab. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government, NSF, or the Data Transparency Lab.

IX. Availability

Our plan is to eventually make XHound available to the research community (either by open-sourcing it, or by making it available as a service).

References

[1] "Bug 757726 - disallow enumeration of navigator.plugins," https://bugzilla.mozilla.org/show_bug.cgi?id=757726.

[2] "W3C Shadow DOM," http://www.w3.org/TR/shadow-dom/.

[3] "The Final Countdown for NPAPI," https://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html, 2014.

[4] "Use Shadow DOM for element-hiding where available," http://old-support.getadblock.com/discussions/suggestions/423-use-shadow-dom-for-element-hiding-where-available, 2014.

[5] "NPAPI Plugins in Firefox," https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/, 2015.

[6] 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.

[7] 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.

[8] "Most effective way to detect ad blockers Completely free & easy to implement," http://www.detectadblock.com/.

[9] "PageFair - Reclaim Your Adblocked Revenue," https://pagefair.com/.

[10] "BlockAdblock — Stop Losing Ad Revenue," http://blockadblock.com/.

[11] K. Boda, "Firegloves," http://fingerprint.pet-portal.eu/?menu=6.

[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. Bortz and D. Boneh, "Exposing private information by timing web applications," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 621–628.

[14] Y. Cao, S. Li, and E. Wijmans, "(Cross-)Browser Fingerprinting via OS and Hardware Level Features," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017.

[15] J. Corpuz, "40 Best Google Chrome Extensions," http://www.tomsguide.com/us/pictures-story/283-best-google-chrome-extensions.html, 2016.

[16] A. Das, N. Borisov, and M. Caesar, "Do you hear what i hear?: fingerprinting smart devices through embedded acoustic components," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 441–452.

[17] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "Accelprint: Imperfections of accelerometers make smartphones trackable." in *NDSS*, 2014.

[18] S. Dutton, "Getting Started with WebRTC," http://www.html5rocks.com/en/tutorials/webrtc/basics/.

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

[20] P. Eckersley, "White House Website Includes Unique Non-Cookie Tracker, Conflicts With Privacy Policy," https://www.eff.org/deeplinks/2014/07/white-house-website-includes-unique-non-cookie-tracker-despite-privacy-policy, 2014.

[21] E. W. Felten and M. A. Schneider, "Timing attacks on web privacy," in *Proceedings of the 7th ACM conference on Computer and communications security*. ACM, 2000, pp. 25–32.

[22] D. Fifield and S. Egelman, "Fingerprinting web users through font metrics," in *Financial Cryptography and Data Security*. Springer, 2015, pp. 107–124.

[23] J. Garcia, "Top Browser Plugins to Increase Browsing Security and Privacy," https://www.webroot.com/blog/2016/02/03/top-browser-plugins-to-increase-browsing-security-and-privacy, 2013.

[24] Google Chrome, "Manifest - Web Accessible Resources," https://developer.chrome.com/extensions/manifest/web_accessible_resources.

[25] Google Chrome Extension, "Imagine Easy Scholar Annotation Tool," https://chrome.google.com/webstore/detail/imagine-easy-scholar-anno/mpioiijlnmnhfbphhafnmilihogfbijk.

[26] Google Chrome Extension, "Unlimited Free VPN - Hola," https://chrome.google.com/webstore/detail/unlimited-free-vpn-hola/gkojfkhlekighikafcpjkiklfbnlmeio.

[27] T. Hupperich, D. Maiorca, M. Kührer, T. Holz, and G. Giacinto, "On the robustness of mobile device fingerprinting: Can mobile users escape modern web-tracking mechanisms?" in *Proceedings of the 31st Annual Computer Security Applications Conference*, ser. ACSAC, 2015.

[28] 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. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/jagpal

[29] S. Jobs, "Apple — Thoughts on Flash," https://www.apple.com/hotnews/thoughts-on-flash/, 2010.

[30] 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)*. San Diego, CA: USENIX Association, Aug. 2014, pp. 641–654. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kapravelos

[31] H. King, "20 ways to make your Chrome browser so much better," http://money.cnn.com/2015/07/01/technology/best-chrome-extensions/, 2015.

[32] T. Klosowski, "Chrome for iOS Adds Supports for App Extensions," http://lifehacker.com/chrome-for-ios-adds-supports-for-extensions-1637664312, 2014.

[33] K. Knibbs, "Damn, Netflix Is Cracking Down on VPNs and Proxies," http://gizmodo.com/damn-netflix-is-cracking-down-on-vpns-and-proxies-1752956270, 2016.

[34] M. Korolov, "Java is the biggest vulnerability for US computers," http://www.csoonline.com/article/2875535/application-security/java-is-the-biggest-vulnerability-for-us-computers.html, 2015.

[35] K. Kotowicz, "Intro to Chrome addons hacking: fingerprinting," http://blog.kotowicz.net/2012/02/intro-to-chrome-addons-hacking.html, 2012.

[36] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints," in *37th IEEE Symposium on Security and Privacy (S&P 2016)*, San Jose, United States, May 2016. [Online]. Available: https://hal.inria.fr/hal-01285470

[37] 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.

[38] J. R. Mayer, "Any person... a pamphleteer," Senior Thesis, Stanford University, 2009.

[39] M. McCorry, "Top 10 Free Must Have SEO Browser Plugins," https://www.branded3.com/blog/top-10-free-must-have-seo-browser-plugins/, 2015.

[40] G. Merzdovnik, M. Huber, D. Buhov, N. Nikiforakis, S. Neuner, M. Schmiedecker, and E. Weippl, "Block Me If You Can: A Large-Scale Study of Tracker-Blocking Tools," in *Proceedings of the 2nd IEEE European Symposium on Security and Privacy (IEEE EuroS&P)*, 2017.

[41] K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham, "Fingerprinting information in JavaScript implementations," in *Proceedings of W2SP 2011*, H. Wang, Ed. IEEE Computer Society, May 2011.

[42] Nicolas Golubovic, "Attacking Browser Extensions, MS Thesis, Ruhr-University Bochum," http://nicolas.golubovic.net/thesis/master.pdf, 2016.

[43] N. Nikiforakis, W. Joosen, and B. Livshits, "PriVaricator: Deceiving Fingerprinters with Little White Lies," in *Proceedings of the 24th International World Wide Web Conference (WWW)*, 2015.

[44] 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*, ser. SP '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 541–555. [Online]. Available: http://dx.doi.org/10.1109/SP.2013.43

[45] Recorded Future, "Gone in a Flash: Top 10 Vulnerabilities Used by Exploit Kits," https://www.recordedfuture.com/top-vulnerabilities-2015/.

[46] A. Sjösten, S. Van Acker, and A. Sabelfeld, "Discovering browser extensions via web accessible resources," in *Proceedings of the 7th ACM Conference on Data and Applications Security and Privacy (CODASPY)*, 2017.

[47] 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.

[48] D. Tong, "Integrating Canvas into your Web App," http://www.html5rocks.com/en/tutorials/canvas/integrating/.

[49] C. F. Torres, H. Jonker, and S. Mauw, "Fp-block: Usable web privacy by controlling browser fingerprinting," in *ESORICS*, 2015.

[50] 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*, ser. WWW '15, 2015, pp. 1286–1295.

[51] Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic fingerprinting revisited: Generate stable device id stealthily with inaudible sound," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 429–440.